Outils cryptographiques pour la protection des contenus et de la vie privée des utilisateurs

Amandine Jambert

Université Bordeaux 1

Numéro d'ordre : 4234 **École doctorale de Mathématiques et Informatique**

Outils cryptographiques pour la protection des contenus et de la vie privée des utilisateurs

THÈSE

présentée et soutenue publiquement le 15 Mars 2011

pour l'obtention du

Doctorat de l'université Bordeaux 1

(spécialité informatique)

par

Amandine Jambert

Composition du jury

Rapporteurs : Olivier Pereira : Professeur à l'Université Catholique de Louvain
David Pointcheval : Directeur de recherche CNRS

Examinateurs : Sébastien Canard : Ingénieur de Recherche (HDR)
Andreas Enge : Directeur de Recherche Scientifique INRIA
Marc Girault : Professeur de Lycée (HDR)
Gilles Zémor : Professeur des Universités *(Directeur de Thèse)*

Mis en page avec la classe thloria.

Remerciements

Ce n'était donc pas une légende, il existe un monde après la thèse. Et alors que ce monde commence à se dévoiler sous mes yeux, il est temps pour moi de remercier ceux qui m'ont permis d'en arriver là.

Je tiens d'abord à remercier ceux qui m'ont permis de négocier la ligne d'arrivée. Je remercie ainsi vivement David Pointcheval et Olivier Pereira qui m'ont fait le privilège d'accepter d'être les rapporteurs de cette thèse et qui par leurs commentaires en ont amélioré le contenu. Je remercie également Andreas Enge que j'ai le plaisir de compter parmi les membres de mon jury.

Je tiens ensuite à remercier ceux qui m'ont permis de tracer ma route. Je ne remercierais jamais assez mon capitaine, Sébastien Canard, de m'avoir proposé pendant mon stage de continuer en thèse et d'avoir ensuite plus que rempli son rôle de responsable de thèse, sa disponibilité, son expertise et ses innombrables conseils m'ont permis de tenir le cap au cours de cette traversée même lorsque tout devenait flou dans la tempête. Je remercie Gilles Zémor pour avoir accepté de prendre la direction de ma thèse en fin de première année et pour avoir permis par ses commentaires plus académiques d'élargir mon horizon. Je souhaite aussi remercier mon premier Amiral, Marc Girault, pour avoir donné sa chance à la mousse intimidée qui souhaitait faire une thèse sous sa direction et lui avoir donné les premières instructions. La tempête l'a arraché à sa flotte, mais son souci de ses thésards dans la tempête restera pour moi un exemple. C'est pour moi un honneur de le voir aujourd'hui dans mon Jury. Je tiens aussi à remercier Michel Milhau, co-responsable de ma thèse avant la tempête, pour son point de vue pratique de la cryptographie.

Il ne suffit pas d'un Amiral et d'un capitaine pour une telle traversée et je me dois à présent de remercier toute l'équipe qui a œuvré en coulisse. Je tiens ainsi à remercier tous les collègues d'Orange Labs que ce soit ceux d'avant la tempête ou ceux que j'ai rejoint par la suite. Je remercie ainsi Fabrice Clerc pour m'avoir ouvert les portes de l'URD NSS, Lyseline Youf pour son soutien en tant que manager de ma nouvelle URD TES, Jef pour son esprit Critique et m'avoir supporté pendant plus de deux ans dans son bureau, Jacques (Traoré) pour ses réflexions, ses conseils et sa patience pendant ma troisième année que j'ai passé chez lui, Emmeline pour la soirée qu'elle a prise, alors qu'elle était en pleine rédaction, pour discuter avec moi de ce qu'était vraiment être thésarde à NSS, Iwen pour ses conseils cryptographiques (ou non) aux pauses et pour m'avoir fait découvrir les merveilles réservées aux spéléos en restant sagement planquée derrière un PC, Nizar pour les discussions interminables sur la vie, l'univers et le reste (mais surtout le reste), Grégoire pour les soirées mémorables et m'avoir aidée à apprendre à Nizar ce qu'était la vraie recette des crêpes... Roch pour m'avoir supporté en pause pendant toute la rédaction, Nicolas pour son soutien dans mes envolées libristes et pour ses implémentations de qualité, Stéphane pour ses commentaires discrets toujours pertinents, Jérémie (et Alba) pour m'avoir permis de partir en conférence sans m'inquiéter pour Sally, Chantale pour sa gentillesse et son aide. Enfin, aux membres du nouveau bateau 6cure merci et que les vents vous soient favorables.

Je souhaite remercier tous les équipiers de l'aventure "batch Groth-Sahaï", un article avec tellement d'équations terriblement longues que nos rapporteurs s'inquiétaient de la possibilité de les faire rentrer dans un format llncs. Rien que l'existence de cette version finale est un petit miracle. Je profite de cette occasion pour remercier Damien Vergnaud pour ses idées, ses commentaires et son soutien et Georg Fuchsbauer pour avoir été le premier à tenter de m'initier aux preuves Groth-Sahaï. Parce que j'assume (enfin) mon côté geek, je remercie NCSoft d'avoir sorti Aion juste à temps pour qu'il devienne le lieu privilégié de longues discussions cryptographiques avec Olivier Blazy, ce qui a permis (entre-autre) le respect de la deadline d'ACNS. Ceci m'amène à remercier Olivier non seulement pour les nombreuses soirées crypto-MMOs qui furent (et sont toujours) l'occasion de parler de nouvelles idées mais aussi pour ses

relectures patientes de ce manuscrit.

Parce qu'une telle traversée ne peut pas se faire en apnée, je remercie tous ceux qui m'ont aidé à changer d'air. Je remercie ainsi tous les membres de mes guildes pour les soirées mémorables de mes premières années et pour m'avoir ensuite supportée sur Skype/TS pendant que je rédigeais. En particulier, je souhaite remercier Aspirin qui m'a régulièrement écouté et rassuré en prenant ses propres étudiants en exemple. Je remercie aussi le groupe de Bordeaux avec une mention spéciale à Gaël Prado pour son soutien artistique et pour avoir pris le temps de dessiner Alice, Bob et compagnie même s'ils ont disparu de la version finale, Stephanie Moreaud pour avoir été un fantastique proxi avec l'administration bordelaise, et enfin, Yannick et Sana, pour les soirées-skype et les week-end "Jeu de Rôle" entre exilés.

J'ai gardé pour la fin les personnes les plus importantes pour moi, ma famille. Merci à Sally pour m'avoir aidée à rédiger même si cela consistait à dormir sur mes brouillons et à attaquer l'imprimante, à Camille, ma sœur par le cœur, pour son soutien téléphonique et pour ne pas m'en avoir voulu d'avoir oublié toutes les dates importantes de cette dernière année (promis, je fais mieux l'année prochaine), à Julie parce qu'elle est une grande sœur fantastique, présente à la seconde où j'ai besoin d'elle, à mes parents pour leur soutien inconditionnel et pour leur foi en moi, et enfin à Thomas pour avoir été si patient, si rassurant pendant cette période difficile et pour m'avoir soutenue tout au long de ma thèse.

Mes parents racontent avec plaisir que lorsque j'avais 9 ou 10 ans et qu'on me demandait ce que je voulais faire, je répondais avec l'innocence de l'enfance que je voulais être trouveuse en maths. C'est grâce à vous tous, qu'aujourd'hui, je réalise quotidiennement mon rêve d'enfant.

Merci.

Table des matières

Introduction

La cryptologie est étymologiquement la science (λόγος) du secret (κρυπτός). Celle-ci réunit la cryptographie, l'écriture (γραφή) secrète, et la cryptanalyse l'étude des attaques contre ces mécanismes de cryptographie. Bien que cette définition soit encore partiellement d'actualité, la cryptologie ne se limite plus aujourd'hui à la protection des secrets. Elle s'est élargie à d'autres problématiques dont les signatures numériques de documents ou les moyens d'authentifications. En parallèle, la cryptologie est passée d'un art pour lequel l'intuition était la qualité fondamentale à une science basée sur les mathématiques utilisant abondamment l'informatique.

Historiquement, la cryptographie était essentiellement utilisée pour obtenir la confidentialité des messages. Le principe utilisé suppose que l'émetteur et le destinataire se soit mis d'accord sur une clé secrète commune. Le texte clair est alors transformé à l'aide de cette clé de telle façon que la transformation ne puisse être inversée que s'il l'on connait la clé commune. Dans ce cas, la clé de chiffrement et de déchiffrement est la même : la clé commune, c'est ce que l'on appelle aujourd'hui le chiffrement symétrique. Ce principe est très ancien, en effet, selon D.Kahn [Kah96], on en retrouve les premières traces au moyen empire égyptien lorsqu'au XXème siècle avant J.-C., un scribe employa des hiéroglyphes non conformes à la langue dans une inscription. Ainsi, E.Drioton relate dans [Dri53] certaines des méthodes cryptographiques utilisées durant l'Égypte antique. Une de ces méthodes, la plus simple peut-être, consistait à modifier l'ordre des symboles. Dans ce cas, la clé de déchiffrement est simplement l'ordre de lecture. Cette méthode était principalement utilisée pour les gravures sur les effigies de scarabées. Cependant le Musée Guimet possède un godet de scribe sur lequel un texte de vingt et un signes est traité selon cette méthode.

Exemple.
E. Drioton donne comme exemple [Dri53] le cas simple suivant.

Chiffré + Clé de déchiffrement = Message clair

Le message clair est alors évident pour les lettrés de l'époque : "Khonsou est (ma) protection".

Depuis, l'évolution des usages et des modes de circulation de l'information ont contraint la cryptologie à constamment se renouveler. Les techniques de cryptographie sont ainsi devenues de plus en plus complexes au fur et à mesure que la lecture et l'écriture se démocratisaient et que les techniques de cryptanalyse s'affinaient. L'information étant capitale, en particulier en temps de guerre, les grands royaumes se dotent, dès le moyen-âge, de cryptologues. Ceux-ci ont pour mission de protéger les dépêches diplomatiques et les ordres militaires, mais aussi de déchiffrer

les secrets de leurs adversaires. À partir de cette époque et jusqu'au XXème siècle, la cryptologie n'est pratiquement pas utilisée en dehors des très hautes sphères de pouvoir. Son utilisation est, en effet, considérée comme une circonstance aggravante pour tout délit, même lors d'échanges épistolaires privées n'ayant rien à voir avec l'accusation. La noblesse utilise donc plutôt des méthodes de stéganographie dans lesquelles le message est simplement plus ou moins caché dans un objet, dans un texte plus long ou dans un discours. Il existe de nombreux exemples de tels procédés, par exemple ce texte légitimiste et anti-bonapartiste anonyme de la fin du XIXème siècle.

Exemple.

> *Vive à jamais* l'empereur des Français
> *La famille royale* est indigne de vivre :
> *Oublions désormais* la race des Capets,
> *La race impériale* doit seule lui survivre !
> *Soyons donc le soutien* de ce Napoléon.
> *Du comte de Chambord* chassons l'âme hypocrite :
> *C'est à lui qu'appartient* cette punition.
> *La raison du plus fort* a son juste mérite.

Ces techniques stéganographiques sont toujours utilisées de nos jours, notamment avec des méthodes d'intégration d'un texte dans les pixels d'images banales. Si les stratégies restent les mêmes, les moyens utilisées sont aujourd'hui bien plus élaborées. Cependant, ces méthodes impliquent d'une part que le procédé utilisé reste secret et d'autre part que l'adversaire ignore où regarder. La cryptologie a ainsi l'avantage de s'affranchir du contenant et de ne transférer que l'information chiffrée.

Le développement des moyens de communications au XIXème siècle, l'essor de de nouveaux moyens de communication comme le télégraphe ou le téléphone, ont donné un nouvel élan à la cryptologie. Les communications étant plus simples à intercepter, la cryptologie est utilisée massivement afin d'assurer la confidentialité totale des messages militaires.

Lors de la première guerre mondiale, les communications s'effectuent par ces nouvelles voies de communication. La cryptographie est donc massivement utilisée afin de garantir leur confidentialité. La maîtrise Française de la cryptographie a alors une influence décisive sur l'issue de la guerre. En effet, dès le début des hostilités, la majorité des messages chiffrés allemands sont décryptés (les textes clairs sont retrouvés). Malheureusement, des indiscrétions politiques et journalistiques gâchent rapidement ces exploits, l'armée allemande change alors de méthode de chiffrement ce qui met en échec les experts français de mars à juin 1918. Lorsque le cryptologue Georges Painvin parvient enfin à casser ce nouveau schéma, la France peut à nouveau anticiper. Le maréchal Foch retourne alors la situation et mène la France à la victoire. Pour la dernière fois, l'intelligence humaine travaille seule avec un papier et un crayon face aux schémas de cryptographie.

Entre les deux guerres, les balbutiements de l'informatique vont conduire la cryptologie à évoluer à nouveau. En effet, il est alors possible de concevoir de nouvelles méthodes de chiffrement entièrement mécaniques. Plusieurs machines, dont une première version de la célèbre machine Enigma qui sera utilisée par les allemands durant la seconde guerre mondiale, voient ainsi le jour. Mais c'est durant la seconde guerre mondiale que ces machines prouveront leur utilité et rendront les premiers ordinateurs incontournables pour la nouvelle cryptologie. C'est notamment la maîtrise de ces outils par les mathématiciens et cryptologues du Service Britannique du Chiffre qui permettra la victoire des alliés.

Depuis, l'expansion d'internet a modifié notre relation à l'information et à l'usage de la cryptologie. L'information se diffuse à large échelle et se propage à grande vitesse. Il devient difficile d'en vérifier la validité ou l'authenticité sans outils adaptés. Nous sommes aujourd'hui entrés dans l'ère de l'information. Les services gouvernementaux, bancaires et commerciaux sont le plus souvent directement accessibles sur internet, c'est-à-dire sur un réseau public mondial. Les besoins en confidentialité sont ainsi sortis du domaine militaire pour intégrer notre vie quotidienne. Et de nouvelles problématiques sont apparues. Comment assurer l'authentification d'un utilisateur souhaitant accéder à un service donné ? Comment garantir que des données ne soient pas modifiées entre l'émetteur et le destinataire ? Comment vérifier qu'un message provient bien d'un émetteur donné ? Ou encore, comment protéger la vie privée des utilisateurs dans un système à mémoire virtuellement infinie ? Dans une société où les informations d'un compte Facebook peuvent être monnayées, comment concilier l'équilibre entre vie privée et intérêt commercial ?

Au cours des vingt dernières années, la cryptologie a répondu à la plupart de ces questions, notamment par l'introduction, en 1976, de la cryptographie à clé publique par W.Diffie et M.E.Hellman [DH76]. Ce nouveau paradigme suppose que l'utilisateur possède une paire de clés, une publique et une privée. Grâce à celui-ci, et aux schémas associés, il est enfin possible de construire des schémas de chiffrement se passant de la mise en place d'une clé commune à l'émetteur et au destinataire. Ce nouveau principe permet, d'autre part, la mise en place de schémas de signature ayant des propriétés similaires à la signature manuscrite. L'émetteur d'un message signe avec sa clé secrète de telle manière qu'un tiers pourra publiquement en vérifier la validité avec la clé publique correspondante.

Aujourd'hui, la cryptographie moderne continue à suggérer de nouvelles solutions pour transposer au monde numérique les outils de sécurité du monde physique, mais elle va aussi plus loin en proposant de nouvelles possibilités. Par exemple, il est possible de signer numériquement un document non seulement en son nom propre, mais aussi en tant que membre d'un groupe d'utilisateurs et cela tout en restant anonyme. Un autre exemple est la possibilité, à laquelle je me suis intéressée, de signer un message de telle façon qu'une personne spécifiquement désignée soit capable de modifier le message tout en gardant une signature valide de l'émetteur original. L'émergence de ces nouveaux outils permet d'envisager des solutions cryptographiques aux nouveaux problèmes issus d'internet, notamment ceux reliés à la protection de la vie privée des internautes.

Dans ce mémoire, je m'intéresserai principalement aux outils de cryptographie à clé publique et à leur application pour la protection des contenus et de la vie privée des usagers de services. Après une première partie introductive, je décrirai les travaux que j'ai effectués sur différents moyens de prouver la connaissance d'un secret ayant une forme particulière. J'ai ainsi travaillé avec S.Canard, I.Coisel et J.Traoré [CCJT] sur les méthodes permettant de prouver qu'un secret appartient à un intervalle public. D'autre part, lors d'une collaboration pour le projet collaboratif PACE, j'ai travaillé avec O.Blazy, G.Fuchsbauer, M.Izabachène, H.Sibert et D.Vergnaud à l'optimisation d'une méthode permettant de prouver que l'on connait une solution d'une équation bilinéaire. Ces derniers travaux ont donné lieu à une publication à la conférence ACNS 2010 [BFI+10a].

Dans une troisième partie, je décrirai mes travaux sur les signatures caméléons. Ces signatures permettent à l'émetteur-signataire de désigner une personne qui sera autorisée à modifier certaines parties du message signé (éventuellement selon certaines conditions). La personne choisie doit alors être capable d'accompagner le message modifié d'une signature valide de l'émetteur initial. Ces travaux, menés conjointement avec S.Canard, ont, pour l'instant, débouchés sur une publication à la conférence CT-RSA 2010 [CJ10a].

Enfin, je présenterai les applications auxquelles j'ai travaillé durant ma thèse. Celles-ci sont issues de cas d'usages courants pour lesquels nous avons cherché à protéger la vie privée des utilisateurs. Je me suis ainsi intéressée à la facturation de service, aux abonnements et, enfin, à la mise en place des techniques de protection de contenus dans un groupe hiérarchisé (la famille par exemple). Pour chacun de ces cas des solutions ne considérant pas l'aspect "vie privée" sont déjà utilisées en pratique, cependant peu de solution intègre cet aspect. Les diverses solutions que je présenterai ici ont données lieux à différents brevets [CJM07, CJM09, CJ09] ainsi qu'à deux publications, une première à la conférence Indocrypt 2008 [CJ08] et une seconde à la conférence Trustbus 2010 [CJ10b].

Première partie

Généralités

Chapitre 1

Mathématiques et outils pour la cryptographie

Sommaire

L'objectif de ce premier chapitre est d'introduire les mathématiques et les outils cryptographiques basiques essentiels à la compréhension de ce mémoire.

Je commencerai par un rappel sur les mathématiques utilisées en cryptographie, puis j'introduirai les principes de base de sécurité utilisés dans ce domaine. Enfin je décrirai un premier ensemble d'outils cryptographiques qui seront utilisés dans les divers schémas et protocoles que je présenterai dans ce mémoire.

1.1 Rappels de mathématiques pour la cryptographie

Cette première section présente les principaux outils mathématiques que nous utiliserons. Elle ne se veut pas exhaustive et un lecteur souhaitant aller plus loin pourra, par exemple, se référer à [MvV97].

1.1.1 Théorie des groupes

Je présente ici les principaux outils de théorie des groupes. Par abus de langage, nous parlerons d'entier ou de nombre au lieu de parler d'entier naturel. La quasi-totalité des groupes utilisés dans ce mémoire sont des groupes finis, c'est-à-dire un groupe dont le nombre de ses éléments, noté $|\mathbb{G}|$, est fini. Le nombre de ses éléments est appelé l'ordre de $\mathbb{G}$.

Définition 1 (Ordre d'un élément).
L'ordre d'un élément a de $\mathbb{G}$ est le plus petit entier positif k tel que $a^k = \mathbb{1}$ avec $a^k = \underbrace{a \cdot a \cdot \ldots \cdot a}_{k \text{ fois}}$.∗

Le théorème de Lagrange énonce une relation entre l'ordre d'un groupe et l'ordre de ses éléments.

Théorème 1 (Théorème de Lagrange). *L'ordre d'un élément de $\mathbb{G}$ divise l'ordre de $\mathbb{G}$.* ♢

Nous aurons besoin, par la suite, de l'indicatrice d'Euler.

Définition 2 (Indicatrice d'Euler).
Pour tout entier n, on note $\phi(n)$, le nombre d'entiers inférieurs à n qui sont premiers avec n. La fonction ϕ est appelée l'indicatrice d'Euler. Elle suit les propriétés suivantes.
- Si p est premier, alors $\phi(p) = p - 1$.
- Si p et q sont premiers entre eux, alors $\phi(pq) = \phi(p)\phi(q)$.
- Si p et q sont deux premiers distincts, alors $\phi(pq) = (p-1)(q-1)$. ∗

Un cas particulier des groupes finis, celui des groupes cycliques, est souvent utilisé en cryptographie.

Définition 3 (Groupe cyclique).
Un groupe $\mathbb{G}$ est un groupe cyclique s'il existe un élément g de $\mathbb{G}$ tel que, pour tout élément a de $\mathbb{G}$, il existe un entier x vérifiant $a = g^x$. Un tel élément g est appelé générateur du groupe $\mathbb{G}$. On écrit alors $\mathbb{G} =< g >$. ∗

Une notion souvent utilisée en cryptographie à clé publique est le logarithme discret d'un élément. Une généralisation de celui-ci permet de définir une représentation d'un élément dans une base.

Définition 4 (Représentation).
Soient $\mathbb{G}$ un groupe cyclique et $g_1, \ldots, g_t$ des générateurs de ce groupe. On appelle représentation de l'élément $a \in \mathbb{G}$ dans la base $g_1, \ldots, g_t$, le t-uplet d'entiers $(x_1, \ldots, x_t)$ tels que $a = \prod_{i=1}^{t} g_i^{x_i}$. ∗

Dans ce mémoire, $\mathbb{Z}_n$ représente l'anneau des classes modulo n et $\mathbb{Z}_n^*$ le groupe des éléments inversibles pour la multiplication de $\mathbb{Z}_n$. Nous avons $|\mathbb{Z}_n^*| = \phi(n)$.

Si p premier, alors $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$ est un groupe cyclique.

1.1.2 Courbes elliptiques et couplages

Une courbe elliptique est une courbe algébrique particulière. Elle possède une loi d'addition géométrique pour laquelle l'élément neutre est un point à l'infini noté $\mathcal{O}$. Une courbe elliptique et sa loi associée forment un groupe. La première utilisation de ces courbes en cryptographie est due aux travaux de V.S.Miller publiés dans [Mil86], il propose alors pour la première fois d'appliquer un protocole cryptographique, l'échange de clés Diffie-Hellman, sur les courbes elliptiques. En effet, les différents protocoles cryptographiques sur les groupes finis peuvent être mis en place sur de telles courbes. L'avantage principal est, que pour un même niveau de sécurité, la taille des éléments utilisés sur une courbe elliptique est plus petite que dans le cas des corps finis.

En ce qui concerne la recherche sur les protocoles cryptographiques, l'introduction des courbes elliptiques en cryptographie a permis d'explorer de nouvelles constructions utilisant des couplages, c'est-à-dire des applications bilinéaires particulières. Ceux-ci sont apparus pour la première fois en 1991 dans [MVO91] dans l'objectif de résoudre le problème du logarithme discret (voir Définition 10). Et aujourd'hui, les couplages sont couramment utilisés dans les schémas de signature et de chiffrement.

Plus formellement, un couplage nécessite un environnement bilinéaire que nous définissons comme suit.

Définition 5 (Environnement bilinéaire pour un groupe d'ordre premier p).
Soient $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ trois groupes cycliques d'ordre premier p.

Un couplage (admissible) est une application $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ ayant les trois propriétés suivantes.

Calculable. Pour tout $(x, y) \in \mathbb{G}_1 \times \mathbb{G}_2$, il existe un algorithme efficace pour calculer $\mathsf{e}(x, y)$.

Bilinéaire. Pour tout $(x, y) \in \mathbb{G}_1 \times \mathbb{G}_2$, et tout $(a, b) \in \mathbb{Z}_p^2$, $\mathsf{e}(x^a, y^b) = \mathsf{e}(x, y)^{ab}$.

Non-dégénérée. Il existe g_1 générateur de $\mathbb{G}_1$ et g_2 générateur de $\mathbb{G}_2$ tel que $\mathsf{e}(g_1, g_2) \neq \mathbb{1}_T$ soit un générateur de $\mathbb{G}_T$.

Enfin, un isomorphisme $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ est souvent utilisé. Il est parfois supposé que cet isomorphisme est à sens unique, c'est-à-dire efficacement calculable uniquement de $\mathbb{G}_2$ vers $\mathbb{G}_1$.

On appelle environnement bilinéaire l'ensemble d'éléments $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e}, \psi)$ avec ψ optionnel. ∗

Dans certaines applications, les groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ sont identiques. L'application ψ peut alors être l'identité. On parle dans ce cas de couplage symétrique, et nous notons $(p, \mathbb{G}, \mathbb{G}_T, g, \mathsf{e}, \psi)$ l'environnement bilinéaire correspondant avec ψ optionnel.

Enfin, il existe également des couplages, introduits par D.Boneh, E.-J.Goh et K.Nissim dans [BGN05], définis dans les groupes d'ordre composites $n = pq$ avec p et q premiers. Bien que l'ordre n puisse être rendu public, il est souvent important que la factorisation $n = pq$ reste secrète. Nous verrons, dans la suite de ce chapitre, l'hypothèse sous-groupe décisionnel (voir Définition 19) qui repose sur la difficulté de cette factorisation.

1.1.3 Fonction et probabilité négligeable

Cette section présente les notions de fonction, et probabilité, négligeables. Ces notions sont utilisées, en cryptographie, pour borner la probabilité qu'un adversaire réussisse une attaque contre un schéma. Une probabilité est dite "négligeable" lorsqu'elle est très proche de 0. Pour définir formellement cette notion nous devons introduire le concept de fonction négligeable.

Définition 6 (Fonction négligeable).
Une fonction $\epsilon : \mathcal{N} \to [0,1]$ est dite négligeable en k si toute fraction rationnelle, aussi petite soit-elle, majore la fonction $\epsilon(k)$ à partir d'un certain rang. Plus formellement, nous avons :

$$\forall c > 0, \exists k_0 > 0 \text{ tels que } \forall k > k_0, \epsilon(k) < \frac{1}{k^c}. \qquad *$$

On peut ainsi définir une probabilité négligeable de la façon suivante.

Définition 7 (Probabilité négligeable).
Une probabilité P, dépendante d'un paramètre k, est dite négligeable si $P(k)$ est une fonction négligeable. *

1.1.4 Interpolation

Nous utiliserons, pour plusieurs de nos solutions, le principe d'interpolation que je rappelle brièvement ici.

Définition 8 (Interpolation).
Soient $(n+1)$ couples $\{(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)\}$. Les $(x_i)_{0 \leq i \leq n}$ sont appelés points d'interpolation et les $(y_i)_{0 \leq i \leq n}$ représentent les valeurs d'interpolation. Pour interpoler une fonction f, on définit ces valeurs comme :

$$y_i = f(x_i), \quad \forall i = 0, \cdots, n \qquad *$$

L'interpolation que nous utiliserons à plusieurs reprises est celle de Lagrange qui consiste à calculer le polynôme d'interpolation comme suit.

Définition 9 (Polynôme d'interpolation de Lagrange).
Le polynôme d'interpolation de Lagrange est l'unique polynôme P_n, de degré n, tel que

$$P_n(x_i) = f(x_i), \quad \forall i = 0, \ldots, n.$$

Le polynôme qui satisfait cette égalité peut être décrit comme ceci.

$$P_n(x) = \sum_{k=0}^{n} l_k(x) f(x_k)$$

où les l_k sont des polynômes de degré n qui forment une base de l'espace $\mathcal{P}_n$ des polynômes de degré au plus n.

$$l_k(x) = \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i} = \frac{x - x_0}{x_k - x_0} \cdots \frac{x - x_{k-1}}{x_k - x_{k-1}} \frac{x - x_{k+1}}{x_k - x_{k+1}} \cdots \frac{x - x_n}{x_k - x_n} \qquad *$$

Le théorème suivant pose l'unicité de ce polynôme.

Théorème 2 (Unicité du polynôme d'interpolation de Lagrange). *Il n'existe qu'un seul polynôme P_n de degré au plus n défini par un ensemble de $n+1$ points.* ♢

1.2 Sécurité en cryptographie

En cryptographie, il est important de s'assurer de la sécurité de nos schémas. En cryptographie à clé publique nous utilisons pour cela des preuves de sécurité établies dans des modèles de sécurité. Ces modèles établissent un environnement général permettant de prouver qu'un ensemble de propriétés nécessaires à un schéma sont atteintes. Il existe différentes techniques pour montrer qu'une propriété est respectée, notamment, en prouvant que casser cette propriété est au moins aussi difficile qu'un problème mathématique dit "difficile". Je décrirai principalement, dans cette section, les deux modèles que nous utiliserons dans ce mémoire. Je présenterai ensuite les principaux problèmes difficiles qui nous seront utiles. Enfin, j'esquisserai les deux techniques de preuve que j'utiliserai dans ce mémoire.

1.2.1 Modèles de sécurité

Nous nous intéressons particulièrement, ici, à deux des modèles les plus utilisés pour prouver la sécurité d'un schéma :le modèle de l'oracle aléatoire et le modèle standard.

Modèle de l'oracle aléatoire

Ce modèle fut suggéré par A.Fiat et A.Shamir dans [FS87] puis modélisé, en 1993, par M.Bellare et P.Rogaway [BR93]. L'objectif de ce modèle est de prouver la sécurité de protocoles efficaces. Pour cela, le modèle propose que les fonctions de hachage (voir Section 1.3.2) soient considérées comme des fonctions aléatoires. Dans ce contexte, l'utilisation d'une fonction de hachage lors d'une preuve est remplacée par un oracle aléatoire dont la sortie est indistinguable, pour un adversaire, de la sortie d'une fonction parfaitement aléatoire. Afin de simuler au mieux une fonction de hachage, l'oracle doit renvoyer la même réponse pour toutes questions identiques. Ce modèle est parfois considéré comme trop idéalisé. En effet, R.Canetti, O.Goldreich et S.Halevi [CGH98, CGH04] ont prouvé l'existence de schémas sûrs dans ce modèle pour lesquels toute implémentation de l'oracle par une fonction de hachage rend le schéma non-sûr. D'autre part, M.Bellare, A.Boldyreva et A.Palacio ont proposé dans [BBP04] des constructions atteignant certaines propriétés uniquement dans ce modèle et pour lesquelles relâcher cette contrainte empêche de prouver leur sécurité. Cependant, ces contre-exemples reposent sur des constructions très spécifiques et artificielles, et le modèle de l'oracle aléatoire permet d'obtenir des schémas efficaces bénéficiant de preuves de sécurité simples et donc plus facilement vérifiables. Ce modèle est donc toujours largement utilisé.

Modèle standard

Le modèle standard ne fait pas d'idéalisation spécifique. En particulier, il n'idéalise pas les fonctions de hachage et considère uniquement les propriétés induites par leur définition (comme la résistance aux collisions, voir Section 1.3.2). Ce modèle permet d'atteindre des propriétés de sécurité plus fortes qui ne reposent que sur des hypothèses calculatoires et non sur des constructions idéalisées. Cependant, les constructions obtenues sont souvent moins efficaces et les preuves plus difficiles. Il existe cependant quelques schémas efficaces dans ce modèle. Nous verrons notamment au prochain chapitre le schéma de signature de D.Boneh et X.Boyen [BB08] qui est très efficace et prouvé sûr dans ce modèle.

Autres modèles

D'autres modèles peuvent parfois être utilisés comme, par exemple, le modèle générique. Celui-ci considère que les opérations arithmétiques ou les comparaisons d'éléments de groupe n'ont pas de propriété particulière. De manière plus générale, l'attaquant est incapable d'obtenir d'information sur la structure du groupe. Ce modèle est généralement utilisé pour définir le niveau de sécurité d'un nouveau problème difficile par rapport aux problèmes connus.

1.2.2 Problèmes difficiles

La sécurité des schémas en cryptographie publique est souvent ramenée à la difficulté de résolution d'un problème mathématique dit "difficile". Il s'agit de problèmes pour lesquels il est facile de créer des instances dont une solution est connue mais dont la résolution générale est calculatoirement difficile. Je vais définir, ici, les principaux problèmes que nous utiliserons dans ce mémoire, chaque problème est relié à une hypothèse calculatoire portant le même nom.

Problèmes basés sur le logarithme discret

Le problème du logarithme discret est un problème de la théorie des nombres fréquemment utilisé en cryptographie. Il repose sur la difficulté de calculer le logarithme discret d'un élément donné.

Définition 10 (Problème du logarithme discret DL).
Soient $\mathbb{G}$ un groupe d'ordre premier p et g un générateur de ce groupe. Étant donné un élément y de $\mathbb{G}$, le problème DL dans $\mathbb{G}$ est de trouver $x \in \mathbb{Z}_p$ tel que $y = g^x$. L'entier x est appelé le logarithme discret de y en base g. ∗

La définition que je viens de donner est celle que j'utilise principalement dans ce mémoire, il est cependant à noter qu'il est possible de donner une définition plus générale n'imposant pas un groupe d'ordre premier. Ce problème peut notamment être utilisé dans le cas d'un groupe d'ordre composé $n = pq$ avec p et q premiers.

Chaque problème calculatoire a une hypothèse associée qui pose que tout adversaire probabiliste, en temps polynomial, a une probabilité de réussite négligeable. Par exemple, pour le problème DL nous avons l'hypothèse suivante.

Définition 11 (Hypothèse du logarithme discret DL).
L'hypothèse du logarithme discret consiste à supposer qu'il n'existe pas d'algorithme probabiliste permettant de résoudre le problème du logarithme discret en temps polynomial avec une probabilité non négligeable. ∗

Ce problème fut dérivé par Diffie et Hellman dans [DH76] en un problème légèrement plus facile : le problème Diffie-Hellman calculatoire, noté CDH.

Définition 12 (Problème Diffie-Hellman calculatoire CDH).
Soient $\mathbb{G}$ un groupe d'ordre premier p et g un générateur de ce groupe. Étant donné (g^a, g^b) avec a et b choisis aléatoirement dans $\mathbb{Z}_p$, le problème CDH dans $\mathbb{G}$ est de calculer g^{ab}. ∗

On remarque qu'il est évident qu'un algorithme capable de résoudre le problème DL peut résoudre le problème CDH.

Il existe une variante décisionnelle de ce problème dans laquelle on souhaite savoir si une proposition donnée est vérifiée. Plus formellement, le problème décisionnel correspondant est le suivant.

Définition 13 (Problème Diffie-Hellman décisionnel DDH).
Soient $\mathbb{G}$ un groupe d'ordre premier p et g un générateur de ce groupe. Étant donné (g^a, g^b, g^c) avec a, b et c dans $\mathbb{Z}_p$, le problème Diffie-Hellman décisionnel, noté DDH, consiste à décider si $g^{ab} = g^c$. ∗

L'hypothèse décisionnelle ne peut être formulée de la même façon que la version calculatoire. En effet, pour un problème décisionnel, un adversaire répondant juste avec une probabilité négligeable est un adversaire qui se trompe systématiquement. Il suffit alors de l'utiliser pour construire un adversaire qui répondra systématiquement l'inverse pour obtenir un adversaire gagnant avec une probabilité écrasante. Pour que le problème soit difficile, il faut donc qu'un adversaire ne puisse trouver la bonne réponse qu'une fois sur deux. Ainsi, l'hypothèse pour les problèmes décisionnels suppose que pour tout adversaire polynomial cherchant à décider le problème, la différence de sa probabilité de réussite avec 1/2 est négligeable. Par exemple, pour le problème DDH nous avons l'hypothèse décisionnelle suivante.

Définition 14 (Hypothèse Diffie-Hellman décisionnel DDH).
L'hypothèse Diffie-Hellman décisionnel suppose que quel que soit l'adversaire probabiliste, en temps polynomial, cherchant à distinguer les distributions (g^a, g^b, g^c) et (g^a, g^b, g^{ab}), la différence de sa probabilité de réussite avec 1/2 est négligeable. ∗

Problèmes difficiles utilisant les couplages

L'utilisation des couplages en cryptographie a introduit de nouveaux problèmes difficiles propres à ce nouvel environnement. D'autre part, certains problèmes difficiles, comme DDH par exemple, sont facilement résolus lorsque certains couplages existent dans le système.

En effet, pour DDH, supposons que nous soyons dans le cas d'un environnement bilinéaire symétrique $(\mathbb{G}, \mathbb{G}_T, p, g, \mathsf{e}, \psi)$ avec p premier. Étant donné $g^a, g^b, g^c \in \mathbb{G}$, distinguer si $c = ab$ consiste simplement à vérifier si $\mathsf{e}(g^a, g^b) = \mathsf{e}(g^c, g)$. Si l'égalité est vraie alors $\mathsf{e}(g^a, g^b) = \mathsf{e}(g, g)^{ab} = \mathsf{e}(g, g)^c$ et donc $c = ab$.

Je vais définir ici les principaux problèmes et hypothèses pour les couplages que nous utiliserons dans la suite de ce mémoire.

J.Camenisch, S.Hohenberger et A.Lysyanskaya ont introduits dans [CHL05], l'hypothèse Diffie-Hellman externe, ou XDH, qui repose sur la difficulté du problème DDH. Or nous venons de voir que dans le cas d'un couplage symétrique ce problème devenait facile, il est donc nécessaire de prendre deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ distincts. L'hypothèse est alors la suivante.

Définition 15 (Hypothèse Diffie-Hellman externe XDH).
Soit $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2)$ un environnement bilinéaire asymétrique avec p premier et pour lequel il n'y a pas d'isomorphisme ψ calculable de $\mathbb{G}_1$ vers $\mathbb{G}_2$. L'hypothèse Diffie-Hellman externe, notée XDH, suppose que l'hypothèse DDH est vraie dans le groupe $\mathbb{G}_1$. ∗

L'hypothèse Diffie-Hellman externe symétrique porte, elle-aussi, sur la difficulté du problème DDH mais suppose cette fois que le problème DDH est difficile dans les deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$. Il est donc toujours nécessaire de prendre deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ distincts. Mais il faut, de plus, s'assurer qu'il n'y ait pas d'isomorphisme efficacement calculable, non seulement de $\mathbb{G}_1$ vers $\mathbb{G}_2$, mais aussi de $\mathbb{G}_2$ vers $\mathbb{G}_1$. L'hypothèse est alors la suivante.

Définition 16 (Hypothèse Diffie-Hellman externe flexible symétrique SXDH).
Soit $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2)$ un environnement bilinéaire asymétrique avec p premier pour lequel il n'y ait pas d'isomorphisme ψ calculable d'un groupe vers l'autre. L'hypothèse Diffie-Hellman externe flexible symétrique, ou SXDH, suppose que l'hypothèse DDH est vraie dans le groupe $\mathbb{G}_1$ et dans le groupe $\mathbb{G}_2$. ∗

Je vais à présent décrire trois problèmes spécifiques aux environnements bilinéaires. Pour chacun d'entre eux l'hypothèse associée est similaire aux exemples que nous avons donnés précédemment pour les cas calculatoires ou décisionnels, je ne la décrirai donc pas.

Le premier problème, a été introduit par D.Boneh, X.Boyen et H.Shacham dans [BBS04]. Ce problème décisionnel est inspiré du problème DDH. Par contre, celui-ci utilise un environnement bilinéaire symétrique.

Définition 17 (Problème Décisionnel Linéaire DLIN).
Soit $(p, \mathbb{G}, \mathbb{G}_T, g, \mathsf{e})$ un environnement bilinéaire symétrique avec p premier.

Étant donné (u, v, w, u^a, v^b, w^c) avec u, v, w choisis aléatoirement dans $\mathbb{G}$ et a, b, c choisis aléatoirement dans $\mathbb{Z}_p$, le problème Décisionnel Linéaire, noté DLIN, consiste à décider si $w^c = w^{a+b}$. ∗

Le problème suivant, a été introduit par D.Boneh et X.Boyen dans [BB04]. Celui-ci leur permet de prouver la sécurité de leur schéma de signature que nous verrons au chapitre suivant.

Définition 18 (Problème q-Diffie-Hellman flexible $q-$SDH).
Soit $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2)$ un environnement bilinéaire avec p premier.

Étant donné $(g_1, g_2, g_2^x, g_2^{x^2}, \ldots, g_2^{x^q})$ avec x choisi aléatoirement dans $\mathbb{Z}_p$, le problème q-Diffie-Hellman flexible, noté $q-$SDH, consiste à trouver un couple $(c, g_1^{\frac{1}{x+c}})$ avec $c \in \mathbb{Z}_p$. ∗

Enfin, le problème de Décision du sous-groupe, a été proposé dans [BGN05] par D.Boneh, E.-J.Goh et K.Nissim.

Définition 19 (Problème de Décision du sous-groupe SD).
Soit $(n, \mathbb{G}, \mathbb{G}_T, g, \mathsf{e})$ un environnement bilinéaire avec $n = pq$, p et q premiers. Soient $\mathbb{G}_p$ le sous-groupe de $\mathbb{G}$ des éléments d'ordre p sans résidu d'ordre q, c'est-à-dire tel que $\forall a \in \mathbb{G}_p, a^p = \mathbb{1}_{\mathbb{G}}$, et $\mathbb{G}_q$ le sous-groupe de $\mathbb{G}$ des éléments d'ordre q sans résidu d'ordre p, c'est-à-dire tel que $\forall a \in \mathbb{G}_q, a^q = \mathbb{1}_{\mathbb{G}}$.

Le problème SD consiste à décider si un élément donné à été choisi aléatoirement dans $\mathbb{G}$ ou dans un de ses sous-groupes. ∗

1.2.3 Sécurité prouvée : principes

Maintenant que nous avons vu les hypothèses calculatoires et décisionnelles que nous utiliserons dans ce mémoire ainsi que les principaux modèles de sécurité, je peux maintenant esquisser comment ceux-ci sont utilisés pour prouver la sécurité d'un schéma.

Le principe de la sécurité prouvée en cryptographie publique se déroule généralement en plusieurs phases.

- Dans une première phase, on établit les propriétés de sécurité nécessaires pour un schéma donné. On appelle modèle du schéma l'ensemble des propriétés nécessaires pour considéré le schéma comme sûr.
- Chaque propriété est ensuite formalisée sous la forme d'une expérience qui définit les oracles auxquels l'adversaire a accès, c'est-à-dire les types de requêtes que celui-ci peut faire et le type de réponse qu'il recevra. Une propriété est considérée comme acquise par un schéma si pour tout adversaire $\mathcal{A}$ en temps polynomial en le(s) paramètre(s) de sécurité la probabilité de son succès dans l'expérience correspondante est négligeable (ou la différence entre sa probabilité de réussite et $1/2$ est négligeable dans le cas d'expériences décisionnelles pour lesquelles gagner revient à choisir correctement un élément parmi deux).
  Par abus de langage, nous parlerons dans ce mémoire d'adversaire polynomial pour désigner un adversaire en temps polynomial en le(s) paramètre(s) de sécurité.

– Chaque propriété est ensuite prouvée en utilisant une technique de preuve.

Dans ce mémoire, nous utiliserons deux techniques de preuves par réduction : les preuves par l'absurde et les preuves par jeux.

Les preuves par réduction

Le principe des preuves par réduction est basé sur l'hypothèse qu'un problème est difficile ou, comme cela sera le cas à plusieurs reprises dans ce mémoire, qu'un autre schéma respecte une propriété donnée.

Preuves par l'absurde Il s'agit ici de prouver par l'absurde que si un adversaire est capable de gagner une expérience $\text{Exp}^{\mathcal{X}}_{\mathbf{prop}}$ contre une propriété **prop** d'un schéma $\mathcal{X}$ alors celui-ci peut être utilisé pour construire un adversaire capable de résoudre le problème difficile ou de gagner l'expérience contre la propriété $\mathbf{prop}^*$ assurée par le second schéma $\mathcal{Y}$.

Plus formellement, on considère deux parties, l'adversaire $\mathcal{A}^{\mathcal{X}}_{\mathbf{prop}}$ efficace contre l'expérience $\text{Exp}^{\mathcal{X}}_{\mathbf{prop}}$ et un challengeur $\mathcal{C}$ contre le problème difficile considéré ou contre l'expérience qui correspond à la propriété $\mathbf{prop}^*$. L'objectif de la preuve est alors de construire un simulateur qui émule l'environnement de l'adversaire de telle sorte que celui-ci ne soit pas capable de distinguer la simulation d'un cas réel. Dans le cas d'un challengeur représentant une expérience, le simulateur peut utiliser tous les oracles de celle-ci afin de parfaire sa simulation. L'adversaire va se comporter normalement et ainsi produire, en temps polynomial, une réponse permettant de gagner contre l'expérience $\text{Exp}^{\mathcal{X}}_{\mathbf{prop}}$ avec une probabilité qui n'est pas négligeable. Le simulateur utilise ensuite cette réponse pour casser le problème difficile ou la propriété représentée par le challengeur.

Il ne reste alors plus qu'à conclure en évaluant la probabilité de succès de l'adversaire (ou son avantage dans le cas d'un problème décisionnel). D'après l'hypothèse de départ, le problème difficile ou l'expérience ne peut être résolue qu'avec une probabilité négligeable (ou un avantage négligeable pour une expérience décisionnelle), en conséquence cette probabilité borne la réussite de l'adversaire. Et il est possible de conclure que le succès de l'adversaire (ou son avantage) est négligeable.

Les preuves par jeu Une seconde technique de preuve que nous utiliserons dans ce mémoire est la technique des preuves par jeu. Cette méthode formalisée par V.Shoup dans [Sho04] consiste, dans un premier temps, à représenter formellement l'expérience de l'attaque de l'adversaire. Cette première représentation est appelé le Jeu 0 ou Jeu initial. On associe à ce jeu un événement S_0 qui représente l'événement "l'adversaire gagne". On dérive ensuite, par étape, ce jeu initial afin de se ramener à un jeu final dont on peut facilement déduire la probabilité. L'objectif est que chaque étape représente un jeu intermédiaire tel que la probabilité que l'adversaire gagne contre ce jeu soit très proche de la probabilité que l'adversaire gagne le jeu précédent. On peut ainsi majorer la réussite de l'adversaire, c'est-à-dire la probabilité de l'événement S_0, par la somme de la probabilité que l'adversaire gagne le jeu final et des différences de probabilité entre chaque jeu. On peut ainsi en déduire le succès (ou l'avantage) de l'adversaire contre l'expérience.

1.3 Outils cryptographiques

Nous utiliserons dans ce mémoire un ensemble d'outils cryptographiques classiques. Je vais à présent définir un premier sous-ensemble d'entre eux, les briques cryptographiques plus im-

portantes seront abordées plus tard, notamment au chapitre 2.

1.3.1 Fonction à sens unique

Une fonction à sens unique est une fonction facile à évaluer mais difficile à inverser. De manière formelle nous avons la définition suivante.

Définition 20 (Fonction à sens unique).
Soit $f : \{0,1\}^m \to \{0,1\}^n$, on dit que f est à sens unique si f suit les deux propriétés suivantes.

- **Efficacité**. Il existe un algorithme efficace qui, pour tout $x \in \{0,1\}^m$, retourne $f(x)$.
- **À sens unique**. Soit λ un paramètre de sécurité, pour tout algorithme efficace, et pour tout élément $y = f(x)$ avec $x \in \{0,1\}^\lambda$, la probabilité de trouver x' tel que $y = f(x')$ est négligeable.
  Plus formellement, soit λ un paramètre de sécurité, l'expérience correspondant à cette propriété définit un adversaire $\mathcal{A}$ face à un oracle fCHALLENGE qui retourne des sorties $\{y_i = f(x_i)\}_{i\in[1,\mathsf{n}]}$ de f pour des x_i choisis aléatoirement dans $\{0,1\}^\lambda$ et gardés secrets. À la fin de l'expérience, l'adversaire retourne un élément x^*. Son succès dans cette expérience est :
  $$\mathsf{Succ}^f_{\mathbf{OW},\mathcal{A}}(\lambda) = Pr[\exists i \in [1,\mathsf{n}] | y_i = f(x^*)].$$
  Une fonction f est dite **à sens unique** si quel que soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^f_{\mathbf{OW},\mathcal{A}}(\lambda)$ est négligeable :
  $$\mathsf{Succ}^f_{\mathbf{OW},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable.} \quad *$$

Un exemple d'une telle fonction est la fonction $f : x \longmapsto g^x$ dans un groupe respectant l'hypothèse du logarithme discret. En effet, pour tout x il est facile de calculer g^x. Par contre, retrouver l'antécédent d'un élément y revient à casser l'hypothèse du logarithme discret. Cette fonction est donc à sens unique sous l'hypothèse DL.

1.3.2 Fonction de hachage

Une fonction de hachage est une fonction à sens unique particulière qui permet, entre autres, de ramener une chaîne de bit de taille quelconque $\{0,1\}^*$ en un condensé de taille fixe $\{0,1\}^\lambda$ avec λ un paramètre de sécurité. Ces fonctions sont largement utilisées en cryptographie sous leur forme dite "cryptographiquement sûre" qui est définit de la façon suivante.

Définition 21 (Fonctions de hachage cryptographiquement sûres $\mathcal{H}$).
Une famille H de fonctions de hachage $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$, avec λ un paramètre de sécurité, est une famille de fonctions de hachage cryptographiquement sûres si elle vérifie les trois propriétés de sécurité suivantes.

Résistance à la pré-image Toute fonction $\mathcal{H} \in H$ est une fonction à sens unique.

Résistance à la seconde pré-image La probabilité qu'un adversaire, étant donnés un message $x \in \{0,1\}^*$ et une fonction $\mathcal{H} \in H$, trouve $x' \neq x$ telle que $\mathcal{H}(x) = \mathcal{H}(x')$ est négligeable.

Résistance aux collisions La probabilité qu'un adversaire, étant donné une fonction $\mathcal{H} \in H$, trouve un couple (x, x') tel que $\mathcal{H}(x) = \mathcal{H}(x')$ et $x' \neq x$ est négligeable.
La **résistance aux collisions** impliquant la **résistance à la seconde pré-image**, nous ne donnons une version plus formelle que de cette dernière propriété. L'expérience qui correspond à la **résistance aux collisions** définit un adversaire $\mathcal{A}$ qui a accès à un oracle

$\mathcal{O}.\mathcal{H}(\mathsf{m})$ qui retourne le condensé du message m pour la fonction $\mathcal{H}$. À la fin de l'expérience, l'adversaire retourne un couple (x_1^*, x_2^*) son succès dans cette expérience est :

$$\mathsf{Succ}^{\mathcal{H}}_{\mathbf{collRes},\mathcal{A}}(\lambda) = Pr[\mathcal{H}(x_1^*) = \mathcal{H}(x_2^*) \text{ avec } x_1^* \neq x_2^*].$$

Une fonction de hachage $\mathcal{H}$ est dite **résistante aux collisions** si quel que soit l'adversaire polynomial $\mathcal{A}$, son succès $\mathsf{Succ}^{\mathcal{H}}_{\mathbf{collRes},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{H}}_{\mathbf{collRes},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.} \quad *$$

Dans ce mémoire nous n'utiliserons que des fonctions de hachage cryptographiquement sûres.

Il existe de très nombreuses fonctions de hachages (MD4, MD5, SHA1, SHA256,...). Toutefois très peu se sont avérées résistantes aux collisions. Notamment, MD4, MD5 et SHA1 ne sont plus considérées comme cryptographiquement sûres. Le NIST (National Institute of Standards and Technology) conseille de leurs préférer SHA256. D'autre part, le nombre de fonction estimée sûre se réduisant, il a lancé en 2007 un appel à contribution afin de choisir une nouvelle fonction, SHA-3, pour être le futur standard. L'issue de la compétition est prévue pour fin 2012.

Remarque. Une fonction de hachage modélisée par un oracle aléatoire respecte l'ensemble de ces propriétés. En effet, pour deux entrées différentes, l'oracle retourne deux aléas distincts. Ainsi, la modélisation atteint la propriété de résistance aux collisions. De plus, la sortie étant indépendante de l'entrée, elle respecte aussi la résistance à la pré-image.

1.3.3 Fonction de hachage caméléon

Nous utiliserons dans ce mémoire un type de fonction de hachage très particulier introduite par H.Krawczyk et T.Rabin [KR00] : les fonctions de hachage caméléon. Une fonction de hachage caméléon est un schéma composé de quatre protocoles permettant d'obtenir une fonction de hachage cryptographiquement sûre pour tout adversaire ignorant la clé secrète du schéma et permettant, avec cette clé, de générer efficacement une seconde pré-image. Enfin il doit être calculatoirement difficile de distinguer la collision du message originellement haché.

Notons qu'il est d'usage, dans le cas des fonctions de hachage caméléon, d'appeler "collision" l'obtention de cette seconde pré-image. Je me conformerai donc ici à l'usage.

Plus formellement, un schéma de fonction caméléon est défini comme suit.

Définition 22 (Fonction de hachage caméléon sûre).
Un schéma de fonction de hachage caméléon $\mathcal{HC}$ est défini par les quatre algorithmes suivants.

- $\mathcal{HC}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système notés $\mathcal{HC}.\mathsf{param}$. Nous considérerons dans la suite que λ est inclus dans les paramètres.
$$\mathcal{HC}.\mathsf{param} \leftarrow \mathcal{HC}.\textsc{Init}(1^\lambda)$$
- $\mathcal{HC}$.GénClé génère la paire de clés de la fonction $(\mathsf{csk}, \mathsf{cpk})$ en utilisant les paramètres du système $\mathcal{HC}.\mathsf{param}$. Nous considérerons dans la suite que $\mathcal{HC}.\mathsf{param}$ est inclus dans la clé publique cpk.
$$(\mathsf{csk}, \mathsf{cpk}) \leftarrow \mathcal{HC}.\textsc{GénClé}(\mathcal{HC}.\mathsf{param})$$
- $\mathcal{HC}$.Hache est la fonction de hachage du schéma. Elle prend en entrée un message m, un aléa r ainsi que la clé publique de la fonction cpk et retourne un condensé h.
$$\mathsf{h} \leftarrow \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}, \mathsf{r})$$

– $\mathcal{HC}$.Forge permet au détenteur de la clé secrète de la fonction de hachage caméléon de générer des collisions sur le message de son choix. Cet algorithme prend en entrées la clé secrète de la fonction de hachage csk, le message souhaité pour la collision m', ainsi que le message m et l'aléa r correspondant au condensé ciblé h. Il retourne un nouvel aléa r' tel que $\mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}, \mathsf{r}) = \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}', \mathsf{r}') = \mathsf{h}$.

$$\mathsf{r}' \leftarrow \mathcal{HC}.\textsc{Forge}(\mathsf{csk}, \mathsf{m}', \mathsf{m}, \mathsf{r}, \mathsf{h})$$

Un tel schéma est considéré comme sûr s'il présente les deux propriétés de sécurité suivantes.

– **Uniformité**. Un adversaire ne doit pas être capable de distinguer un couple (m, r) avec r choisit aléatoirement d'un couple $(\mathsf{m}', \mathsf{r}')$ obtenu par collision, c'est-à-dire avec :

$$\mathsf{r}' = \mathcal{HC}.\textsc{Forge}(\mathsf{csk}, \mathsf{m}, \mathsf{m}', \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}, \mathsf{r})).$$

– **Résistance aux collisions**. Il doit être infaisable pour un adversaire, ignorant la clé secrète csk, de trouver deux couples (m, r) et $(\mathsf{m}', \mathsf{r}')$ tels que $\mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}, \mathsf{r}) = \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}', \mathsf{r}')$ et $\mathsf{m}' \neq \mathsf{m}$ pour une clé publique cpk donnée.
Plus formellement, l'expérience qui correspond à la **résistance aux collisions** définie un adversaire $\mathcal{A}$ connaissant la clé publique de la fonction de hachage caméléon cpk et qui a accès à un oracle $\mathcal{O}.\mathcal{HC}.\textsc{Forge}(\mathsf{m}', \mathsf{m}, \mathsf{r}, \mathsf{h})$ retournant les aléas correspondants. Nous notons $\{\mathsf{m}'_i, \mathsf{m}_i\}_{i\in[1,\mathsf{n}]}$ l'ensemble des messages m' et m des requêtes à l'oracle $\mathcal{O}.\mathcal{HC}.\textsc{Forge}$ et $\{\mathsf{r}_i\}_{i\in[1,\mathsf{n}]}$ les réponses correspondantes. À la fin de l'expérience l'adversaire retourne un quintuplet $(\mathsf{cpk}, \mathsf{m}^*, \mathsf{r}^*, \tilde{\mathsf{m}}^*, \tilde{\mathsf{r}}^*)$, à condition que $(\mathsf{m}^*, \mathsf{r}^*) \notin \{(\mathsf{m}'_i, \mathsf{r}'_i)\}_{i\in[1,\mathsf{n}]}$, $(\tilde{\mathsf{m}}^*, \tilde{\mathsf{r}}^*) \notin \{(\mathsf{m}'_i, \mathsf{r}'_i)\}_{i\in[1,\mathsf{n}]}$ et $(\mathsf{m}^*, \mathsf{r}^*) \neq (\tilde{\mathsf{m}}^*, \tilde{\mathsf{r}}^*)$ le succès de l'adversaire dans cette expérience est :

$$\mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda) = Pr\big[\mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}^*, \mathsf{r}^*) = \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \tilde{\mathsf{m}}^*, \tilde{\mathsf{r}}^*)\big].$$

Une fonction de hachage $\mathcal{HC}$ est dite **résistante aux collisions** si le succès $\mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda)$ de tout adversaire polynomial $\mathcal{A}$ est négligeable.

$$\mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.} \qquad *$$

Les premiers schémas de fonction de hachage caméléon, notamment les solutions proposées dans [KR00], partagent une faiblesse à l'attaque dite "par exposition de clé" (*Key Exposure attack*). À partir de deux couples message-aléa et de leur condensé commun, il est possible de retrouver la clé secrète de la fonction et ainsi pouvoir construire des collisions sur n'importe quel message. Nous utiliserons, à dessein, un tel schéma dans ce mémoire. Nous appelons un tel schéma un schéma de fonction de hachage caméléon faible. Le schéma dont nous utiliserons la faiblesse dans ce mémoire est défini comme suit.

Construction 1 (Schéma de fonction de hachage caméléon faible).
Soit λ un paramètre de sécurité, le schéma fonctionne de la manière suivante.

$f\mathcal{HC}.\textsc{Init}(\lambda)$ On choisit p premier et on prend g un générateur de $\mathbb{Z}^*_p$.

$$f\mathcal{HC}.\mathsf{param} = \{g, \mathbb{Z}^*_p, \lambda\} \leftarrow f\mathcal{HC}.\textsc{Init}(\lambda)$$

$f\mathcal{HC}.\textsc{GénClé}(f\mathcal{HC}.\mathsf{param})$ La clé secrète csk est choisie aléatoirement dans $\{0,1\}^\lambda$, la clé publique correspondante est $\mathsf{cpk} = \{g^{\mathsf{csk}}, f\mathcal{HC}.\mathsf{param}\}$. Par abus de langage, on dira cpk pour parler de g^{csk}.

$$(\mathsf{csk}, \mathsf{cpk}) \leftarrow f\mathcal{HC}.\textsc{GénClé}(f\mathcal{HC}.\mathsf{param})$$

$f\mathcal{HC}$.Hache($\mathsf{cpk}, \mathsf{m}, \mathsf{r}, f\mathcal{HC}.\mathsf{param}$) Le condensé correspondant au message m et à l'aléa r pour la clé publique cpk est obtenu comme suit.

$$\mathsf{h} = \mathsf{cpk}^{\mathsf{m}} \cdot g^{\mathsf{r}}$$

$$\mathsf{h} \leftarrow f\mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}, \mathsf{r})$$

$f\mathcal{HC}$.Forge($\mathsf{csk}, \mathsf{m}, \mathsf{m}', \mathsf{h}, \mathsf{r}$) Pour obtenir une collision entre le message m, avec l'aléa r, et le message m' sur le même condensé h, on calcule

$$\mathsf{r}' = \mathsf{csk}(\mathsf{m} - \mathsf{m}') + \mathsf{r}.$$

$$\mathsf{r}' \leftarrow f\mathcal{HC}.\textsc{Forge}(\mathsf{csk}, \mathsf{m}, \mathsf{m}', \mathsf{h}, \mathsf{r})$$

L'attaque "par exposition de clé" pour ce schéma se déroule de la façon suivante.

Soit $\mathcal{A}$ un adversaire connaissant (m, r) et $(\mathsf{m}', \mathsf{r}')$ relié au même condensé h. Nous avons $\mathsf{h} = \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}, \mathsf{r}) = \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}', \mathsf{r}')$. Dans ce schéma, les aléas sont reliés par l'équation $\mathsf{r}' = \mathsf{csk}(\mathsf{m} - \mathsf{m}') + \mathsf{r}$. $\mathcal{A}$ peut ainsi retrouver la clé secrète de la fonction en posant :

$$\mathsf{csk} = \frac{\mathsf{r}' - \mathsf{r}}{\mathsf{m} - \mathsf{m}'}.$$

De nombreuses solutions ont été développées depuis. Il existe aujourd'hui des solutions, comme celles présentées par G.Ateniese et B. de Medeiros dans [Ad04], prouvées sûres dans le modèle de l'oracle aléatoire. Mais aussi quelques constructions, comme la solution proposée par D.Catalano, R.Gennaro, N.Howgrave-Graham, et P.Q.Nguyen dans [CGHGN01] sous le nom "d'engagement à trappe", qui sont prouvées sûres dans le modèle standard.

Le choix du schéma de fonction caméléon dépendra donc des hypothèses de sécurité considérées et du modèle souhaité. Dans ce mémoire, lorsque j'utilise une fonction de hachage caméléon, je suppose que l'on choisit un schéma sûr mais je n'impose pas d'instanciation précise. Afin de montrer le fonctionnement d'un tel schéma, je donne ici un exemple d'une telle fonction, qui a été prouvée sûre sous l'hypothèse du logarithme discret, .

Construction 2 (Schéma de fonction de hachage caméléon [Ad04]).
Soient λ un paramètre de sécurité et $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$, avec λ un paramètre de sécurité, une fonction de hachage cryptographiquement sûre (voir Définition 1.3.2). Le schéma fonctionne de la manière suivante.

$\mathcal{HC}$.Init(λ) On choisit p premier tel que $p = 2q + 1$ avec q premier et on prend g un générateur du sous-groupe des résidus quadratiques de $\mathbb{Z}_p^*$, c'est-à-dire dans le sous-groupe des éléments qui sont un carré dans $\mathbb{Z}_p^*$. g est donc d'ordre q.

$$\mathcal{HC}.\mathsf{param} = \{g, p, \lambda\} \leftarrow \mathcal{HC}.\textsc{Init}(\lambda)$$

$\mathcal{HC}$.GénClé($\mathcal{HC}.\mathsf{param}$) La clé secrète csk est choisie aléatoirement dans $[1, q-1]$, la clé publique correspondante est $\mathsf{cpk} = \{g^{\mathsf{csk}}, \mathcal{HC}.\mathsf{param}\}$. Par abus de langage, on dira cpk pour parler de g^{csk}.

$$(\mathsf{csk}, \mathsf{cpk}) \leftarrow \mathcal{HC}.\textsc{GénClé}(\mathcal{HC}.\mathsf{param})$$

$\mathcal{HC}$.Hache($\mathsf{cpk}, \mathsf{m}, \mathsf{r}, \mathcal{HC}.\mathsf{param}$) Dans ce schéma, l'aléa $\mathsf{r} = (\mathsf{r}_1, \mathsf{r}_2)$ a été choisit dans $\mathbb{Z}_q \times \mathbb{Z}_q$. Le condensé correspondant au message m et à l'aléa r pour la clé publique cpk est obtenu comme suit.

$$e = \mathcal{H}(\mathsf{m}, \mathsf{r}_1) \qquad \text{et} \qquad \mathsf{h} = \mathsf{r}_1 - (\mathsf{cpk}^e g^{\mathsf{r}_2} \bmod p) \bmod q$$

$$\mathsf{h} \leftarrow \mathcal{HC}.\textsc{Hache}(\mathsf{cpk}, \mathsf{m}, \mathsf{r})$$

$\mathcal{HC}$.Forge($\mathsf{csk}, \mathsf{m}, \mathsf{m}', \mathsf{h}, \mathsf{r}$) Pour obtenir une collision entre le message m, avec l'aléa r, et le message m' sur le condensé h, on choisit aléatoirement $k' \in [1, q-1]$ puis on calcule

$$\mathsf{r}'_1 = C + (g^{k'} \bmod p) \bmod q, \qquad e' = \mathcal{H}(\mathsf{m}', \mathsf{r}'_1) \qquad \text{et} \qquad \mathsf{r}'_2 = k' - e'\mathsf{csk} \bmod q$$

Nous avons alors :

$$\begin{aligned} \mathsf{r}'_1 - (\mathsf{cpk}^{e'} g^{\mathsf{r}'_2} \bmod p) \bmod q &= C + (g^{k'} \bmod p) - (g^{e'\mathsf{csk}} g^{\mathsf{r}'_2} \bmod p) \bmod q \\ &= C \end{aligned}$$

$$\mathsf{r}' = (\mathsf{r}'_1, \mathsf{r}'_2) \leftarrow \mathcal{HC}.\textsc{Forge}(\mathsf{csk}, \mathsf{m}, \mathsf{m}', \mathsf{h}, \mathsf{r})$$

⟲

1.3.4 Fonction et générateur pseudo-aléatoire

Nous allons à présent nous intéresser à un autre type de fonction à sens unique : les générateurs et les fonctions pseudo-aléatoires.

Définition 23 (Générateur pseudo-aléatoire).
On appelle générateur pseudo-aléatoire, noté $\mathcal{PRG}$, toute fonction de $\{0,1\}^k$ dans $\{0,1\}^n$ pour $n > k$ ayant les deux propriétés suivantes.

- **À sens unique**. $\mathcal{PRG}$ doit être une fonction facile à évaluer mais difficile à inverser comme défini en Définition 20.
- **Pseudo-aléatoire**. Il doit être calculatoirement impossible de distinguer une sortie de $\mathcal{PRG}$ d'une chaîne de n bits aléatoires.
  Plus formellement, l'expérience correspondant à cette propriété définit un adversaire $\mathcal{A}$ face à un oracle $\mathcal{PRG}.\textsc{Challenge}_b$ qui retourne une sortie de $\mathcal{PRG}$ si $b = 0$ et une chaîne aléatoire sinon. À la fin de l'expérience, l'adversaire retourne un bit b^*, son avantage dans cette expérience est :
  $$\mathsf{Adv}^{\mathcal{PRG}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda) = |Pr[b = b^*] - 1/2|.$$
  Un générateur pseudo-aléatoire $\mathcal{PRG}$ est dit **pseudo-aléatoire** si l'avantage $\mathsf{Adv}^{\mathcal{PRG}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda)$ de tout adversaire polynomial $\mathcal{A}$ est négligeable.
  $$\mathsf{Adv}^{\mathcal{PRG}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.}$$ ∗

Définition 24 (Fonction pseudo-aléatoire).
Une fonction pseudo-aléatoire, noté $\mathcal{PRF}$, est une fonction déterministe qui associe à tout couple $(\kappa, \mathsf{m}) \in \{0,1\}^k \times \{0,1\}^k$ un élément dans $\{0,1\}^n$. On appelle κ la clé secrète ou la graine de la fonction pseudo-aléatoire. Une telle fonction doit suivre les deux propriétés suivantes.

- **À sens unique**. $\mathcal{PRF}$ doit être une fonction facile à évaluer mais difficile à inverser en ses deux variables, de manière similaire à la Définition 20.
- **Pseudo-aléatoire**. Même en connaissant m, il doit être calculatoirement impossible de distinguer une sortie de $\mathcal{PRF}(*, \mathsf{m})$ d'une chaîne de n bits aléatoires.
  Plus formellement, l'expérience qui correspond à cette propriété définit un adversaire $\mathcal{A}$ face à un oracle $\mathcal{PRG}.\textsc{Challenge}_b(\mathsf{m})$ qui retourne une sortie de $\mathcal{PRG}$ sur le message m si $b = 0$ et une chaîne aléatoire sinon. À la fin de l'expérience, l'adversaire retourne un bit b^*, son avantage dans cette expérience est :

  $$\mathsf{Adv}^{\mathcal{PRF}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda) = |Pr[b = b^*] - 1/2|.$$

  Une fonction pseudo-aléatoire $\mathcal{PRF}$ est dit **pseudo-aléatoire** si l'avantage $\mathsf{Adv}^{\mathcal{PRF}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda)$ de tout adversaire polynomial $\mathcal{A}$ est négligeable.

  $$\mathsf{Adv}^{\mathcal{PRF}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.} \qquad *$$

Le détail de ces deux outils ne sera pas abordé dans ce mémoire, pour une information plus complète et quelques exemples, le lecteur pourra se référer à [MvV97].

1.3.5 Engagement

Un schéma d'engagement permet à un prouveur de mettre en gage une valeur sans la révéler. Une fois engagée la valeur ne peut plus être modifiée mais elle pourra, si besoin, être révélée de telle sorte qu'il sera publiquement possible de s'assurer qu'il s'agit bien de la valeur ayant préalablement été engagée. Plus formellement un schéma d'engagement est défini de la façon suivante.

Définition 25 (Engagement).
Un schéma d'engagement $\mathcal{E}ng$ est défini par les trois algorithmes suivants.

- $\mathcal{E}ng.\textsc{Init}$ prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système $\mathcal{E}ng.\mathsf{param}$.

  $$\mathcal{E}ng.\mathsf{param} \leftarrow \mathcal{E}ng.\textsc{Init}(1^\lambda)$$

- $\mathcal{E}ng.\textsc{Engage}$ permet d'engager une valeur m. Il renvoie un engagement C et un ensemble d'aléas R permettant de vérifier la validité de l'engagement.

  $$(C, R) \leftarrow \mathcal{E}ng.\textsc{Engage}(\mathsf{m}, \mathcal{E}ng.\mathsf{param})$$

- $\mathcal{E}ng.\textsc{Vérifie}$ permet de s'assurer de la validité d'un triplet valeur m, engagement C, aléas R. Cet algorithme retourne vrai si le triplet est valide et faux sinon.

  $$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{E}ng.\textsc{Vérifie}(\mathsf{m}, C, R)$$

Un protocole d'engagement est sûr s'il suit les deux propriétés suivantes.

- **Indistinguabilité**. (*hiding*) Il doit être infaisable d'obtenir de l'information sur le message m à partir de l'engagement C. Plus formellement, un ensemble d'engagement sur une valeur x doit être parfaitement ou calculatoirement indistinguable d'un ensemble d'engagements sur une valeur x', $x \neq x'$.
- **Résistance aux collisions**. (*binding*) La valeur m ne doit pas pouvoir être modifiée après avoir été engagée. Plus formellement, nous avons que pour tout message m et pour toute paire (C, R), il doit être parfaitement ou calculatoirement impossible de trouver $\mathsf{m}' \neq \mathsf{m}$ et R' tels que $\mathcal{E}ng.\textsc{Vérifie}(\mathsf{m}, C, R) = \mathcal{E}ng.\textsc{Vérifie}(\mathsf{m}', C, R') = \mathsf{vrai}$. *

Il est à noter qu'un engagement ne peut être à la fois parfaitement **indistinguable** et parfaitement **résistant aux collisions**.

Je vais à présent donner l'exemple d'un des engagements les plus utilisés, l'engagement introduit par T.P.Pedersen dans [Ped92].

Construction 3 (Engagement de Pedersen).
L'engagement de Pedersen est composé des trois algorithmes suivants.

- $\mathcal{E}ng.\textsc{Init}(1^\lambda)$ L'algorithme choisit un groupe cyclique $\mathbb{G}$ d'ordre premier p. Il choisit ensuite aléatoirement deux générateurs g et h de $\mathbb{G}$ et retourne les paramètres du système $\mathcal{E}ng.\mathsf{param} = (\lambda, g, h, \mathbb{G})$.
- $\mathcal{E}ng.\textsc{Engage}(\mathsf{m}, \mathcal{E}ng.\mathsf{param})$ Le prouveur utilise cet algorithme pour s'engager sur le message $\mathsf{m} \in \mathbb{Z}_p$. Pour cela il choisit aléatoirement $r \in \mathbb{Z}_p$ et calcule $C = g^{\mathsf{m}} h^r$. Enfin, l'algorithme retourne (C, r).
- $\mathcal{E}ng.\textsc{Vérifie}(\mathsf{m}, C, r)$ Le vérifieur s'assure que l'égalité $C = g^{\mathsf{m}} h^r$ est vérifiée. Si c'est le cas il retourne vrai sinon il retourne faux. ↻

Les engagements extractibles sont une variante des schémas d'engagement pour lesquels il existe une clé permettant d'ouvrir un engagement et d'obtenir le message qui y est contenu.

Définition 26 (Engagement extractibles).
Un schéma d'engagement extractible $\mathcal{EE}$ est défini par les trois algorithmes d'un protocole d'engagement, le protocole d'initialisation retournant en plus une paire de clés $(\mathsf{sk}, \mathsf{pk})$, auxquels on ajoute un algorithme d'ouverture.

- $\mathcal{EE}.\textsc{Init}$ prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système $\mathcal{E}ng.\mathsf{param}$ ainsi qu'une paire de clés $(\mathsf{sk}, \mathsf{pk})$. Nous considérons dans la suite que la clé publique est comprise dans les paramètres du schéma.

$$(\mathcal{E}ng.\mathsf{param}, \mathsf{sk}) \leftarrow \mathcal{E}ng.\textsc{Init}(1^\lambda)$$

- $\mathcal{EE}.\textsc{Engage}$ et $\mathcal{EE}.\textsc{Vérifie}$ sont identiques aux algorithmes d'un schéma d'engagement.
- $\mathcal{EE}.\textsc{Ouvre}$ prend en entrée un engagement C ainsi que la clé secrète du schéma sk et retourne la valeur m engagée dans C.

$$\mathsf{m} \leftarrow \mathcal{EE}.\textsc{Ouvre}(\mathsf{sk}, C)$$

Un tel protocole doit suivre des propriétés similaires à un protocole d'engagement afin d'être considéré comme sûr. ∗

1.3.6 Accumulateur

Un schéma d'accumulateur $\mathcal{ACC}$ permet d'agréger un ensemble de valeurs $\mathsf{V} = \{\mathsf{v}_1, \cdots, \mathsf{v}_n\}$ dans une seule valeur courte Acc que l'on nomme l'accumulateur de l'ensemble V. De plus, il permet de prouver qu'une valeur v_i appartient à l'ensemble V des valeurs accumulées. Pour cela, il utilise un témoin t_i relié à la valeur à prouver v_i et à l'accumulateur de l'ensemble Acc. L'algorithme publique $\mathcal{ACC}.\textsc{Teste}$ permet de vérifier si la valeur à bien été accumulée dans Acc en calculant l'accumulateur correspondant à la valeur v_i et à son témoin t_i. En pratique, il suffit de tester s'il s'agit bien de l'accumulateur Acc.

$$\mathsf{Acc} = \mathcal{ACC}.\textsc{Teste}(\mathsf{v}_i, \mathsf{t}_i)$$

Plus formellement, un schéma d'accumulateur est défini comme suit.

Définition 27 (Accumulateur).
Un schéma d'accumulateur, noté $\mathcal{ACC}$, est composé des cinq algorithmes suivants.

> $\mathcal{ACC}$.Gén prend en entrées un paramètre de sécurité λ et, éventuellement, un entier n indiquant le nombre maximum de valeurs qui pourront être accumulées. Il retourne une paire de clés d'accumulation $(\mathsf{ask}, \mathsf{apk})$, un accumulateur vide Acc_0 et, éventuellement, une donnée supplémentaire d'état de l'accumulateur etat_0 indiquant, par exemple, les valeurs contenus dans Acc_0.
>
> $$((\mathsf{ask}, \mathsf{apk}), \mathsf{Acc}_0, \mathsf{etat}_0) \leftarrow \mathcal{ACC}.\text{Gén}(\lambda, n)$$
>
> $\mathcal{ACC}$.AjouteEns prend en entrées la clé secrète de l'accumulateur ask, un accumulateur Acc pouvant être vide, un ensemble fini de valeurs $\mathsf{V} = \{\mathsf{v}_1, \cdots, \mathsf{v}_k\}$ à ajouter dans Acc et, éventuellement, une donnée supplémentaire d'état de l'accumulateur etat. Il retourne l'accumulateur Acc' mis à jour, l'ensemble des témoins $\mathcal{T} = \{\mathsf{t}_i, \forall i \in [1, k]\}$ correspondant aux valeurs de l'ensemble V et, s'il est défini, le nouvel état de l'accumulateur etat'.
>
> $$(\mathsf{Acc}', \mathcal{T}, \mathsf{etat}') \leftarrow \mathcal{ACC}.\text{AjouteEns}(\mathsf{ask}, \mathsf{Acc}, \mathsf{V}, \mathsf{etat}, \mathsf{aux})$$
>
> $\mathcal{ACC}$.Teste prend en entrées une valeur v et un témoin t et retourne l'accumulateur Acc correspondant à ceux-ci.
>
> $$\mathsf{Acc} \leftarrow \mathcal{ACC}.\text{Teste}(\mathsf{v}, \mathsf{t})$$
>
> *Remarque.* Cet algorithme peut être utilisé pour vérifier qu'un témoin et une valeur appartiennent à un accumulateur pressenti. Pour cela il suffi de vérifier l'égalité entre ce dernier et la sortie de l'algorithme public $\mathcal{ACC}$.Teste.

Un tel protocole est considéré comme sûr s'il respecte la propriété de **résistance aux collisions** définie comme suit.

– **Résistance aux collisions**. Soit Acc un accumulateur sur un ensemble de valeurs $\mathsf{V} = \{\mathsf{v}_1, \cdots, \mathsf{v}_n\}$. Il doit être impossible de trouver, avec une probabilité plus que négligeable, un couple $(\mathsf{v}^*, \mathsf{t}^*)$ tel que $\mathsf{v}^* \notin \mathsf{V} = \{\mathsf{v}_1, \cdots, \mathsf{v}_n\}$ et $\mathcal{ACC}.\text{Teste}(\mathsf{v}^*, \mathsf{t}^*) = \mathsf{Acc}$.

*

J'ai choisi de donner ici la définition la plus générale. Il est à noter cependant qu'il existe des schémas pour lesquels certains paramètres sont vides, voir même pour lesquels la clé secrète $\mathsf{ask} = \perp$.

De nombreux travaux ont été effectués sur les accumulateurs [ASM08, CHKO08, ATSM09, CKS09, Tar09] permettant la construction d'accumulateurs reposants sur différentes hypothèses et proposant des solutions pour chacun des deux environnements que nous considérons dans ce mémoire : le modèle de l'oracle aléatoire et le modèle standard.

Conclusion

J'ai décrit dans ce chapitre l'ensemble des outils de base que nous utiliserons dans ce mémoire. Nous allons à présent nous intéresser aux outils classiques permettant de répondre aux objectifs majeurs de la cryptographie : garantir la confidentialité des communications et assurer l'intégrité et l'authenticité des messages.

Chapitre 2

Outils cryptographiques avancés

Sommaire

Historiquement, la cryptographie a pour but de définir et d'étudier les méthodes permettant d'assurer la confidentialité des messages, c'est-à-dire mettre en place des systèmes permettant à un émetteur d'envoyer un message à un destinataire de telle sorte que toute personne l'interceptant soit dans l'incapacité d'accéder à son contenu. La cryptographie est aujourd'hui plus diversifiée, elle propose, entre autres, des outils permettant de garantir l'intégrité et l'authentification. Ainsi, un émetteur peut envoyer un message à un destinataire de façon à ce que celui-ci soit non seulement convaincu que le message n'a pas été modifié au cours du transfert mais aussi que le message provient bien de l'émetteur pressenti.

Ce chapitre introduit les principales notions utilisées en cryptographie moderne pour répondre, de manière générique, à ces deux problématiques. Nous insisterons principalement sur les méthodes qui seront réutilisées dans la suite de ce mémoire.

Nous nous intéressons, dans un premier temps, à la confidentialité, c'est-à-dire aux moyens de s'assurer que seul l'émetteur et le ou les destinataire(s) légitime(s) d'un message en connaissent

le contenu. Pour répondre à cette problématique, la cryptographie utilise les schémas de chiffrement. Les premiers schémas de chiffrement furent des schémas symétriques. Dans ce type de schéma, l'émetteur et le destinataire du message utilisent une clé secrète commune pour chiffrer et déchiffrer un message. Ces schémas sont généralement très efficaces. Cependant, ils nécessitent que les deux parties se soient mises d'accord, au préalable, sur la clé commune. En 1976, W.Diffie et M.E.Hellman proposèrent, dans [DH76], le premier schéma de chiffrement à clé publique. Dans ce cas, le (futur) destinataire est muni d'une paire de clés, une clé secrète et une clé qu'il publie. L'émetteur utilise la clé publique pour chiffrer les messages tandis que le destinataire utilisera sa clé secrète pour les déchiffrer.

2.1 Confidentialité

Dans le contexte de ce mémoire, nous nous intéresserons uniquement aux schémas de chiffrement à clé publique.

2.1.1 Principe et définition

Afin de garantir la confidentialité des documents, un schéma de chiffrement à clé publique doit respecter un ensemble de "critères" de sécurité. De manière informelle, un tel schéma doit (au minimum) respecter les deux critères suivants.

- Un chiffré correctement formé doit pouvoir être correctement déchiffré par le destinataire.
- Il doit être difficile de retrouver un message à partir de son chiffré en ne connaissant pas la clé secrète de déchiffrement.

De façon plus formelle, un schéma de chiffrement à clé publique se présente sous la forme suivante.

Définition 28 (Schéma de chiffrement à clé publique).
Un schéma de chiffrement, noté $\mathcal{C}h$, est composé de quatre algorithmes.

- $\mathcal{C}h$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système qu'on notera $\mathcal{C}h.\mathsf{param}$. Nous considérerons dans la suite que λ est inclus dans les paramètres $\mathcal{C}h.\mathsf{param}$ du schéma.

$$(\mathcal{C}h.\mathsf{param}) \leftarrow \mathcal{C}h.\textsc{Init}(1^\lambda)$$

- $\mathcal{C}h$.GénClé prend en entrées les paramètres du système $\mathcal{C}h.\mathsf{param}$ et génère une paire de clés $(\mathsf{cpk}, \mathsf{csk})$.

$$(\mathsf{cpk}, \mathsf{csk}) \leftarrow \mathcal{C}h.\textsc{GénClé}(\mathcal{C}h.\mathsf{param})$$

- $\mathcal{C}h$.Chiffre est l'algorithme de chiffrement. Il prend en entrées les paramètres publics $\mathcal{C}h.\mathsf{param}$, un message m et la clé publique cpk. Il retourne un chiffré c du message m.

$$\mathsf{c} \leftarrow \mathcal{C}h.\textsc{Chiffre}(\mathcal{C}h.\mathsf{param}, \mathsf{m}, \mathsf{cpk})$$

- $\mathcal{C}h$.Déchiffre est l'algorithme de déchiffrement. Il prend en entrées les paramètres publics du schéma $\mathcal{C}h.\mathsf{param}$, un chiffré c et la clé secrète csk. Il retourne le clair m correspondant au chiffré c.

$$\mathsf{m} \leftarrow \mathcal{C}h.\textsc{Déchiffre}(\mathcal{C}h.\mathsf{param}, \mathsf{c}, \mathsf{csk}) \qquad *$$

2.1.2 Notions de sécurité

Un schéma de chiffrement doit être capable de résister à certaines attaques pour être considéré comme sûr. Différents niveaux de sécurité existent pour ce type de schéma. Ceux-ci sont habituellement définis en fonction des attaques auxquelles il résiste. Ces attaques sont caractérisées selon deux critères : l'objectif et les moyens de l'attaquant. Le niveau de sécurité le plus élevé sera obtenu lorsque l'attaquant disposera des moyens les plus importants dans l'objectif d'effectuer l'attaque la plus facile. Nous allons d'abord voir les différents objectifs possibles pour l'attaquant, puis nous donnerons les moyens dont il peut disposer pour son attaque.

Objectifs de l'adversaire

L'objectif général de l'attaquant est de retrouver le texte clair, ou au moins, d'obtenir de l'information sur celui-ci. L'objectif à atteindre pour considérer une attaque comme réussie peut être plus ou moins difficile : retrouver la clé du destinataire, retrouver un message clair à partir d'un chiffré, ou "simplement" distinguer les chiffrés de deux messages différents. Plus précisément, les quatre niveaux d'objectifs les plus courants sont les suivants.

Cassage total. L'attaquant doit réussir à obtenir la clé secrète du destinataire attaqué. Il pourra ainsi déchiffrer n'importe quel message.
On note habituellement **UBK** cet objectif pour *Unbreakability*.

Inverser le chiffrement. L'attaquant doit arriver à simuler de manière efficace l'algorithme de déchiffrement. Il pourra ainsi déchiffrer les messages qu'il souhaite sans pour autant connaître la clé secrète du destinataire.
Ce but consiste à casser la propriété de fonction **à sens unique** du schéma de chiffrement.
On note habituellement cet objectif **OW** pour *One-Wayness* en anglais.

Distinguer deux chiffrés. L'attaquant choisit deux messages m_0 et m_1, il doit ensuite parvenir à retrouver, avec une probabilité significativement meilleure que $1/2$, si un chiffré c_b est un chiffré du message m_0 ou un chiffré du message m_1.
Ici l'adversaire souhaite attaquer la propriété d'**Indistinguabilité** du schéma, on note **IND** ce but. Il est à noter que cette propriété est aussi connue sous le nom de **Sécurité sémantique**.

Modifier un chiffré. L'attaquant doit réussir, à partir d'un chiffré c d'un message m qu'il ne connait pas, à générer un chiffré c' sur un message m' dont il peut ne pas connaître la valeur mais qui soit tel que m et m' soient différents mais reliés entre-eux. Le chiffrement Elgamal que je décris en section suivante, construction 4, est un exemple de schéma malléable. Pour cette construction, nous verrons qu'un attaquant peut générer le chiffré correspondant à un message m' relié au message initial m par la relation $\mathsf{m}' = x\mathsf{m}$ pour n'importe quel x de son choix.
Cet objectif consiste à attaquer la propriété de **Non-Malléabilité** du schéma, un tel objectif est ainsi noté **NM**.
Un exemple de schéma *malléable* est le schéma de chiffrement Elgamal que nous verrons en section 2.1.3.

Moyens de l'adversaire

Les moyens de l'attaquant définissent la puissance de celui-ci. Les trois niveaux généralement utilisés sont les suivants.

Attaque à clairs choisis. L'adversaire peut obtenir les messages chiffrés correspondant aux messages clairs de son choix. Cette attaque est toujours possible dans le cadre des

chiffrements à clé publique.

On note ce niveau de moyen **CPA** pour *Chosen Plaintext Attack*.

Attaque à chiffrés choisis. L'adversaire a accès à un oracle de déchiffrement lui permettant d'obtenir les messages clairs correspondants aux chiffrés de son choix jusqu'au moment où il accède au chiffré challenge.

Ce niveau de moyen est noté **CCA1** pour *Chosen Ciphertext Attack*.

Attaque à chiffrés choisis adaptative. L'adversaire a accès à un oracle de déchiffrement, pendant toute la durée de son attaque, lui permettant d'obtenir les messages clairs correspondants aux chiffrés de son choix (excepté sur le chiffré challenge). On dit que cette attaque est adaptative car l'adversaire peut adapter ses requêtes à tout moment durant son attaque.

Ce niveau de moyen est noté **CCA2** pour *Chosen Ciphertext Adaptative Attack*.

Conclusion

Nous venons de voir les deux critères définissant l'attaque d'un adversaire : son objectif et ses moyens. La sécurité obtenue par un schéma est défini par un couple $\mathbf{objectif - moyen}$ avec $\mathbf{objectif} \in \{\mathbf{UBK}, \mathbf{OW}, \mathbf{NM}, \mathbf{IND}\}$ et $\mathbf{moyen} \in \{\mathbf{CPA}, \mathbf{CCA1}, \mathbf{CCA2}\}$. Ce couple représente le fait qu'un adversaire contre le schéma est incapable de réussir l'attaque ayant pour but **objectif** même avec les ressources **moyen** (en général, sous l'hypothèse qu'un problème donné soit difficile).

Par exemple, un schéma de chiffrement indistinguable contre des attaques à chiffrés choisis adaptatives sera dit $\mathbf{IND - CCA2}$.

Il est à noter qu'il existe des techniques de transformation, notamment [FO99b, FO99a, Poi00, OP01], qui permettent d'obtenir un schéma résistant aux attaques **CCA1** ou **CCA2** à partir d'un schéma seulement résistant aux attaques **CPA**.

De plus, les différentes propriétés sont reliées entre elles comme présenté en Figure 2.1. Ces implications ont été démontrées dans [BDPR98] par M.Bellare, A.Desai, D.Pointcheval et P.Rogaway.

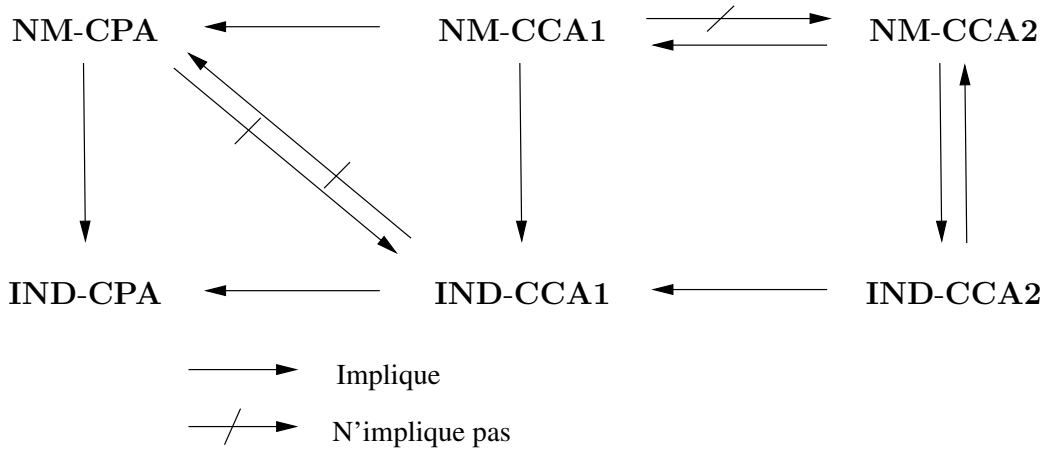

FIGURE 2.1 – Relations entre les notions de sécurité

2.1.3 Exemple : chiffrement Elgamal

Le chiffrement dit Elgamal a été introduit par T.Elgamal dans [ElG85], il s'agit du premier schéma de chiffrement basé sur le logarithme discret. En effet, ce schéma est **IND − CPA**, sous l'hypothèse DDH et simplement **OW − CPA** sous l'hypothèse CDH.

Construction 4 (Schéma de chiffrement Elgamal).
Le schéma de chiffrement Elgamal est composé des quatre algorithmes suivants.

- $\mathcal{C}h$.Init(1^λ). L'algorithme choisit un groupe cyclique $\mathbb{G}$ d'ordre premier p. Il choisit ensuite aléatoirement un générateur g de $\mathbb{G}$ et retourne les paramètres du système $\mathcal{C}h.\mathsf{param} = (\lambda, g, \mathbb{G})$.
- $\mathcal{C}h$.GénClé($\mathcal{C}h.\mathsf{param}$). Cette procédure choisit aléatoirement la clé secrète csk dans $\mathbb{Z}_p^*$ et calcule la clé publique $\mathsf{cpk} = g^{\mathsf{csk}} \in \mathbb{G}$.
- $\mathcal{C}h$.Chiffre($\mathcal{C}h.\mathsf{param}, \mathsf{m}, \mathsf{cpk}$). L'algorithme de chiffrement choisit aléatoirement $r \in \mathbb{Z}_p^*$, il calcule ensuite
$$\mathsf{c}_1 = \mathsf{m} \cdot \mathsf{cpk}^r \quad \text{et} \quad \mathsf{c}_2 = g^r.$$
Le chiffré du message m pour la clé publique cpk est $\mathsf{c} = (\mathsf{c}_1, \mathsf{c}_2)$.
- $\mathcal{C}h$.Déchiffre($\mathcal{C}h.\mathsf{param}, \mathsf{c}, \mathsf{csk}$). Le déchiffrement de c est obtenu en calculant :
$$\mathsf{m} = \frac{\mathsf{c}_1}{\mathsf{c}_2^{\mathsf{csk}}}.$$

*

Ce schéma de chiffrement est malléable. En effet, supposons que nous ayons le chiffré $\mathsf{c} = (\mathsf{c}_1, \mathsf{c}_2)$ correspondant au message clair m, il est alors possible de générer un nouveau chiffré sur un message relié à m, et cela sans connaitre m. Il suffit de choisir x et de calculer $\mathsf{c}'_1 = x\mathsf{c}_1$. On obtient ainsi un chiffré valide $\mathsf{c}' = (\mathsf{c}'_1, \mathsf{c}_2)$ sur le message $\mathsf{m}'(= x\mathsf{m})$ relié à m.

La malléabilité de ce schéma ne lui permet pas d'être résistant à une attaque à chiffrés choisis contre l'indistinguabilité. Une variante de ce schéma, le schéma "double chiffrement Elgamal" (Construction 5), permet d'atteindre cette sécurité, y compris contre les attaques adaptatives, sous l'hypothèse DDH. Pour cela, il propose de chiffrer le message à deux reprises, avec deux clés distinctes, et d'y ajouter une preuve que l'émetteur connait les aléas utilisés pour ces chiffrements. La preuve de sécurité correspondante à été proposée par P.-A.Fouque et D.Pointcheval dans [FP01]. Plus formellement, le schéma en lui-même fonctionne de la façon suivante.

Construction 5 (Schéma du double chiffrement Elgamal).
Le schéma de double chiffrement Elgamal est composé des quatre algorithmes suivants.

- $\mathcal{C}h$.Init(1^λ). L'algorithme choisit un groupe cyclique $\mathbb{G}$ d'ordre premier p. Il choisit ensuite aléatoirement un générateur g de $\mathbb{G}$ et retourne les paramètres du système $\mathcal{C}h.\mathsf{param} = (\lambda, g, \mathbb{G})$.
- $\mathcal{C}h$.GénClé($\mathcal{C}h.\mathsf{param}$). Cette procédure choisit aléatoirement x et y dans $\mathbb{Z}_p^*$, on pose $\mathsf{csk} = (x, y)$ la clé secrète. La clé publique cpk correspondante est composée de deux parties $h_x = g^x \in \mathbb{G}$ et $h_y = g^y \in \mathbb{G}$. On note $\mathsf{cpk} = (h_x, h_y)$
- $\mathcal{C}h$.Chiffre($\mathcal{C}h.\mathsf{param}, \mathsf{m}, \mathsf{cpk}$). Le chiffrement du message m correspond à deux chiffrement Elgamal avec deux aléas différents. L'algorithme de chiffrement choisit aléatoirement $r, s \in \mathbb{Z}_p^*$ et calcule ensuite
$$\mathsf{c}_1 = \mathsf{m} \cdot h_x^r \quad , \quad \mathsf{c}_2 = g^r \quad \text{et} \quad \mathsf{c}_3 = \mathsf{m} \cdot h_y^s \quad , \quad \mathsf{c}_2 = g^s.$$

Le chiffré du message m pour la clé publique cpk est $\mathsf{c} = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \mathsf{c}_4, \pi)$ avec π une preuve qui assure que l'on connait r et s tels que le quadruplet soit bien formé. Ceci est une preuve simple à mettre en place, cependant ce type de preuve n'étant réellement introduite qu'au chapitre suivant, je ne rentre pas dans les détails ici.

– $\mathcal{C}h.\text{DÉCHIFFRE}(\mathcal{C}h.\mathsf{param}, \mathsf{c}, \mathsf{csk})$. Le déchiffrement de c est obtenu en calculant :

$$\mathsf{m}_1 = \frac{\mathsf{c}_1}{\mathsf{c}_2^x} \quad \text{et} \quad \mathsf{m}_2 = \frac{\mathsf{c}_3}{\mathsf{c}_4^y}.$$

Il vérifie que la preuve π est correcte et que $\mathsf{m}_1 = \mathsf{m}_2$

*

2.2 Authentification et Intégrité

Dans cette section nous nous intéressons aux moyens mis à notre disposition par la cryptographie pour assurer l'authentification et l'intégrité d'un message. L'authentification permet de garantir l'identité de l'auteur d'un message, tandis que l'intégrité permet de certifier qu'un message n'a pas été modifié entre l'émetteur et son destinataire.

La cryptographie à clé publique propose pour cela les schémas de signature. L'émetteur détient une paire de clés, une clé qu'il garde secrète et une qu'il publie. Il utilise ensuite sa clé secrète pour générer une signature d'un message. La validité de cette signature est publiquement vérifiable à l'aide de la clé publique de l'émetteur. L'intégrité du message et l'authentification de l'auteur est publique.

La cryptographie à clé secrète nous propose une solution différente : les codes d'authentification de message ou MAC. Ces schémas impliquent que l'émetteur et le destinataire partagent une même clé secrète. Un MAC est généré par l'émetteur et vérifié par le destinataire avec cette clé commune. Si le MAC du message est valide alors le message n'a pas été modifié. L'authentification se fait simplement par le fait que si le destinataire n'a pas généré un MAC donné alors c'est qu'il provient bien de l'émetteur. Dans ce cas seul l'émetteur et le destinataire peuvent vérifier l'intégrité du message et authentifier son auteur.

Les signatures étant au centre de ce mémoire et le MAC un simple outil nous nous concentrerons principalement sur ces premières et ne donnerons qu'une définition sommaire des MAC en fin de section.

2.2.1 Signature : principe et définition

La signature numérique est un mécanisme de cryptographie à clé publique permettant de s'assurer de l'identité de l'auteur d'un document et de garantir l'intégrité du message. Un schéma de signature doit ainsi respecter un ensemble de "critères" de sécurité.

– L'identité du signataire d'un message doit pouvoir être déterminée avec certitude.
– Il doit être impossible de construire une fausse signature du signataire sur un document.
– Un document signé ne doit plus pouvoir être modifié.
– Le signataire ne doit pas pouvoir nier qu'il est à l'origine d'une signature.

Plus formellement, un schéma de signature se présente sous la forme suivante.

Définition 29 (Signature).
Un schéma de signature $\mathcal{S}$ est défini par les quatre algorithmes suivants.

– $\mathcal{S}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il initialise le schéma et retourne les paramètres du système notés $\mathcal{S}.\mathsf{param}$. Nous considérerons dans la suite que λ est inclus dans les paramètres.

$$\mathcal{S}.\mathsf{param} \leftarrow \mathcal{S}.\textsc{Init}(1^\lambda)$$

– $\mathcal{S}$.GénClé permet de générer la paire de clés du signataire $(\mathsf{pk}, \mathsf{sk})$ en fonction des paramètres du système $\mathcal{S}.\mathsf{param}$. On suppose dans la suite que les paramètres $\mathcal{S}.\mathsf{param}$ sont inclus dans la clé publique du signataire pk.

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{S}.\textsc{GénClé}(\mathcal{S}.\mathsf{param})$$

– $\mathcal{S}$.Signe est l'algorithme de signature. Il prend en entrée la paire de clés du signataire $(\mathsf{pk}, \mathsf{sk})$ et un message m, il retourne une signature σ sur ce message m (ou $\perp$ en cas d'erreur).

$$\sigma \leftarrow \mathcal{S}.\textsc{Signe}(\mathsf{pk}, \mathsf{sk}, \mathsf{m})$$

– $\mathcal{S}$.Vérifie permet de vérifier une signature σ sur un message m en utilisant la clé publique du signataire pk. Il retourne vrai si la signature est valide et faux si elle ne l'est pas.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{S}.\textsc{Vérifie}(\mathsf{m}, \sigma, \mathsf{pk}) \qquad *$$

Tout schéma de signature doit satisfaire l'ensemble des critères de sécurité informels donnés en introduction pour être considéré comme sûr. Nous allons maintenant nous intéresser à une définition plus formelle de ce qu'est un schéma de signature "sûr".

2.2.2 Sécurité pour les signatures

De manière similaire aux schémas de chiffrement, un schéma de signature doit être capable de résister à certaines attaques pour être considéré comme sûr. Nous définissons ainsi la sécurité d'un schéma de signature selon la puissance et l'objectif de l'adversaire. Je donnerai ici les différents buts et objectifs possibles pour l'attaquant, comme proposés par S.Goldwasser, S.Micali et R.Rivest notamment dans [GMR85b].

Objectifs de l'adversaire

Le but à atteindre pour considérer une attaque comme réussie peut être plus ou moins difficile, cela peut être retrouver la clé du signataire à "simplement" générer une signature d'un message ne signifiant pas forcément quelque chose. Plus précisément, nous avons les quatre niveaux d'objectifs suivants.

Cassage total. L'attaquant doit réussir à obtenir la clé secrète du signataire. Il pourra ainsi signer n'importe quel message de son choix.
On note habituellement **UBK** cet objectif pour *Unbreakability*.

Falsification universelle. L'attaquant doit trouver un algorithme capable de simuler de manière efficace l'algorithme signature. Il pourra ainsi générer des signatures valides du signataire pour n'importe quel message de son choix sans connaître la clé secrète du signataire.
Cet objectif est habituellement noté **UF** en référence à son nom anglais *Universal Unforgeability*.

Falsification Sélective. L'attaquant choisi un message avant de commencer son attaque et doit ensuite réussir à construire une signature valide du signataire sur ce message.
Habituellement, cet objectif est noté **SU** en référence à *Selective Unforgeability*.

Falsification Existentielle. L'attaquant doit réussir à établir une signature valide du signataire sur un message (celui-ci pouvant n'avoir aucun sens).
On note généralement cet objectif **EU** pour *Existential Unforgeability*.

Moyens de l'adversaire

Nous allons voir à présent les types d'attaques les plus couramment utilisés dans le cas de la signature électronique.

Attaque sans message. L'adversaire n'a accès qu'aux paramètres publics du système et à la clé publique du signataire qu'il souhaite attaquer.
On note ce niveau de moyen **NMA** en référence à la version anglaise *No Message Attack*.

Attaque à message connu. L'adversaire dispose, en plus des paramètres publics et de la clé publique du signataire qu'il attaque, d'une liste de couples message-signature produits par le signataire.
Ce niveau de moyen est noté **KMA** pour *Known Message Attack*.

Attaque à message choisi. L'adversaire, en plus de connaître les paramètres publics du schéma ainsi que la clé publique du signataire cible, dispose d'un oracle de signataire lui permettant d'obtenir des signatures valides du signataire pour les messages de son choix.
Cette attaque est, par défaut, "adaptative" : l'adversaire peut choisir le prochain message à demander à l'oracle après avoir reçu la signature correspondante au message précédemment demandé. L'attaque est dite "non-adaptative" si l'attaquant dois demander en une seule fois les signatures qui correspondent aux messages qu'il a choisi par un unique appel à l'oracle.
On note **CMA** la force de l'adversaire dans ce cas, en référence à *Chosen Message Attack*.

Conclusion

Nous venons de voir que l'attaquant est défini sur deux critères : son objectif et ses moyens. La sécurité obtenue par un schéma est par un couple **objectif** $-$ **moyen** avec **objectif** $\in \{\mathbf{UBK}, \mathbf{UF}, \mathbf{SU}, \mathbf{EU}\}$ et **moyen** $\in \{\mathbf{NMA}, \mathbf{KMA}, \mathbf{CMA}\}$.

Par exemple, pour un schéma de signature résistant aux falsifications existentielles par attaques à messages choisis, qui est le niveau le plus élevé de sécurité possible ici, nous dirons que le schéma est $\mathbf{EU} - \mathbf{CMA}$.

De manière plus formelle, l'expérience qui correspond à la résistance aux falsifications existentielles par attaques à messages choisis, $\mathbf{EU} - \mathbf{CMA}$, définit un adversaire $\mathcal{A}$ connaissant la clé publique du signataire qu'il attaque ainsi que les paramètres publics du système. Celui-ci a accès à un oracle $\mathcal{O}.\textsc{Signe}(\mathsf{m})$ retournant une signature valide du signataire pour le message m. À la fin de l'expérience, l'adversaire retourne un couple message-signature (m, σ) et gagne l'expérience si σ est une signature valide du signataire du message m et qu'il n'a jamais demandé à l'oracle de signer le message m. Le succès de l'adversaire $\mathcal{A}$ est noté $\mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda)$.

Un schéma de signature $\mathcal{S}$ est dit $\mathbf{EU} - \mathbf{CMA}$ sûr si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU}-\mathbf{CMA},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU}-\mathbf{CMA},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.}$$

Remarque. Dans le cas du modèle de l'oracle aléatoire, l'adversaire aura toujours accès à un oracle supplémentaire $\mathcal{O}$.Hache représentant la fonction de hachage.

2.2.3 Exemple : signature Boneh-Boyen [BB04]

Nous allons voir à présent un des deux schémas de signature proposés par D.Boneh et X.Boyen dans [BB04]. Le schéma que je décris ici à la particularité d'être prouvé **EU − CMA** sûr, sans oracle aléatoire, sous l'hypothèse que le problème $q-$SDH (voir Définition 18) est difficile.

Construction 6 (Signature Boneh-Boyen [BB04]).
Le schéma de signature Boneh-Boyen $\mathcal{S}$ est défini par les quatre algorithmes suivants.

- $\mathcal{S}$.Init(1^λ). Cet algorithme génère un environnement bilinéaire $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e}, \psi)$ selon l'hypothèse $q-$SDH (voir Définition 18). Pour rappel, nous avons $\mathbb{G}_1$ et $\mathbb{G}_2$ deux groupes d'ordre premier p avec g_1 un générateur de $\mathbb{G}_1$ et g_2 un générateur de $\mathbb{G}_2$. Enfin, ψ est un isomorphisme tel que $g_1 = \psi(g_2)$. On note $\mathcal{S}.\mathsf{param} = \big(\lambda, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e}, \psi\big)$.
- $\mathcal{S}$.GénClé$(\mathcal{S}.\mathsf{param})$ Cette procédure choisit aléatoirement x et y dans $\mathbb{Z}_p^*$ et calcule $u = g_2^x \in \mathbb{G}_2$ et $v = g_2^y \in \mathbb{G}_2$. On note $z = \mathsf{e}(g_1, g_2) \in \mathbb{G}_T$. La clé secrète du signataire est $\mathsf{sk} = (x, y)$ et la clé publique correspondante $\mathsf{pk} = (\mathcal{S}.\mathsf{param}, u, v, z)$.
- $\mathcal{S}$.Signe$(\mathsf{pk}, \mathsf{sk}, \mathsf{m})$ est l'algorithme de signature. Il prend en entrée la paire de clés du signataire $(\mathsf{pk}, \mathsf{sk})$ et un message m. Le signataire choisit aléatoirement $r \in \mathbb{Z}_p^*$ tel que $x + \mathsf{m} + yr \neq 0 \bmod p$ et calcule :
$$A = g_1^{\frac{1}{x+\mathsf{m}+yr}}$$
Le signataire obtient la signature $\sigma = (A, r)$ du message m.
- $\mathcal{S}$.Vérifie$(\mathsf{m}, \sigma, \mathsf{pk})$ Le vérifieur s'assure que $\sigma = (A, r)$ est une signature du signataire, désigné par la clé publique $\mathsf{pk} = (\mathcal{S}.\mathsf{param}, u, v, z)$, du message m en vérifiant l'égalité suivante.
$$\mathsf{e}(A, u \cdot g_2^{\mathsf{m}} \cdot v^r) = z$$
Si l'égalité est valide, il retourne vrai, sinon il retourne faux. ↺

Il sera intéressant pour la suite de noter que D.Boneh et X.Boyen proposent, dans [BB04], une variante de cette solution, que nous appellerons *signature Boneh-Boyen faible*, qui est **EU − CMA** sûre uniquement contre les attaques non-adaptatives. Cette seconde solution, bien que moins intéressante au premier abord, est à l'origine de divers schémas ayant des propriétés particulières dont certaines de mes constructions que je décrirai dans ce mémoire. Ces deux solutions étant très proches l'une de l'autre, je ne donnerai ici que les différences notables avec la solution présentée. Ainsi, la solution "non-adaptative" diffère de celle que nous venons de voir uniquement dans le fait qu'elle n'utilise qu'une seule sous-clé secrète x (dont la valeur publique correspondante est toujours $u = g_2^x$). La variable y n'existant pas, v n'apparait pas non plus. Une signature d'un message m est alors l'unique élément de $\mathbb{G}_1$ suivant.

$$A = g_1^{\frac{1}{x+\mathsf{m}}}$$

La vérification d'une telle signature revient à tester la validité de la relation :

$$\mathsf{e}(A, u \cdot g_2^{\mathsf{m}}) = z.$$

2.2.4 MAC et HMAC

Comme nous l'avons vu en introduction, les fonctions de code d'authentification de message, noté MAC pour *Message Authentication Code*, permettent de garantir l'intégrité d'un message avec une architecture symétrique. L'émetteur et le destinataire partagent une clé secrète commune qu'ils utilisent pour générer et vérifier le code d'authentification du message.

Plus formellement un MAC se présente sous la forme suivante.

Définition 30 (Code d'authentification de message MAC).
Un code d'authentification de message est une fonction $\text{MAC} : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\}^{\lambda'}$, avec λ et λ' deux paramètres de sécurité. Par abus de langage, on appelle MAC du message m avec la clé secrète sk la valeur $\text{MAC}(\mathsf{sk}, \mathsf{m})$.

On dit d'une telle fonction qu'elle est **objectif − moyen** sûre si elle résiste à une attaque d'un adversaire ayant pour but **objectif** avec les moyens **moyen**. Les différents niveaux d'objectifs et de moyens possibles sont identiques à ceux définis pour les signatures en Section 2.2.2. Un adversaire aura ainsi pour objectif le cassage total noté **UBK**, la falsification universelle **UF**, la falsification sélective **SU** ou bien la falsification existentielle notée **EU** et effectuera une attaque sans message, notée **NMA**, une attaque à message connu **KMA** ou encore une attaque à message choisi **CMA**. ∗

Les constructions que nous utiliserons dans ce mémoire sont des MAC basés sur des fonctions de hachage cryptographiquement sûres, notés HMAC. Une telle fonction est considérée comme **objectif − moyen** sûre si elle atteint la propriété de **résistance aux collisions** comme définie pour les fonctions de hachage (voir Chapitre 1 Section 1.3.2), et qu'elle résiste à une attaque **objectif − moyen** comme définie plus haut.

Nous allons à présent nous intéresser à quelques schémas de signatures ayant des propriétés supplémentaires permettant de répondre à des problématiques plus complexes.

2.3 Signatures particulières

Aujourd'hui, de plus en plus de schémas de signature ne se limitent pas aux propriétés que nous avons décrites jusque là. Les signatures caméléons auxquelles nous consacrerons une partie de ce mémoire sont un de ces schémas de signature qui font "un peu plus" que simplement authentifier l'auteur d'un message et garantir au destinataire que le message n'a pas été modifié. Je vais brièvement présenter ici trois exemples de signatures particulières que nous utiliserons dans la suite de ce mémoire : les signatures dites "Camenisch-Lysyanskaya", les signatures de groupe et un type de signature de groupe particulières que l'on appelle signature de liste.

Pour la description de ces trois schémas, nous devons poser l'hypothèse de travail suivante.

Hypothèse de travail : Cette hypothèse de travail pose pour acquis deux concepts connus et largement utilisés en cryptographie à clé publique : les preuves de connaissances. Je donnerai ici uniquement un rapide aperçu de celles-ci, elles seront développés plus en détails au chapitre 3 qui leur est entièrement dédié.

Nous considérons qu'il existe des protocoles interactif (ou non) appelés *preuve de connaissance* qui permettent de prouver la connaissance de secrets suivant un prédicat et cela de telle

sorte qu'aucune information sur les secrets ne soit révélée (à part qu'ils suivent le prédicat). Nous notons un tel protocole comme suit.

$$\textsc{Pok}(\textbf{secrets} : \textbf{prédicat})$$

D'autre part, nous considérons qu'il existe des schémas de *signature de connaissance* permettant de signer un message m en tant que détenteur de secrets **secrets** suivant un prédicat **prédicat** tel qu'il ne dévoile aucune information sur les secrets détenus (à part le fait qu'ils suivent le prédicat). Nous notons un tel protocole comme suit.

$$\textsc{Sok}(\textbf{secrets} : \textbf{prédicat})(\mathsf{m})$$

2.3.1 Signatures Camenisch-Lysyanskaya

Les signatures "Camenisch-Lysyanskaya" [CL02, CL04], sont appelées ainsi en référence aux auteurs les ayant introduites : J.Camenisch et A.Lysyanskaya [CL04]. En plus des propriétés usuelles d'un schéma de signature, ces schémas doivent assurer les propriétés particulières suivantes.

1. Le signataire peut signer un engagement sans pour autant connaître les valeurs engagées de telle façon qu'il soit possible, par la suite, de construire une preuve que l'on connait une des valeurs engagées sans révéler ni la signature, ni l'ensemble des valeurs engagées.
2. Le signataire peut signer un message m décomposé en blocs $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_\ell$ de telle façon qu'il soit possible, par la suite, de construire une preuve que l'on connait la signature du message contenant le bloc m_i sans révéler ni la signature, ni le message complet.

De manière plus formelle un schéma de signature "Camenisch-Lysyanskaya" se déroule de la façon suivante.

Définition 31 (Schéma de signature Camenisch-Lysyanskaya).
Un schéma de signature "Camenisch-Lysyanskaya" $\mathcal{CL}$ est composé de six algorithmes. Un tel schéma suppose qu'un schéma d'engagement sûr comme défini en Définition 25 est utilisé pour générer les engagements (C, R) qui seront signés.

- $\mathcal{CL}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité et éventuellement une valeur t correspondant au nombre maximum de valeurs pouvant être intégrées dans un engagement que le signataire acceptera de signer. Il initialise les paramètres du système notés $\mathcal{CL}.\mathsf{param}$. Nous considérerons dans la suite que λ et t sont inclus dans les paramètres $\mathcal{CL}.\mathsf{param}$.
$$\mathcal{CL}.\mathsf{param} \leftarrow \mathcal{CL}.\textsc{Init}(1^\lambda, t)$$
- $\mathcal{CL}$.GénClé prend en entrée les paramètres du système $\mathcal{CL}.\mathsf{param}$ et génère la paire de clés du signataire $(\mathsf{sk}, \mathsf{pk})$. Nous considérerons dans la suite que $\mathcal{CL}.\mathsf{param}$ est inclus dans la clé publique pk.
$$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{CL}.\textsc{GénClé}(\mathcal{CL}.\mathsf{param})$$
- $\mathcal{CL}$.Signe est l'algorithme utilisé pour signer les messages. Il prend en entrée un message m composé de ℓ blocs $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_\ell$, la clé secrète du signataire sk et la clé publique du schéma clpk et retourne une signature σ (ou $\perp$ en cas d'erreur).
$$\sigma \leftarrow \mathcal{CL}.\textsc{Signe}(\mathsf{m}, \mathsf{sk}, \mathsf{clpk})$$

– $\mathcal{CL}$.CSigne est un algorithme permettant à un signataire de signer un engagement. Il prend en entrée un engagement C et une preuve qu'il est bien formé π_C, la clé secrète du signataire sk et la clé publique du schéma clpk et retourne une signature σ ainsi, qu'éventuellement, un ensemble d'aléas R' reliés à l'engagement (ou $\perp$ en cas d'erreur).

$$(\sigma, R') \leftarrow \mathcal{CL}.\textsc{Signe}(C, \pi_C, \mathsf{sk}, \mathsf{clpk})$$

– $\mathcal{CL}$.Prouve est un protocole, éventuellement non-interactif, de preuve de connaissance entre un prouveur et un vérifieur. Ce protocole permet au prouveur de convaincre le vérifieur qu'il connait une signature σ sur un engagement C dont il connait les valeurs mises en gages $\mathsf{m}_1, \cdots, \mathsf{m}_\ell$, sans divulguer ni les valeurs ni la signature. Ce protocole correspond à la preuve de connaissance suivante.

$$\textsc{Pok}\big(C, \mathsf{m}_1, \cdots, \mathsf{m}_\ell, \sigma, : \sigma = \mathcal{CL}.\textsc{CSigne}(c) \wedge \{\mathsf{m}_1, \cdots, \mathsf{m}_\ell\} \in C\big)$$

$$\mathcal{CL}.\textsc{Prouve}(\text{Vérifieur} : \mathsf{pk} \quad ; \quad \text{Prouveur} : \sigma, \mathsf{m}, C, R)$$

– $\mathcal{CL}$.Vérifie est un algorithme public permettant de vérifier une signature σ sur un message $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_\ell$ en utilisant la clé publique du signataire pk. Il retourne vrai si la signature est valide et faux sinon.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{CL}.\textsc{Vérifie}(\mathsf{pk}, \sigma, \mathsf{m}) \qquad *$$

Il n'existe pas de modèle propre aux signatures Camenisch-Lysyanskaya, un tel schéma est dit $\mathbf{objectif-moyen}$ sûr, avec $\mathbf{objectif} \in \{\mathbf{UBK}, \mathbf{UF}, \mathbf{SU}, \mathbf{EU}\}$ et $\mathbf{moyen} \in \{\mathbf{NMA}, \mathbf{KMA}, \mathbf{CMA}\}$, s'il s'agit d'un schéma de signature $\mathbf{objectif-moyen}$ sûr si l'on considère l'engagement comme un message classique et si les preuves du schéma se comportent comme des preuves de connaissance "sans divulgation". Ce second critère implique que l'on ne peut pas apprendre d'information sur les valeurs engagées ou sur la signature, à part le fait qu'elles sont connues du prouveur, que l'engagement est valide et qu'il existe une signature du signataire sur celui-ci. Nous verrons plus en détail les critères de sécurité liés aux preuves de connaissance au Chapitre 3.

Pour résumer, prouver qu'une signature Camenisch-Lysyanskaya est sûre revient à prouver d'une part les critères de sécurité liés à son statut de signature (voir Définition 29) et d'autre part que les preuves de connaissance sans divulgation ne donnent pas d'information autre que la connaissance ou non du secret (voir les preuves $\mathcal{ZKPK}$ au Chapitre 3 pour la formalisation correspondante).

Exemple. Il existe diverses constructions de tels schémas, décrites notamment dans [CL02, CL04, ASM08]. Je vais à présent décrire un exemple d'une telle solution. La solution choisie est la base de la solution que nous utiliserons plusieurs fois dans ce mémoire. Il s'agit d'un schéma sûr sous l'hypothèse q-SDH qui est issue des signatures Boneh-Boyen faibles et dont le principe a été posé dans [CL04]. Cette solution a été choisie comme premier exemple car il s'agit d'une première version, plus simple, de la construction que nous utiliserons dans la suite de ce mémoire. La sécurité de ces deux constructions repose sur l'hypothèse q-SDH, or j'utiliserai ces solutions dans des contextes qui requièrent déjà cette hypothèse. L'utilisation de ce type de construction me permet ainsi de faire reposer la sécurité du système global sur une seule hypothèse.

Construction 7 (Schéma de signature Camenisch-Lysyanskaya q-SDH).
Ce schéma utilise les engagements de Pedersen (voir Chapitre 1) et est composé des algorithmes suivants.

$\mathcal{CL}$.Init(1^λ). Cet algorithme génère un environnement bilinéaire asymétrique $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e})$ avec p premier. Pour rappel, un tel environnement est composé de deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ d'ordre p, d'un générateur pour chacun de ces groupes g_1 et g_2 ainsi que d'une forme bilinéaire calculable non dégénérée $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Les paramètres du système sont $\mathcal{CL}.\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e})$.

$$\mathcal{CL}.\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e}) \leftarrow \mathcal{CL}.\textsc{Init}$$

$\mathcal{CL}$.GénClé($\mathcal{CL}.\mathsf{param}$). Cet algorithme choisit aléatoirement $g, h, h_0, h_1, \cdots, h_t \in \mathbb{G}_1$. La clé secrète du signataire est $\mathsf{sk} = \gamma \in \mathbb{Z}_p$ et correspond à la valeur publique $w = g_2^\gamma$. Nous considérons dans la suite, que les paramètres du système sont inclus dans la clé publique du signataire $\mathsf{pk} = (w, g, h, h_0, h_1, \cdots, h_t, \mathcal{CL}.\mathsf{param})$.

$$(\mathsf{sk} = \gamma, \mathsf{pk} = (w, g, h, h_0, h_1, \cdots, h_t, \mathcal{CL}.\mathsf{param})) \leftarrow \mathcal{CL}.\textsc{GénClé}(\mathcal{CL}.\mathsf{param})$$

$\mathcal{CL}$.Signe($\mathsf{m}, \mathsf{sk}, \mathsf{clpk}$). Pour signer le message $\mathsf{m} = \mathsf{m}_0 \| \cdots \| \mathsf{m}_k$, le signataire choisit aléatoirement $x \in \mathbb{Z}_p$ et calcule

$$A = (g_1 h_0^{\mathsf{m}_0} h_1^{\mathsf{m}_1} \cdot \ldots \cdot h_k^{\mathsf{m}_k})^{\frac{1}{\gamma + x}}.$$

Il obtient ainsi la signature $\sigma = (A, x)$.
On remarque que $h_0^{\mathsf{m}_0} h_1^{\mathsf{m}_1}$ correspond à un engagement de Pedersen avec m_0 prenant la place de l'aléa.

$$\sigma = (A, x) \leftarrow \mathcal{CL}.\textsc{Signe}(\mathsf{m}, \mathsf{sk}, \mathsf{pk})$$

$\mathcal{CL}$.CSigne($C, \pi_C, \mathsf{sk}, \mathsf{pk}$). Cet algorithme prend en entrées un engagement de Pedersen C et la paire de clés du signataire $(\mathsf{sk} = \gamma, \mathsf{pk})$. Il choisit aléatoirement $x \in \mathbb{Z}_p$ et génère $A = (g_1 C)^{\frac{1}{\gamma + x}}$. La signature σ est le couple (A, x).

$$\sigma = (A, x) \leftarrow \mathcal{CL}.\textsc{CSigne}(C, \mathsf{sk}, \mathsf{pk})$$

Remarque. On remarquera qu'une signature $\sigma = (A, x)$ d'un engagement C sur un ensemble de valeurs $\mathsf{m}_0, \cdots, \mathsf{m}_k$ correspond à la signature classique du message de $k+1$ blocs $\tilde{\mathsf{m}} = \mathsf{m}_0 \| \cdots \| \mathsf{m}_k$.

$\mathcal{CL}$.Prouve. Ce protocole se déroule en deux phases. Dans un premier temps, l'utilisateur génère un engagement sur la signature σ. Pour cela, il choisit aléatoirement r et u dans $\mathbb{Z}_p$ et calcule $C_1 = Ah^r$ et $C_2 = g^r h^u$. L'utilisateur utilise ensuite un protocole de preuve de connaissance sans divulgation (voir Chapitre 3) pour montrer d'une part qu'il connait la signature qui correspond aux engagements C_1,C_2 et d'autre part qu'il s'agit d'une signature de l'engagement C dont il connait les valeurs engagées $\mathsf{m}_0, \cdots, \mathsf{m}_k$ avec m_0 éventuellement un aléa.

$$\begin{aligned}\pi_\sigma = \quad & \textsc{Pok}(\mathsf{m}_0, \cdots, \mathsf{m}_k, x, rx, r, u, ux : C_2 = g^r h^u \wedge 1 = C_2^x g^{-rx} h^{-ux} \wedge \\ & e(g_1, g_2)/e(C_1, w) = e(C_1, g_2)^x e(h, g_2)^{-rx} e(h, w)^{-r} \textstyle\prod_{j=0}^{k} e(h_j, g_2)^{-\mathsf{m}_j})\end{aligned}$$

$\mathcal{CL}$.Vérifie($\sigma, \mathsf{m}, \mathsf{pk}$). Cet algorithme publique permet de vérifier la validité d'une signature $\sigma = (A, x)$ sur un message $\mathsf{m} = \mathsf{m}_0 \| \cdots \| \mathsf{m}_k$ avec la clé publique pk. Pour cela il vérifie la relation suivante.

$$e(A, g_2)^x e(A, w) e(h_0, g_2)^{-\mathsf{m}_0} \cdot \ldots \cdot e(h_{k-1}, g_2)^{-\mathsf{m}_k} = e(g_1, g_2)$$

Si la relation est correcte alors il retourne vrai, sinon il retourne faux.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{CL}.\text{VÉRIFIE}(\sigma, \mathsf{m}, \mathsf{pk})$$ ⟲

Cette solution n'est pas résistante aux attaques adaptatives. [ACJT00, CL02] préconisent de modifier l'algorithme $\mathcal{CL}$.CSIGNE afin de calculer conjointement l'aléa de l'engagement de Pedersen. Une solution pour obtenir la résistance aux attaques adaptatives et de remplacer l'algorithme de signature $\mathcal{CL}$.CSIGNE par un protocole de signature interactif. Le principe des protocoles de signature interactif est utilisé dans de nombreux articles, notamment [CHL05, DP06, ACJT00, BBS04]. Il est assez simple de l'appliquer au schéma que nous venons de voir. Dans la suite de ce mémoire, nous utiliserons la version interactive de ce schéma. Je décris donc brièvement l'adaptation nécessaire.

Signature Camenisch-Lysyanskaya en pratique. Le schéma reste identique à la construction que nous venons de voir, excepté que l'algorithme $\mathcal{CL}$.CSIGNE devient un protocole interactif entre un utilisateur u et le signataire. Ce protocole permet à l'utilisateur d'engager un ensemble de valeurs $\mathsf{m}_1, \cdots, \mathsf{m}_t$, puis de faire signer l'engagement obtenu par le signataire de telle sorte que celui-ci n'apprenne rien sur les valeurs engagées. Il est même possible que l'utilisateur choisisse seulement une partie des valeurs, par exemple $\mathsf{m}_1, \cdots, \mathsf{m}_k$ et que les valeurs restantes, $\mathsf{m}_{k+1}, \cdots, \mathsf{m}_t$, soient choisies par le signataire. L'utilisateur obtenant, en fin de protocole, une signature valide des valeurs $\mathsf{m}_1, \cdots, \mathsf{m}_t$ sans que le signataire aient obtenu d'information sur les valeurs $\mathsf{m}_1, \cdots, \mathsf{m}_k$ qu'il a gardé secrète. Cette variante est légèrement plus complexe que la solution de base mais elle nous sera utile par la suite, c'est donc celle-ci que je vais maintenant décrire.

Dans le cas du schéma $\mathcal{CL}$ que nous venons de voir, le protocole interactif de signature se déroule de la façon suivante.

Construction 8.

Utilisateur. L'utilisateur choisit aléatoirement s' dans $\mathbb{Z}_p$, puis s'engage sur les valeurs $s', \mathsf{m}_1, \ldots, \mathsf{m}_k$ avec un engagement de Pedersen.

$$C' = h_0^{s'} h_1^{\mathsf{m}_1} \cdots h_k^{\mathsf{m}_k} \text{ avec } h_0, h_1, \cdots, h_k \in \mathbb{G}_1$$

Il prouve ensuite que C' est correctement formé de manière similaire à la construction classique. On note $\pi_{C'}$ cette preuve de connaissance $\mathcal{ZKPK}$.

$$\pi_{C'} = \text{POK}(s', \mathsf{m}_1, \ldots, \mathsf{m}_k : C' = h^{s'} h_1^{\mathsf{m}_1} \cdots h_k^{\mathsf{m}_k})$$

Remarque. Dans certaines constructions l'utilisateur peut avoir à justifier de la validité de certaines valeurs, par exemple, dans [CHL05] l'utilisateur doit attester qu'une des valeurs m_i est sa clé secrète d'utilisateur. Afin d'éviter la redondance, l'utilisateur peut justifier de la validité de ces contraintes directement dans la preuve $\pi_{C'}$.

Signataire. Le signataire vérifie la validité de la preuve $\pi_{C'}$. Si celle-ci est fausse, il retourne $\perp$. Sinon il choisit aléatoirement $s" \in \mathbb{Z}_p$ puis génère l'engagement suivant.

$$C = C' h_{k+1}^{m_{k+1}} \cdots h_t^{m_t} h_0^{s''}$$

Enfin, le signataire choisit aléatoirement $x \in \mathbb{Z}_p$ et génère $A = (g_1 C)^{\frac{1}{\gamma+x}}$. La signature σ est le couple (A, x).

$$\sigma = (A, x) \leftarrow \mathcal{CL}.\text{CSIGNE}(C, \mathsf{sk}, \mathsf{pk})$$

L'aléa s'' et la signature σ sont ensuite envoyés à l'utilisateur.

Utilisateur. L'utilisateur obtient $C = h_0^{s'+s''} h_1^{\mathsf{m}_1} \cdot \cdots \cdot h_t^{m_t}$ et la signature Camenisch-Lysyanskaya $\sigma = (A, x)$ du message $\mathsf{m} = (\mathsf{m}_0, \mathsf{m}_1, \cdots, \mathsf{m}_\ell)$ avec $\mathsf{m}_0 = s' + s''$. ↺

2.3.2 Signatures de groupe

Les signatures de groupes, introduites par D.Chaum et E.van Heyst dans [Cv91], permettent à un signataire de signer des messages au nom d'un groupe auquel il est enregistré. Ces signatures sont souvent utilisées afin de permettre une forme d'anonymat aux membres du groupe : le signataire d'un message est caché au sein du groupe tant qu'il n'y a pas de litige. Une telle signature a les trois propriétés supplémentaires suivantes.

1. Seul un membre du groupe peut signer au nom du groupe.
2. La vérification d'une signature de groupe permet de s'assurer qu'un membre du groupe à générer la signature mais ne permet pas de savoir lequel.
3. En cas de litige, une autorité d'ouverture est capable "d'ouvrir" la signature et de retrouver l'identité du signataire correspondant, de telle sorte qu'il puisse prouver que c'est bien le signataire.

Un schéma de signature de groupe comporte trois acteurs.

– Le gestionnaire de groupe, noté GG, a pour rôle d'enregistrer les nouveaux membres du groupe.
– L'ensemble U des utilisateurs u_i qui sont membres du groupe car enregistrés auprès du gestionnaire de groupe. Tout utilisateur de U a le pouvoir de signer, anonymement, au nom du groupe.
– L'autorité d'ouverture, noté AO, a pour rôle, en cas de litige, de lever l'anonymat sur les signatures.

Il existe principalement deux modèles pour ces signatures. Le modèle dit BMW du nom de ses auteurs M.Bellare, D.Micciancio et B.Warinschi, proposé dans [BMW03], a pour objectif de modéliser les schémas de signature de groupe statiques, c'est-à-dire pour lesquels les membres du groupe sont fixés une fois pour toute au début de l'utilisation du schéma et avant toute signature. Le modèle BSZ de M.Bellare, H.Shi et C.Zhang [BSZ05] a été conçu pour répondre à la problématique des signatures de groupes dynamiques, c'est-à-dire le cas où des membres peuvent rejoindre le groupe à tout moment. Toutes les signatures de groupes que nous utiliserons dans ce mémoire devrons être sûres dans ce second modèle, je décrirai donc ici les signatures de groupes et les notions de sécurité qui leurs sont associées dans le modèle BSZ uniquement.

Définition 32 (Schéma de signature de Groupe).
Un schéma de signature de groupe $\mathcal{SG}$ comporte sept algorithmes.

– $\mathcal{SG}.\text{INIT}$ prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système notés $\mathcal{SG}.\mathsf{param}$ ainsi que la clé publique du groupe gpk, la clé secrète du gestionnaire de groupe ggsk et la clé secrète de l'autorité d'ouverture osk. Nous considérerons dans la suite que λ et $\mathcal{SG}.\mathsf{param}$ sont inclus dans la clé publique du groupe.

$$(\mathsf{gpk}, \mathsf{ggsk}, \mathsf{osk}) \leftarrow \mathcal{SG}.\text{INIT}(1^\lambda)$$

– $\mathcal{SG}$.Rejoins est un protocole interactif d'enregistrement entre un utilisateur u_i, éventuellement muni d'une paire de clés long-terme "personnelle" notée $(\mathsf{psk}, \mathsf{ppk})$, et le gestionnaire de groupe GG en possession de sa clé secrète ggsk. À la fin du protocole, le nouveau membre obtient un certificat d'appartenance au groupe cer_i et une clé secrète d'utilisateur du groupe usk_i. Pour sa part, le gestionnaire de groupe ajoute une entrée $\mathsf{reg}[i]$ dans la table d'enregistrement reg comprenant l'ensemble des échanges effectués durant cette procédure. Notons que l'autorité d'ouverture a un accès en lecture à la table d'enregistrement reg.

$$(\mathsf{reg}, \mathsf{u}_i : \mathsf{usk}, \mathsf{cer}_i) \leftarrow \mathcal{SG}.\textsc{Rejoins}(\mathsf{u}_i : \mathsf{gpk}, (\mathsf{psk}, \mathsf{ppk}); \mathsf{GG} : \mathsf{gpk}, \mathsf{ggsk})$$

– $\mathcal{SG}$.Signe est l'algorithme de signature. Il prend en entrées un message m, la clé secrète et le certificat de l'utilisateur $(\mathsf{usk}_i, \mathsf{cer}_i)$ ainsi que la clé publique du groupe gpk. L'algorithme retourne une signature σ du message m (ou $\perp$ en cas d'erreur).

$$\sigma \leftarrow \mathcal{SG}.\textsc{Signe}(\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{gpk})$$

– $\mathcal{SG}$.Vérifie est un algorithme public permettant de vérifier une signature σ sur un message m grâce à la clé publique du groupe gpk. Il retourne vrai si la signature est valide et faux si elle ne l'est pas.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SG}.\textsc{Vérifie}(\mathsf{m}, \sigma, \mathsf{gpk})$$

– $\mathcal{SG}$.Ouvre permet à l'autorité d'ouverture de retrouver l'identité de l'émetteur d'une signature. Pour cela, il prend la paire (m, σ) dont il veut connaitre l'auteur, sa clé secrète d'autorité d'ouverture osk et la clé publique du groupe gpk. En bénéficiant d'un accès en lecture à la table d'enregistrement reg, il retourne l'identité du signataire u_i ainsi qu'une preuve π_u que c'est bien cet utilisateur qui est l'auteur de la signature σ.

$$(\mathsf{u}_i, \pi_\mathsf{u}) \leftarrow \mathcal{SG}.\textsc{Ouvre}(\mathsf{m}, \sigma, \mathsf{osk}, \mathsf{gpk})$$

– $\mathcal{SG}$.Juge est un algorithme public permettant de vérifier, à l'aide d'une preuve π_u générée par l'autorité d'ouverture, si l'utilisateur u_i suspecté est bien l'auteur de la paire message-signature (m, σ). Pour cela, il prend en entrées la paire concernée (m, σ), la preuve π_u provenant (a priori) de l'algorithme $\mathcal{SG}$.Ouvre ainsi que l'identité u_i de l'utilisateur suspecté. L'algorithme retourne vrai si u_i est bien le signataire impliqué dans le couple (m, σ) et faux sinon.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SG}.\textsc{Juge}(\mathsf{m}, \sigma, \mathsf{u}_i, \pi_\mathsf{u}) \qquad *$$

Un schéma de signature de groupe est dit sûr dans le modèle BSZ s'il assure les quatre propriétés de sécurité suivantes.

– **Consistance**. Toute signature émise de manière honnête par un membre honnête doit être valide (l'algorithme $\mathcal{SG}$.Vérifie doit accepter la signature produite honnêtement par l'algorithme de signature $\mathcal{SG}$.Signe) et l'algorithme d'ouverture doit pouvoir correctement en identifier le signataire (l'algorithme $\mathcal{SG}$.Juge doit accepter la preuve produite par l'algorithme d'ouverture $\mathcal{SG}$.Ouvre).
Dans l'expérience correspondante, on considère un adversaire $\mathcal{A}$ ayant accès à la clé publique du groupe gpk et à deux oracles, le premier lui permettant d'ajouter au groupe des utilisateurs honnêtes de son choix et le second lui permettant de lire la table d'enregistrement reg.
À la fin de l'expérience, l'adversaire retourne une paire $(\mathsf{u}_i, \mathsf{m})$. Il gagne l'expérience si l'utilisateur u_i est un membre honnête du groupe et qu'un des trois points suivant est vérifié.

- La signature de groupe σ de u_i pour le message m obtenue par l'algorithme $\mathcal{SG}$.SIGNE n'est pas considérée comme valide par l'algorithme de vérification $\mathcal{SG}$.VÉRIFIE.
- L'ouverture effectuée par l'algorithme $\mathcal{SG}$.OUVRE sur le couple (σ, m) retourne un utilisateur autre que u_i.
- L'algorithme $\mathcal{SG}$.JUGE n'est pas capable de vérifier la preuve retournée par l'algorithme d'ouverture.

Le succès de l'adversaire $\mathcal{A}$ dans cette expérience est noté $\mathsf{Succ}^{\mathcal{S}}_{\mathbf{const},\mathcal{A}}(\lambda)$.
Un schéma de signature de groupe $\mathcal{SG}$ est dit **consistant** si le succès $\mathsf{Succ}^{\mathcal{SG}}_{\mathbf{const},\mathcal{A}}(\lambda)$ de tout adversaire polynomial $\mathcal{A}$ est nul.

$$\mathsf{Succ}^{\mathcal{SG}}_{\mathbf{const},\mathcal{A}}(\lambda) = 0.$$

– **Anonymat**. Il ne doit pas être possible de distinguer le signataire d'un message, même en choisissant le message et en limitant les signataires possibles dans un ensemble de deux membres honnêtes que l'on choisi.
Dans l'expérience correspondante, un bit b aléatoirement et secrètement choisi est pris en entrée de l'expérience. On considère ensuite un adversaire $\mathcal{A}$ ayant accès à la clé publique du groupe gpk, à la clé secrète du gestionnaire de groupe ggsk et ayant accès à un ensemble d'oracles lui permettant notamment de corrompre des utilisateurs en modifiant leur clé publique ou en récupérant leur clé secrète, d'écrire dans la base de registre, de demander l'ouverture des signatures et d'interagir en tant que gestionnaire de groupe avec des utilisateurs honnêtes. Enfin, l'adversaire à accès à un oracle de challenge auquel il fournit un message m de son choix ainsi que l'identité de deux utilisateurs honnêtes du groupe, que nous notons u_0 et u_1. Cet oracle retourne une signature de groupe de l'utilisateur u_0 si $b = 0$ ou une signature de groupe de l'utilisateur u_1 si $b = 1$.
À la fin de l'expérience, l'adversaire retourne un bit b^* et gagne l'expérience si $b = b^*$.
L'avantage de l'adversaire polynomial $\mathcal{A}$ dans cette expérience est

$$\mathsf{Adv}^{\mathcal{SG}}_{\mathbf{anon},\mathcal{A}}(\lambda) = |Pr[1 \longleftarrow \mathrm{Exp}^{\mathcal{HC}}_{\mathbf{uni},\mathcal{A}}(\lambda)] - 1/2|.$$

Un schéma de signature de groupe $\mathcal{SG}$ est dit **anonyme** si l'avantage $\mathsf{Adv}^{\mathcal{SG}}_{\mathbf{anon},\mathcal{A}}(\lambda)$ de tout adversaire polynomial $\mathcal{A}$ est négligeable.

$$\mathsf{Adv}^{\mathcal{SG}}_{\mathbf{anon},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.}$$

– **Traçabilité**. Il doit être impossible de produire une signature valide pour laquelle l'autorité d'ouverture n'est pas capable de retrouver l'identité du signataire ou pour laquelle il est convaincu de l'identité du signataire mais n'est pas capable de le prouver.
Dans l'expérience correspondante, on considère un adversaire $\mathcal{A}$ ayant accès à la clé publique du groupe gpk, à la clé secrète de l'autorité d'ouverture osk ainsi qu'à un ensemble d'oracles lui permettant de corrompre des utilisateurs en modifiant leur clé publique ou en récupérant leur clé secrète, d'ajouter au groupe des utilisateurs honnêtes de son choix, de lire dans la base de registre et d'effectuer le protocole $\mathcal{SG}$.REJOINS entre un utilisateur corrompu et le gestionnaire de groupe.
À la fin de l'expérience, l'adversaire retourne une paire (m, σ). Il gagne alors l'expérience si la signature σ est une signature valide du message m tel que l'autorité d'ouverture retourne qu'elle n'a été effectuée par personne ($\mathsf{u}_i = 0$) ou telle que le juge refuse la preuve d'ouverture. Le succès de l'adversaire polynomial $\mathcal{A}$ dans cette expérience est noté $\mathsf{Succ}^{\mathcal{S}}_{\mathbf{trace},\mathcal{A}}(\lambda)$.

Un schéma de signature de groupe $\mathcal{SG}$ est dit **traçable** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SG}}_{\mathbf{trace},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SG}}_{\mathbf{trace},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.}$$

– **Non-diffamation**. Personne (pas même les autorités) ne doit être capable de faire faussement accuser quelqu'un d'avoir signer un message. Nous considérons ici que les deux autorités (le gestionnaire de groupe et l'autorité d'ouverture) sont corrompues. De plus un sous-ensemble des utilisateurs peut lui aussi être corrompu.
Dans l'expérience correspondante, on considère un adversaire $\mathcal{A}$ ayant accès à la clé publique du groupe gpk, à la clé secrète de l'autorité d'ouverture osk et du gestionnaire de groupe ggsk ainsi qu'à un ensemble d'oracles lui permettant de corrompre des utilisateurs en modifiant leur clé publique ou en récupérant leur clé secrète, d'obtenir des signatures de groupe d'utilisateurs honnêtes qu'il choisit pour les messages de son choix, d'écrire dans la base de registre et d'interagir en tant que gestionnaire de groupe avec des utilisateurs honnêtes.
À la fin de l'expérience, l'adversaire retourne un quadruplet $(\mathsf{m}, \sigma, \mathsf{u}_i, \pi_{\mathsf{u}})$. Il gagne l'expérience si la signature σ est une signature valide du signataire u_i du message m alors qu'elle n'a jamais été demandée à l'oracle de signature et que l'utilisateur u_i est un utilisateur honnête dont la clé secrète n'a jamais été demandée par l'adversaire à un oracle.
Le succès de l'adversaire polynomial $\mathcal{A}$ dans cette expérience est noté $\mathsf{Succ}^{\mathcal{S}}_{\mathbf{nd},\mathcal{A}}(\lambda)$.
Un schéma de signature de groupe $\mathcal{SG}$ satisfait la propriété de **Non-diffamation** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SG}}_{\mathbf{nd},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SG}}_{\mathbf{nd},\mathcal{A}}(\lambda) < \epsilon, \text{ avec } \epsilon \text{ négligeable.}$$

Exemple Différents schémas de signatures de groupe sont construits à partir du schéma de signature [BB04] et d'un schéma de chiffrement, notamment [DP06, FI05, BBS04]. Ces deux schémas sont basés sur la même construction, [DP06] étant une amélioration dynamique de [BBS04] qui ne considère que le cas statique. Nous prendrons ici, comme exemple, la solution XSGS de C.Delerablée et D.Pointcheval [DP06]. Dans ce cas, le schéma de chiffrement peut être soit le (double) chiffrement Elgamal [ElG85, DP06] soit le (double) chiffrement linéaire [BBS04]. Afin de ne pas complexifier inutilement la lecture de ce mémoire, je décrirai le fonctionnement de ce schéma uniquement dans le cas du double chiffrement Elgamal qui nécessite, en plus de l'hypothèse q-SDH (voir Définition 18), l'hypothèse XDH (voir Définition 15). Par contre, il fournit alors des signatures plus courtes. Je décris ici la construction décrite dans [DP06], le protocole permettant à un utilisateur de se joindre au groupe est donc la version prouvée sûre dans un cas concurrent, c'est-à-dire dans le cas où plusieurs exécutions peuvent être effectuées en parallèle.

Construction 9 (Signature de groupe [DP06]).
Le schéma de signature de groupe $\mathcal{SG}$ proposé dans [DP06] utilise un schéma de signature, noté $\mathcal{S}$, $\mathbf{EU-CMA}$ sûr ainsi qu'un schéma d'engagement *extractible* parfaitement **résistant aux collisions** et calculatoirement **Indistinguable** que nous notons $\mathcal{EE}$ (voir Définition 26). Il est à noter que l'ouverture des engagements n'est utilisée que pour prouver la sécurité du schéma, la clé d'ouverture de l'engagement n'est, en pratique, connue de personne nous l'ignorons donc dans notre description.

$\mathcal{SG}.\textsc{Init}(1^\lambda)$ Cet algorithme consiste en la génération d'un environnement bilinéaire asymétrique $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e})$ avec p premier. Pour rappel, un tel environnement est composé des deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ d'ordre p, d'un générateur pour chacun de ces groupes

g_1 et g_2 ainsi que d'une forme bilinéaire calculable non dégénérée $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. D'autre part, il initialise le schéma d'engagement avec l'algorithme $\mathcal{EE}.\textsc{Init}$ et obtient les paramètres correspondants, notés $\mathcal{EE}.\mathsf{param}$.
Il génère ensuite les clés du système. Il choisit aléatoirement γ, ζ_1 et ζ_2 dans $\mathbb{Z}_p^*$ et k dans $\mathbb{G}_1$ puis il calcule $w = g_2^\gamma$, $h = k^{\zeta_1}$ et $g = k^{\zeta_2}$. Il obtient ainsi les clés suivantes pour le schéma.
La clé secrète du gestionnaire de groupe est $\mathsf{ggsk} = \gamma$, la clé secrète du gestionnaire d'ouverture $\mathsf{osk} = (\zeta_1, \zeta_2)$ et enfin, la clé publique du groupe $\mathsf{gpk} = (w, k, h, g, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e}, \mathcal{EE}.\mathsf{param})$. On peut remarquer que la clé secrète du gestionnaire de groupe ggsk correspond à une clé de signature tandis que la clé secrète du gestionnaire d'ouverture est une clé de déchiffrement pour le schéma de double chiffrement Elgamal (voir Définition 5).

$$(\mathsf{gpk}, \mathsf{ggsk}, \mathsf{osk}) \leftarrow \mathcal{SG}.\textsc{Init}(1^\lambda)$$

$\mathcal{SG}.\textsc{Rejoins}(\mathsf{u}_i : \mathsf{gpk}, (\mathsf{psk}, \mathsf{ppk}); \mathsf{GG} : \mathsf{gpk}, \mathsf{ggsk})$ Ce protocole est un protocole de signature interactif entre le gestionnaire de groupe, jouant le rôle du signataire, et le nouveau membre du groupe, c'est-à-dire l'utilisateur. On suppose que l'utilisateur possède une paire de clés personnelle $(\mathsf{psk}, \mathsf{ppk})$ pouvant être utilisée pour le schéma de signature "classique" $\mathcal{S}$.

Utilisateur u_i. L'utilisateur choisit aléatoirement y' dans $\mathbb{Z}_p$, calcule $C = h^{y'}$ et s'engage sur la valeur y' avec le schéma d'engagement $\mathcal{EE}$ en calculant $(c, r) = \mathcal{EE}.\textsc{Engage}(y')$. Il prouve ensuite que c est correctement formé, c'est à dire qu'il s'agit bien d'un engagement sur la valeur y' contenue dans C. En utilisant une notation plus formelle, il construit la preuve de connaissance sans divulgation suivante.

$$\pi_c = \textsc{Pok}(y', r : (c, r) = \mathcal{EE}.\textsc{Engage}(y') \wedge C = h^{y'})$$

Il envoie ensuite au signataire les valeurs C et c ainsi que la preuve π_c.

Gestionnaire de Groupe GG. Le gestionnaire de groupe vérifie la validité de la preuve π_c et que $C \in \mathbb{G}_1$. Si la preuve est fausse ou si C n'est pas un élément de $\mathbb{G}_1$, il retourne $\perp$. Sinon il choisit aléatoirement $x, y'' \in \mathbb{Z}_p$ puis calcule les variables suivantes.

$$A = (g_1 C h^{y''})^{\frac{1}{\mathsf{ggsk}+x}}$$

A et x sont les deux premières parties du certificat d'appartenance au groupe que l'utilisateur cherche à obtenir dans ce protocole et y'' sera essentielle pour la formation de la dernière partie de ce certificat. Il faut donc assurer à l'utilisateur que ces différents éléments ont été construits correctement. On note $B = e(g_1 C h^{y''}, g_2)/e(A, w)$ et $D = e(A, g_2)$, deux valeurs que l'on peut générer à partir de variables connues de l'utilisateur, de A et y'', sans utiliser x. Nous avons $g_1 C h^{y''} = A^{\mathsf{ggsk}+x}$ et $w = g_2^{\mathsf{ggsk}}$ donc $B = e(A^{\mathsf{ggsk}+x}, g_2) e(A, g_2^{\mathsf{ggsk}})^{-1}$. Il suffit alors d'utiliser les propriétés de bilinéarité de e pour voir que $B = e(A^x, g_2)$, ce qui implique $B = D^x$. Ainsi, prouver que A a été correctement généré avec un y'' donné et un x secret, revient à montrer que l'on connait le logarithme discret de B en base D. On note π_{BD} cette preuve.

$$\pi_{BD} = \textsc{Pok}(x : B = D^x)$$

Le gestionnaire de groupe envoie A, y'' et π_{BD} à l'utilisateur.

Utilisateur u_i. L'utilisateur calcule d'abord les valeurs B et D à partir des éléments en sa possession.

$$B = e(g_1 C h^{y''}, g_2)/e(A, w) \qquad D = e(A, g_2)$$

Il vérifie ensuite la preuve π_{BD} et que $A \in \mathbb{G}_1$. Si la preuve est fausse ou $A \notin \mathbb{G}_1$, il retourne $\perp$. Sinon, il pose $y = y' + y''$ et signe A avec sa paire de clé long-terme $\sigma_{\mathsf{u}_i} = \mathcal{S}.\textsc{Signe}(\mathsf{ppk}, \mathsf{psk}, A)$. Il envoie ensuite cette signature au gestionnaire de groupe.

Gestionnaire de Groupe GG. Le gestionnaire de groupe vérifie la signature σ_{u_i} de l'utilisateur pour le message A avec l'algorithme de vérification fourni par le schéma de signature $\mathcal{S}$. Si $\mathcal{S}.\textsc{Vérifie}(A, \sigma_{\mathsf{u}_i}, \mathsf{ppk})$ retourne vrai, il ajoute $\mathsf{reg}[i] = (\mathsf{ppk}, A, x, \sigma_{\mathsf{u}_i})$ à sa table de registre et envoie x à l'utilisateur. Sinon il retourne $\perp$.

Utilisateur u_i. L'utilisateur vérifie la validité du triplet (A, x, y) obtenu en testant la relation suivante.

$$e(A, g_2)^x e(A, x) e(h, g_2)^{-y} = e(g_1, g_2)$$

Si la relation est valide, l'utilisateur a obtenu le certificat d'appartenance au groupe $\mathsf{cert}_i = (A, x)$ relié à sa clé secrète d'utilisateur $\mathsf{usk}_i = y$, sinon le protocole a échoué et il retourne $\perp$.

$\mathcal{SG}.\textsc{Signe}(\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{gpk})$ Le membre du groupe, u_i, souhaite produire une signature d'un message m. Il doit prouver qu'il a bien un certificat valide d'appartenance au groupe cert_i correctement relié à sa clé secrète d'utilisateur usk_i et cela sans les révéler mais en permettant à l'autorité d'ouverture de révoquer son anonymat.

Dans un premier temps, il chiffre A en utilisant le double chiffrement Elgamal dont l'autorité d'ouverture possède la clé de déchiffrement. Il choisit donc aléatoirement $\alpha, \beta \in \mathbb{Z}_p^*$ puis calcule le chiffré de A qui correspond aux quatre valeurs suivantes.

$$T_1 = k^\alpha, \quad T_2 = Ah^\alpha, \quad T_3 = k^\beta, \quad T_4 = Ag^\beta$$

Il doit ensuite prouver deux prédicats. Premièrement, (T_1, T_2, T_3, T_4) est effectivement un double chiffré Elgamal correctement formé, c'est-à-dire que le signataire connait α et β tels que $T_1 = k^\alpha$, $T_3 = k^\beta$ et $T_2/T_4 = h^\alpha/g^\beta$. Deuxièmement, il doit prouver qu'il s'agit du chiffré d'un élément A appartenant à un certificat valide (A, x) qu'il connait et qui est relié à sa clé secrète usk_i. Pour cela, il suffit de remarquer que si la relation de vérification d'un certificat que nous avons vu plus haut est modifiée pour y remplaçant astucieusement A par son chiffré, on obtient la relation suivante.

$$e(T_2, g_2)^x e(h, g_2)^{-x\alpha} e(T_2, w) e(h, w)^{-\alpha} e(h, g_2)^{-\mathsf{usk}} = e(g_1, g_2)$$

Pour le second point, il suffit à l'utilisateur de prouver qu'il connait x, usk, α et $x\alpha$ tel que cette relation soit vérifiée.

D'autre part, l'utilisateur doit signer le message en s'identifiant comme celui-qui connait les secrets correspondant aux deux prédicats que nous venons de voir. Pour combiner ces trois objectifs (les deux prédicats et la signature), nous utilisons la signature de connaissance sans divulgation suivante.

$$\begin{aligned} \pi_\sigma \quad = \quad & \textsc{Sok}\big(\alpha, \beta, x, x\alpha, \mathsf{usk} : T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge \\ & e(T_2, g_2)^x e(h, w)^{-\alpha} e(h, g_2)^{-x\alpha} e(h, g_2)^{-\mathsf{usk}} = e(g_1, g_2)/e(T_2, w)\big)(\mathsf{m}) \end{aligned}$$

L'utilisateur obtient ainsi la signature de groupe $\sigma = (T_1, T_2, T_3, T_4, \pi_\sigma)$ du message m.

$\mathcal{SG}.\textsc{Vérifie}(\mathsf{m}, \sigma, \mathsf{gpk})$ Cet algorithme vérifie la signature de connaissance π_σ du message m en utilisant les éléments T_1, T_2, T_3, T_4 et la clé publique du groupe gpk. Si celle-ci est valide, la signature est correcte. Sinon la signature est fausse.

$\mathcal{SG}$.Ouvre($\mathsf{m}, \sigma, \mathsf{osk}, \mathsf{gpk}$) Pour rappel, la clé secrète osk du gestionnaire d'ouverture correspond à la clé de déchiffrement du double chiffrement Elgamal utilisé par le membre pour chiffrer A. L'autorité d'ouverture n'a donc qu'à déchiffrer T_1, T_2, T_3, T_4, pour retrouver $A : A = T_2/T_1^{\zeta_1}$. Il génère ensuite une preuve de connaissance sans divulgation, vérifiable par l'algorithme public $\mathcal{SG}$.Juge, que A est bien le déchiffré de T_1, T_2, T_3, T_4, c'est-à-dire assurer que l'on connait ζ_1 reliée à la clé publique $h = k^{\zeta_1}$ et que $T_2/A = T_1^{\zeta_1}$.

$$\pi_{ouv} \quad = \quad \textsc{Pok}\big(\zeta_1 : T_2/A = T_1^{\zeta_1} \wedge h = k^{\zeta_1}\big) \qquad \circlearrowleft$$

Dans le schéma XSGS [DP06], les auteurs posent $h_0 = h$ afin de simplifier les calculs de la signature de groupe. En particulier, il est ainsi possible de calculer $z = x\alpha + \mathsf{usk}$ et de simplifier la signature de connaissance comme suit.

$$\begin{aligned} \pi_\sigma \quad = \quad & \textsc{Sok}\big(\alpha, \beta, z : T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge \\ & e(T_2, g_2)^x e(h, w)^{-\alpha} e(h, g_2)^{-z} = e(g_1, g_2)/e(T_2, w)\big)(\mathsf{m}) \end{aligned}$$

La version non-optimisée a, elle, l'intérêt de permettre de prouver certains prédicats sur la valeur usk. Les deux versions ont donc chacune leurs points forts.

2.3.3 Signature de liste

Les signatures de listes (*list signatures* ou *Direct Anonymous Attestations (DAA)* en anglais), introduites en 2003 par S.Canard et J.Traoré [CT03], sont une variante des signatures de groupes qui permettent, dans certains cas, de relier deux signatures d'un même utilisateur. Celles-ci étant basées sur les signatures de groupe, je commence par une description de ces dernières.

Les signatures de listes sont une variante des signatures de groupes que nous venons de voir. Celles-ci permettent, dans certains cas, de relier deux signatures d'un même utilisateur. Par exemple, les solutions [CT03, CSST06] permettent de relier deux signatures du même utilisateur durant une même séquence (un même intervalle de temps). Un autre exemple est la solution [BCC04] dans laquelle deux signatures provenant d'un même membre du groupe peuvent être identifiées comme telles si elles ont le même destinataire. C'est cette dernière solution que nous utiliserons dans la suite de ce mémoire et c'est donc celle-ci que je décrirai ici.

La principale différence entre une signature de groupe et une signature de liste réside dans l'ajout, durant la procédure de signature, d'un témoin qui permettra de relier deux signatures. Par exemple, une solution est d'utiliser un témoin $T = h_D^{\mathsf{usk}}$ dans lequel h_D est un générateur du groupe $\mathbb{G}$ spécifique au destinataire et usk la clé secrète de l'émetteur. Pour un couple signataire-destinataire, cette valeur est identique d'une signature à une autre. Ainsi deux signatures du même membre du groupe pour le même destinataire pourront être reliées grâce à T. L'objectif étant que deux signatures du même signataire pour deux destinataires différents ne puissent pas être reliées entre elles, il est important que les générateurs h_{D_1} et h_{D_2}, correspondants à deux destinataires distincts soient différents l'un de l'autre et que trouver le logarithme discret de l'un en utilisant l'autre comme base soit difficile. Une possibilité pour empêcher que ces générateurs soient choisis les uns en fonction des autres, est que h_D soit calculé en utilisant une fonction de hachage résistante aux collisions sur des valeurs publiques spécifiques à chaque destinataire, par exemple un nom et une adresse.

Enfin, par défaut, les signatures de liste ne comportent pas de procédure de révocation d'anonymat des signataires. Mais si besoin, une telle procédure peut être ajoutée en utilisant un schéma de chiffrement de manière analogue aux signatures de groupe.

Exemple.
Une solution de signature de liste peut être déduite de la version longue de la signature de groupe XSGS décrite en section précédente. Pour cela il suffit d'y adjoindre la valeur $T = h_D^{\mathsf{usk}}$. La signature de connaissance π_σ est donc modifiée pour y ajouter l'assurance que l'utilisateur connait le logarithme discret de T en base h_D et que celui-ci possède bien la clé secrète d'utilisateur usk reliée au certificat de l'utilisateur signant. De manière plus formelle, la signature de connaissance devient :

$$\begin{aligned}\pi_\sigma = \quad & \textsc{Pok}\big(\alpha, \beta, x, x\alpha, \mathsf{usk} : T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge \\ & T = h_D^{\mathsf{usk}} \wedge \\ & e(T_2, g_2)^x e(h, w)^{-\alpha} e(h, g_2)^{-x\alpha} e(h_0, g_2)^{-\mathsf{usk}} = e(g_1, g_2)/e(T_2, w)\big)(\mathsf{m})\end{aligned}$$ •

Conclusion

J'ai décrit dans cette partie l'ensemble des notions et systèmes cryptographiques de base que nous utiliserons dans ce mémoire à l'exception des preuves de connaissances. Nous avons pu entrevoir au cours de ce chapitre l'importance que pouvait avoir la possibilité de prouver la connaissance d'un secret suivant un prédicat donné. C'est donc à ce type de schémas et à ma contribution pour ceux-ci que nous allons à présent nous intéresser.

Deuxième partie

Preuves de connaissance

Chapitre 3

Généralités sur les preuves de connaissances

Sommaire

En 1985, S.Goldwasser, S.Micali et C.Rackoff montrent dans [GMR85a, GMR89] qu'il est possible de prouver l'appartenance d'un mot à un langage, sans le divulguer. C'est à partir de ce prédicat que les preuves de connaissances ont été développées. Celles-ci permettent de prouver à un tiers que l'on connait un secret qui suit un prédicat, c'est-à-dire une réponse valide à une question qui peut éventuellement avoir plusieurs réponses valides, sans que le tiers puisse savoir la valeur de notre secret. À partir de cette idée, deux types de preuves sont apparues. Le premier type de preuve regroupe les preuves de connaissance sans divulgation. Ces preuves, introduites par U.Feige, A.Fiat et A.Shamir en 1988 [FFS88], permettent de prouver la connaissance d'un secret qui suit un prédicat sans donner aucune information sur le secret. Le second type de preuve considère que le prédicat peut avoir plusieurs réponses valides chaque réponse étant un témoin. Le secret connu du prouveur est alors un témoin "comme les autres", excepté qu'il est connu du prouveur. Le principe de ces preuves est alors de montrer que l'on connait un témoin qui suit le prédicat sans que le vérifieur soit capable de savoir quel témoin a été utilisé. Et cela même si le vérifieur connait tous les témoins qui correspondent au prédicat. Ces preuves, introduites en 1990 par U.Feige et A.Shamir [FS90], sont appelées preuves de connaissances à témoin indistinguable.

Je définirai d'abord ces deux types de preuves de connaissances de manière générique. Je présenterai ensuite quelques exemples de preuves permettant de prouver la connaissance de secret(s). Nous verrons enfin comment il est possible, notamment grâce à l'heuristique de Fiat-Shamir, d'obtenir des preuves de connaissance non-interactives, c'est-à-dire pour lesquelles il n'y a pas d'interaction entre le prouveur et le vérifieur.

3.1 Définitions

Nous allons à présent définir de manière plus formelle ce qu'est une preuve de connaissance, une preuve de connaissance sans divulgation et une preuve de connaissance à témoin indistinguable.

Définition 33 (Protocole interactif de preuve de connaissance $\mathcal{PK}$).
Un protocole interactif de preuve de connaissance, noté $\mathcal{PK}$, est un protocole $(\mathcal{P}, \mathcal{V})$ entre un prouveur $\mathcal{P}$ et un vérifieur $\mathcal{V}$ qui permet au vérifieur de s'assurer que le prouveur connait bien un secret x lié à un prédicat P. Il doit posséder les deux propriétés suivantes.

Complétude. (*Completeness*) Une preuve émise par un prouveur honnête, c'est à dire connaissant un secret x lié au prédicat P, sera acceptée avec une probabilité écrasante.

Validité. (*Soundness*) Une preuve émise par un prouveur malhonnête, c'est à dire ne connaissant aucun x répondant au prédicat P, sera acceptée avec une probabilité négligeable.

Nous notons une preuve de connaissance de valeurs secrètes vérifiant un prédicat de la manière suivante.

$$\textsc{Pok}\Big(\text{ valeurs secrètes } : \text{ prédicat }\Big)$$

∗

Définition 34 (Preuve de connaissance sans divulgation $\mathcal{ZKPK}$).
Une preuve de connaissance est dite sans divulgation si le vérifieur n'obtient aucune information sur le secret x autre que l'assurance que celui-ci est connu du prouveur. Un tel protocole est noté $\mathcal{ZKPK}$ pour *Zero Knowledge Proof of Knowledge*. On dit d'un tel protocole qu'il est sûr s'il possède les propriétés de **complétude**, de **validité** ainsi que la propriété suivante.

Sans divulgation. (*Zero Knowledge*) Il doit être possible, avec un algorithme en temps polynomial, de construire une transcription parfaitement (resp. calculatoirement) indistinguable d'une instance du protocole.

On dit alors que le protocole est parfaitement (resp. calculatoirement) **sans divulgation**. ∗

O.Goldreich et H.Krawczyk ont prouvé dans [GK96] qu'il est impossible de générer une preuve de connaissance sans divulgation qui se déroulerait en moins de quatre passes, une passe étant une transmission d'information de l'un des protagonistes vers l'autre. Une notion de divulgation de connaissance plus faible permet d'obtenir des protocoles composés de seulement trois passes. Pour cela, on suppose que le vérifieur se conduit de manière honnête, on parle alors de preuve de connaissance $\mathcal{ZKPK}$ sûre face à un vérifieur honnête ou simplement de preuve $\mathcal{ZKPK}$ à vérifieur honnête.

Définition 35 ($\mathcal{ZKPK}$ à vérifieur honnête).
Une preuve de connaissance sans divulgation est dite à vérifieur honnête si la propriété de **sans divulgation** est conditionnée par le fait que le vérifieur se conduise de manière honnête durant le protocole. Un tel protocole est noté $\mathcal{HV}-\mathcal{ZKPK}$. On dit qu'il est sûr s'il possède les propriétés de **complétude**, de **validité** ainsi que la propriété suivante.

Sans divulgation face à un vérifieur honnête. Il doit être possible, avec un algorithme en temps polynomial, de construire une transcription de façon parfaitement (resp. calculatoirement) indistinguable d'une instance du protocole impliquant un vérifieur honnête.

On dit alors que le protocole est parfaitement, statistiquement (resp. calculatoirement) **sans divulgation** face à un vérifieur honnête. ∗

Remarque. Un sous ensemble de ces protocoles assure, de plus, que si un prouveur répond correctement à deux questions distinctes sur le même engagement, alors il est possible de retrouver le secret engagé. Cette caractéristique étant une conséquence de la manière dont ils atteignent la propriété de **validité**, nous dirons de ces constructions qu'ils atteignent la propriété de **super-validité** [1].

Une seconde variante des preuves de connaissance $\mathcal{ZKPK}$ permettant des protocoles en trois passes est ce qu'on appelle les preuves de connaissance à témoin indistinguable. Pour ces schémas il existe toujours plusieurs témoins possibles pour chaque postulat et l'ensemble de ces témoins peuvent, éventuellement, être divulgués durant la preuve. La notion importante est qu'il doit être impossible de retrouver quel témoin parmi l'ensemble des témoins possibles à été utilisé dans la preuve. De manière plus formelle nous avons la définition suivante.

Définition 36 (Preuve de connaissance à témoin indistinguable $\mathcal{WIPK}$).
Une preuve de connaissance est dite à témoin indistinguable si le vérifieur est incapable de distinguer deux instances du protocole dans lesquelles le prouveur utilise des témoins différents correspondant au même prédicat P. Un tel protocole est noté $\mathcal{WIPK}$ pour *Witness Indistinguishable Proof of Knowledge*. On dit qu'il est sûr s'il possède les propriétés de **complétude**, de **validité** ainsi que la propriété suivante.

Témoin indistinguable. Pour tout vérifieur $\mathcal{V}$, et pour toutes paires de témoins (θ_1, θ_2), le protocole dans lequel le prouveur $\mathcal{P}$ utilise le témoin θ_1 comme secret est parfaitement, statistiquement ou calculatoirement indistinguable du protocole dans le quel $\mathcal{P}$ utilise le témoin θ_2 comme secret.

On dit alors que le protocole est parfaitement, statistiquement ou calculatoirement à **témoin indistinguable**. ∗

Dans la suite de ce mémoire nous utiliserons principalement des preuves $\mathcal{HV}-\mathcal{ZKPK}$ ou $\mathcal{WIPK}$ en trois passes. De manière générique, ces preuves se déroulent de la façon suivante.

Définition 37 (Protocole de preuve de connaissance en trois passes).
Un protocole de preuve de connaissance est dit à trois passes s'il comporte trois échanges entre le prouveur est le vérifieur.

Engagement. Le prouveur choisit un aléa afin de générer un engagement b qu'il envoie au vérifieur.

Question. Une fois l'engagement reçu, le vérifieur pose une question c au prouveur.

Réponse. Le prouveur calcule ensuite une réponse d à la question c en fonction de son engagement b et de son secret x, puis retourne cette réponse au vérifieur. ∗

3.2 Quelques exemples de preuve de connaissance

Nous allons à présent voir quelques exemples de preuve de connaissance $\mathcal{HV}-\mathcal{ZKPK}$ ou $\mathcal{WIPK}$ en trois passes sur les principaux prédicats que nous utiliserons dans la suite de ce mémoire. Les exemples que je donne ici sont basés sur le protocole d'identification proposé par C.P.Schnorr [Sch90], il est à noter que ces différentes preuves peuvent être générées à partir d'autres protocoles, notamment le protocole d'identification GPS introduit par M.Girault [Gir91] et prouvé par G.Poupard et J.Stern [PS98].

1. Cette notation est une adaptation de celle proposée par B. Chevallier-Mames dans [CM06].

3.2.1 Protocole d'identification de C.P.Schnorr

En 1990, C.P.Schnorr propose dans [Sch90] une solution de preuve de connaissance parfaitement sans divulgation (face à un vérifieur honnête) en trois passes sous l'hypothèse que le problème du logarithme discret est difficile. L'objectif original de cette solution est de permettre l'identification du prouveur, c'est-à-dire vérifier que celui-ci connait bien une clé secrète x correspondant à une clé publique connue des deux parties $y = g^x \bmod p$ avec p premier. Ce protocole permet ainsi de prouver la connaissance du logarithme discret x en base g d'une valeur publique y. Nous notons une telle preuve de la manière suivante.

$$\textsc{Pok}\Big(x : y = g^x\Big)$$

Le protocole d'identification de Schnorr fonctionne de la façon suivante.

Construction 10 (Protocole d'identification de Schnorr $\mathcal{HV} - \mathcal{ZKPK}$).

- Init. On choisit deux nombres premiers p et q tels que q divise $p-1$. On prend ensuite aléatoirement un élément g d'ordre q dans $\mathbb{Z}_p^*$.
- GénClé. Le prouveur génère la clé publique $y = g^x \bmod p$ correspondant à son secret $x \in \mathbb{Z}_q^*$ qui correspond à sa clé secrète.
- Prouve. Le protocole interactif de preuve d'identification se déroule de la façon suivante.

  Engagement. Le prouveur choisit a aléatoirement dans $\mathbb{Z}_q^*$ puis génère un engagement $b = g^a \bmod p$ qu'il envoie au vérifieur.

  Question. Une fois l'engagement reçu, le vérifieur choisit aléatoirement c dans $\mathbb{Z}_q$. c est la question qui est envoyée au prouveur.

  Réponse. Le prouveur calcule ensuite une réponse $d = a - c \cdot x \bmod q$ reliée à l'aléa a, à la question c et au secret x. Le vérifieur peut se convaincre que le prouveur connait bien le secret en testant si $b = g^d \cdot y^c \bmod p$. ↻

Théorème 3. *Le protocole d'identification de Schnorr à vérifieur honnête atteint la propriété de* ***super-validité****.* ♢

Démonstration. Supposons que le prouveur ait engagé un secret x, nous avons $b = g^a \bmod p$ pour $a \in \mathbb{Z}_q^*$. De plus, le vérifieur a posé deux questions c et c', $c \neq c'$ auxquelles le prouveur a répondu en utilisant deux fois l'engagement b. Les réponses correspondantes sont $d = a - c \cdot x \bmod q$ et $d' = a - c' \cdot x \bmod q$. Nous sommes alors capables de retrouver le secret x :

$$x = \frac{d - d'}{c' - c} \bmod q.$$

□

Le protocole d'identification de Schnorr est une $\mathcal{HV} - \mathcal{ZKPK}$. Ce schéma permet à un simulateur de générer un triplet (b, c, d) valide sans connaître x. Pour cela il suffit de choisir aléatoirement c et d puis de calculer $b = g^d \cdot y^c$.

3.2.2 Preuve de connaissance d'une représentation

Le protocole de Schnorr que nous venons de voir peut-être généralisé afin d'obtenir une preuve de connaissance sans divulgation d'une représentation dans une base $(g_1, \cdots, g_m)$ d'un élément public y (avec $(g_1, \cdots, g_m)$ une famille de générateurs q dans $\mathbb{Z}_p^*$). Le secret est ici le

m-uplet $(x_1, \cdots, x_m)$ tel que $y = \prod_{i=1}^{m} g_i^{x_i} \bmod p$. Nous notons une telle preuve de la manière suivante.

$$\textsc{Pok}\Big(x_1, \cdots, x_m : y = \prod_{i=1}^{m} g_i^{x_i}\Big)$$

Le protocole se déroule de la façon suivante.

Construction 11 ($\mathcal{HV}-\mathcal{ZKPK}$ et $\mathcal{WIPK}$ d'une représentation).

- Init. On choisit deux nombres premiers p et q tels que q divise $p-1$. On choisit aléatoirement une famille de générateurs $g_1, \cdots, g_m$ d'ordre q dans $\mathbb{Z}_p^*$.
- GénClé. Le prouveur génère la valeur publique $y = \prod_{i=1}^{m} g_i^{x_i} \bmod p$ correspondant à son secret $x = (x_1, \cdots, x_m) \in \mathbb{Z}_q^{*m}$.
- Prouve. Le protocole interactif de preuve se déroule de la façon suivante.

  Engagement. Le prouveur choisit $a_1, \cdots, a_m$ aléatoirement dans $\mathbb{Z}_q^*$ puis génère un engagement $b = \prod_{i=1}^{m} g_i^{a_i} \bmod p$ qu'il envoie au vérifieur.

  Question. Une fois l'engagement reçu, le vérifieur génère un aléa c qui constitue la question qu'il envoie au prouveur.

  Réponse. Le prouveur calcule ensuite pour tout $i \in [1,m], d_i = a_i - c \cdot x_i \bmod q$. Il obtient ainsi la réponse $d = \{d_i\}_{i\in[1,m]}$ reliée aux aléas $a_1, \cdots, a_m$, à la question c et au secret x. Le vérifieur peut se convaincre que le prouveur connait bien la représentation en testant si $b = y^c \prod_{i=1}^{m} g_i^{d_i} \bmod p$. ↻

Théorème 4. *Ce protocole conserve la propriété de* ***super-validité*** *du protocole d'identification de Schnorr à vérifieur honnête dont il est inspiré.* ♢

Démonstration. Supposons que le prouveur ait engagé un secret $x = (x_1, \cdots, x_m) \in \mathbb{Z}_q^{*m}$, nous avons $b = \prod_{i=1}^{m} g_i^{a_i} \bmod p$ avec $\forall i, a_i \in \mathbb{Z}_q^*$. De plus, le vérifieur a posé deux questions c et c', $c \neq c'$ auxquelles le prouveur a répondu en utilisant deux fois l'engagement b. Les réponses correspondantes sont $d = \{d_i = a_i - c \cdot x_i \bmod q\}_{i\in[1,m]}$ et $d' = \{d'_i = a_i - c' \cdot x_i \bmod q\}_{i\in[1,m]}$. Nous sommes alors capables de retrouver le secret x de la façon suivante :

$$\forall i \in [1,m], \quad x_i = \frac{d_i - d'_i}{c' - c}.$$

□

Comme dans le protocole de Schnorr, ce schéma permet à un simulateur de générer un triplet (b, c, d) valide sans connaître x. Pour cela il suffit de choisir aléatoirement c et $d = \{d_i\}_{i\in[1,m]}$ puis de calculer $b = y^c \prod_{i=1}^{m} g_i^{d_i} \bmod p$.

D'autre part, cette preuve est aussi une preuve de connaissance à témoin indistinguable, $\mathcal{WIPK}$. Un attaquant connaissant tous les témoins possibles pour la preuve, c'est-à-dire tous les m-uplet $(t_1, \cdots, t_m)$ tels que $y = \prod_{i=1}^{m} g_i^{t_i} \bmod p$, n'est pas capable de distinguer lequel est celui que le prouveur connait.

3.2.3 Preuve de connaissance d'égalité de logarithmes discrets

Une autre preuve basée sur le protocole de Schnorr permet de prouver l'égalité de deux logarithmes discrets dans deux bases différentes. Le prouveur engage les deux valeurs en utilisant le même aléa. Nous notons une telle preuve de la manière suivante.

$$\textsc{Pok}\Big(x : y_1 = g_1^x \wedge y_2 = g_2^x\Big)$$

Plus formellement, nous avons le protocole suivant.

Construction 12 ($\mathcal{HV}-\mathcal{ZKPK}$ d'égalité de logarithmes discrets).

- Init. On choisit deux nombres premiers p et q tels que q divise $p-1$. On choisit aléatoirement deux générateurs g_1, g_2 d'ordre q dans $\mathbb{Z}_p^*$.
- GénClé. Le prouveur génère les valeurs publiques $y_1 = g_1^x \bmod p$ et $y_2 = g_2^x \bmod p$ qui correspondent à son secret $x \in \mathbb{Z}_q^*$.
- Prouve. Le protocole interactif de preuve se déroule de la façon suivante.

  Engagement. Le prouveur choisit a aléatoirement dans $\mathbb{Z}_q^*$ puis génère les deux engagements suivants $b_1 = g_1^a \bmod p$ et $b_2 = g_2^a \bmod p$. Il envoie ensuite ces deux engagements au vérifieur.

  Question. Une fois les engagements reçus, le vérifieur génère un aléa c qui constitue la question posée au prouveur.

  Réponse. Le prouveur calcule ensuite une réponse $d = a - c \cdot x \bmod q$ reliée à l'aléa a, à la question c et au secret x. Le vérifieur peut se convaincre que les deux logarithmes discrets connus du prouveur sont égaux en testant si $b_1 = y^c g_1^d \bmod p$ et $b_2 = y^c g_2^d \bmod p$. ↻

Théorème 5. *Ce protocole conserve la propriété de* ***super-validité*** *du protocole d'identification de Schnorr à vérifieur honnête dont il est inspiré.* ♢

La **super-validité** de ce protocole se démontre comme pour le protocole d'identification de Schnorr.

De même, un simulateur peut générer un quadruplet (b_1, b_2, c, d) valide en utilisant la méthode vue pour le protocole original.

D'autre part, le prédicat "le prouveur connait un secret qui est d'une part le logarithme discret de y_1 en base g_1 et d'autre part le logarithme discret de y_2 en base g_2" ne compte qu'un seul témoin : x. Nous ne sommes donc pas dans le cadre d'une preuve de connaissance à témoin indistinguable, $\mathcal{WIPK}$.

3.2.4 Preuve de connaissance d'un élément OU d'un autre

Une preuve du "OU" est une preuve de connaissance permettant au prouveur de convaincre le vérifieur qu'il connait une (ou plusieurs) valeurs parmi un ensemble, sans révéler la ou lesquelles. Nous commençons par le cas ou le prouveur connait le logarithme discret d'une valeur parmi un ensemble. Nous notons une telle preuve de la manière suivante.

$$\textsc{Pok}\Big(x : \bigvee_{i=1}^{n} y_i = g^x\Big)$$

On attribue à un manuscrit non publié de B.Schoenmakers, la preuve de connaissance $\mathcal{HV}-\mathcal{ZKPK}$ et $\mathcal{WIPK}$ d'un logarithme discret d'une valeur "OU" d'une autre que nous allons décrire ici. Il est à noter que celle-ci est basée sur la preuve d'identification de Schnorr que nous avons vue plus haut. Plus formellement, nous avons le protocole suivant.

Construction 13 (Preuve du OU $\mathcal{HV}-\mathcal{ZKPK}$ et $\mathcal{WIPK}$ de B.Schoenmakers).

- Init. On choisit deux nombres premiers p et q tels que q divise $p-1$. On choisit aléatoirement un générateur g d'ordre q dans $\mathbb{Z}_p^*$.

– GénClé. Le prouveur choisit un indice j dans $[1, n]$ qui sera l'indice de l'élément qu'il connait. On pose I l'ensemble des indices de $[1, n]$ différent de j. Le prouveur génère la valeur publique $y_j = g^x \bmod p$ correspondante à son secret $x \in \mathbb{Z}_q^*$. Il choisit ensuite $\{y_i\}_{i \in I}$ des éléments de $< g >$ [2].
– Prouve. La preuve interactive se déroule de la façon suivante.

Engagement. Le prouveur choisit aléatoirement $a_j \in \mathbb{Z}_q$ et génère l'engagement $b_j = g^{a_j} \bmod p$. D'autre part il simule les autres engagements en choisissant, pour tout $i \in I$ les valeurs c_i et d_i aléatoirement dans $\mathbb{Z}_q$ et en calculant les engagement correspondants $b_i = y_i^{c_i} g^{d_i} \bmod p$.

Enfin, il envoie au vérifieur les engagements $\{b_i\}_{i \in [1,n]}$.

Question. Une fois l'engagement reçu, le vérifieur envoie au prouveur un aléa $c \in \mathbb{Z}_q$.

Réponse. Le prouveur calcule $c_j = c \bigoplus_{i \in I} c_i$ et $d_j = a_j - x c_j \bmod q$. Il obtient ainsi un ensemble de réponses $(c_i, d_i)_{i \in [1,n]}$ qu'il envoie au vérifieur.

– Vérifie. Le vérifieur peut se convaincre que le prouveur connait un des logarithmes discrets en vérifiant que $c = \bigoplus_{i \in [1,n]} c_i$ et que, pour tout $i \in [1, n], b_i = y_i^{c_i} g^{d_i} \bmod p$. ⟲

Théorème 6. *Ce protocole conserve la propriété de* ***super-validité*** *du protocole d'identification de Schnorr à vérifieur honnête dont il est inspiré.* ♢

La **super-validité** de ce protocole se démontre de la même façon que pour le protocole d'identification de Schnorr.

De même, un simulateur peut générer un quadruplet $(\{b_i\}_{i \in [1,n]}, c, \{c_i\}_{i \in [1,n]}, \{d_i\}_{i \in [1,n]})$ valide en utilisant la méthode vue pour le protocole original pour obtenir les triplet (b_i, c_i, d_i) pour tout $i \in [1, n]$, puis en calculant c à l'aide d'un XOR sur les c_i pour tout $i \in [1, n]$.

D'autre part ce protocole est un protocole de preuve de connaissance à témoin indistinguable, $\mathcal{WIPK}$. Un attaquant connaissant tous les témoins possibles pour la preuve, c'est-à-dire les logarithmes discrets de toutes les valeurs publiques $\{y_i\}_{i \in [1,n]}$, n'est pas capable de distinguer lequel est celui que le prouveur connait.

Remarque. Une généralisation de ce type de preuve du "OU" est possible. Par exemple, R.Cramer, I.Damgård et B.Schoenmakers se sont intéressés dans [CDS94] aux preuves de connaissances $\mathcal{HV} - \mathcal{ZKPK}$ du logarithme discret de d valeurs parmi n, le nombre d de valeurs ciblées étant public mais les d valeurs ciblées parmi les n possibles restant secrète.

3.3 Preuves de connaissance non-interactives

Pour certaines applications, il est nécessaire d'éviter tout échange entre le prouveur et le vérifieur. La preuve étant vérifiée a posteriori, il est important d'y inclure une notion de non-répudiation. L'objectif est alors un protocole de preuve non-interactif pour lequel le prouveur ne peut renier la connaissance du secret a posteriori. A.Fiat et A.Shamir ont proposés en 1986 une heuristique permettant de transformer une preuve de connaissance sans divulgation en trois passes, sûre face à un vérifieur honnête, en une preuve de connaissance non interactive [CS97]. Une seconde notion très proche des preuves de connaissance non-interactives est la notion de

2. Ces éléments peuvent avoir été générés en dehors de cette étape, il faut cependant que ni un tiers ni le vérifieur n'ait vu l'ensemble $\{y_i\}_{i \in I}$ afin qu'il ne puisse pas déduire l'élément y_j cachant le secret.

signature de connaissances. Ces signatures permettent de signer un message de telle sorte que l'auteur soit authentifié par la connaissance du (ou des) secret(s) qui suivent le prédicat prouvé. Il est à noté que l'heuristique de Fiat-Shamir [CS97] permet aussi de transformer une preuve de connaissance sans divulgation en trois passes, sûre face à un vérifieur honnête, en une telle signature de connaissance.

De manière plus formelle, nous définissons ces deux concepts de la façon suivante.

Définition 38 (Preuve de connaissance non-interactive).
On appelle preuve de connaissance non-interactive, noté $\mathcal{NIPK}$, un protocole non-interactif permettant à un prouveur de convaincre un tiers qu'il connait un secret x lié à un prédicat P. Un tel protocole doit assurer les deux mêmes propriétés que les protocoles non-interactifs, à savoir la **complétude** et la **validité** comme définies en Définition 33.

De plus, une preuve de connaissance non-interactive sans divulgation, notée $\mathcal{NIZKPK}$, devra respecter la propriété de **Sans divulgation** (comme défini en Définition 34 et une preuve de connaissances à témoin indistinguable, notée $\mathcal{NIWIPK}$, devra respecter la propriété de **Témoin indistinguable**.

Nous notons une telle preuve de connaissance de valeurs secrètes vérifiant un prédicat de manière similaire à la version interactive.

$$\textsc{Pok}\Big(\text{ valeurs secrètes } : \text{ prédicat }\Big) \qquad *$$

Définition 39 (Signature de connaissance).
On appelle signature de connaissance un protocole non-interactif permettant à un prouveur d'une part de convaincre un tiers qu'il connait un secret x suivant un prédicat P et d'autre part d'assurer qu'il est l'auteur d'un message m en assurant l'intégrité de ce dernier.

Un tel protocole est dit **objectif − moyen** sûr, avec **objectif** $\in \{\mathbf{UBK}, \mathbf{UF}, \mathbf{SU}, \mathbf{EU}\}$ et **moyen** $\in \{\mathbf{NMA}, \mathbf{KMA}, \mathbf{CMA}\}$, s'il assure les propriétés de **Complétude** et de **Validité** comme définies en Définition 33 et qu'il résiste à un attaquant ayant pour but **objectif** avec les moyens **moyen** comme défini pour les signatures.

Nous notons une telle signature de connaissance de valeurs secrètes vérifiant un prédicat sur un message comme suit.

$$\textsc{Sok}\Big(\text{ valeurs secrètes } : \text{ prédicat }\Big)(\mathsf{m}) \qquad *$$

Le principe de l'heuristique Fiat-Shamir est d'utiliser une fonction de hachage H et de remplacer la question c qui est aléatoirement choisie par le vérifieur dans un protocole interactif par un haché de l'engagement, de variable publiques, et dans le cas des signatures de connaissances, du message. L'idée est que si la fonction de hachage se comporte comme une fonction aléatoire, alors la question générée non-interactivement sera aussi aléatoire que la question envoyée par le vérifieur honnête. Trouver la réponse à l'une ou l'autre de ces deux questions est alors aussi difficile. D'autre part, si la fonction de hachage est résistante aux collisions, alors il sera difficile de falsifier la signature.

En utilisant cette heuristique, nous pouvons transformer le protocole d'identification de Schnorr, que nous avons décrit plus tôt, en une signature de connaissance de la façon suivante.

Construction 14 (Signature de connaissance de Schnorr).

– Init. On choisit deux nombres premiers p et q tels que q divise $p-1$. On prend ensuite aléatoirement un élément g d'ordre q dans $\mathbb{Z}_p^*$. Enfin, on choisit H une fonction de hachage.

– GénClé. Le prouveur génère la clé publique $\mathsf{pk} = g^x \bmod p$ correspondant à son secret $x \in \mathbb{Z}_q^*$ qui sera sa clé secrète.
– Signe. On retrouve dans l'algorithme de signature les trois phases du protocole interactif.

  Engagement. Le prouveur choisit a aléatoirement dans $\mathbb{Z}_q^*$ puis génère un engagement $b = g^a \bmod p$.

  Question. Le prouveur calcule $c = H(\mathsf{m}\|\mathsf{pk}\|p\|q\|b)$ qui sert de question.

  Réponse. Le prouveur calcule ensuite une réponse $d = a - c \cdot x \bmod q$ reliée à l'aléa a, à la question c et au secret x. La signature de connaissance sur le message m est $\sigma = (b, d)$.
– Vérifie. Le vérifieur recalcule $c = H(\mathsf{m}\|\mathsf{pk}\|p\|q\|b)$ puis vérifie si $b = g^d \cdot y^c \bmod p$. Si c'est le cas, le prouveur connait le secret. ↺

La preuve de sécurité de cette signature a été présentée par D.Pointcheval et J.Stern dans [PS00].

Conclusion

J'ai exposé dans ce chapitre les principales preuves de connaissance que j'ai utilisées pour mes divers travaux. Il manque deux types de preuves régulièrement utilisées en cryptographie auxquelles nous allons nous intéresser maintenant plus en détail. Tout d'abord les preuves d'appartenance à un intervalle qui seront décrites au chapitre suivant. Celles-ci permettent de prouver que le secret que nous connaissons appartient bien à un intervalle publiquement connu. Nous aborderons ensuite les preuves de connaissance non-interactives d'un secret (ou d'un ensemble de secrets) pour lesquelles le prédicat est une équation bilinéaire, sous la forme des preuves dites de Groth-Sahaï du nom de leurs inventeurs J.Groth et A.Sahai [GS07, GS08].

Chapitre 4

Preuve de connaissance d'un secret dans un intervalle

Sommaire

Nous allons à présent nous intéresser aux preuves de connaissance qui permettent à un prouveur d'attester que son secret appartient à un intervalle d'entiers public. Ce type de preuve est utilisé dans certaines propositions de monnaie électronique [CHL05] ou de coupon électronique [CES+05, CGH06] dans lesquelles il est nécessaire de prouver que la pièce ou le coupon que l'on utilise appartient à l'intervalle des pièces ou des coupons qui n'ont pas encore été utilisés. Certaines solutions de vote électronique [CGS97, BFP+01] utilisent ces preuves de connaissances afin de vérifier qu'un vote appartient à l'ensemble des candidats valides numérotés de 1 à n. Enfin, elles sont aussi utilisées dans le contexte des attestations anonymes [CL04] qui permettent à un individu détenant une attestation sur un ensemble d'attributs de justifier qu'il a une attestation valide tout en ne révélant que les champs pertinents pour chaque échange. Par exemple une attestation pourra être une carte d'identité, chaque entrée (nom, prénom, âge, nationalité, etc) représentant un attribut. Le détenteur de la carte pourra alors utiliser les preuves de connaissance qu'un secret appartient à un intervalle pour prouver qu'il a l'âge nécessaire et suffisant pour accéder à un service sans pour autant révéler son âge exact (par exemple qu'il a entre 18 et 25 ans pour bénéficier d'un tarif préférentiel).

Ce chapitre retrace mes travaux avec S.Canard, I.Coisel et J.Traoré, ceux-ci ont donné lieu à un article actuellement en soumission. Afin de simplifier la lecture, je joins cet article en Annexe A.

Hypothèse de travail. Par défaut, nous supposerons dans ce chapitre que le secret x dont on souhaite prouver la connaissance appartient à $\mathbb{Z}_q^*$ pour q premier et qu'il se situe dans l'intervalle d'entiers noté $[a, b]$ avec $a, b \in \mathbb{Z}_q^*$. Lorsque cette hypothèse est possible, elle permet la mise en place de schémas plus efficaces.

De plus, nous supposons que le secret x est engagé avec une mise en gage de Pedersen [Ped92] : $C = g^x h^r$. Il existe d'autres solutions, le vote électronique [BFP+01, CGS97] utilise, par exemple, un chiffré du secret à la place d'un engagement. Mais ce choix permet d'uniformiser les solutions que nous présentons afin de mieux comparer leur efficacité.

Enfin, nous nous plaçons dans le contexte des intervalles d'entiers. Afin de simplifier la lecture, nous parlons dans ce chapitre "d'intervalle", sans préciser à chaque fois qu'il s'agit d'intervalle d'entiers.

4.1 État de l'art

De nombreux travaux ont été effectués sur les preuves de connaissance d'un secret appartenant à un intervalle. Certaines des ces solutions, notamment [BCDv88] et [CFT98] proposent des solutions efficaces mais qui ne considèrent pas un intervalle exact. Nous limiterons ici notre étude aux solutions considérant un intervalle exact, c'est-à-dire aux preuves pour lesquelles le vérifieur a bien l'assurance que le secret x appartient à l'intervalle $[a, b]$. Afin d'avoir un aperçu global des solutions proposées dans la littérature nous regroupons les solutions en fonction des méthodes qu'elles utilisent.

4.1.1 Décomposition en somme de carrés

La première famille de solutions, introduite par F.Boudot dans [Bou00], propose d'utiliser le fait que $a \leq x \leq b$ si et seulement si $x - a \geq 0$ et $b - x \geq 0$ puis de prouver que $x_a = x - a$ (resp. $x_b = b - x$) est une valeur positive (ou nulle) en utilisant une caractérisation basée sur une somme de carrés.

La première solution proposée par F.Boudot dans [Bou00] utilise la caractérisation suivante.

Lemme 1. *Soient X et L deux entiers, avec $X < L$.*

$$0 \leq X \iff \exists (x_1, x_2) \in \mathbb{Z}^2 \text{ tel que } X = x_1^2 + x_2 \text{ et } 0 \leq x_2 < 2\sqrt{L}. \qquad \diamondsuit$$

Une seconde caractérisation est proposée par F.Boudot dans [Bou00], la solution correspondante ayant été développée par H.Lipmaa dans [Lip03]. Cette caractérisation en somme de quatre carrés se présente de la façon suivante.

Lemme 2. *Soit X un entier. X est positif ou nul si, et seulement si, il peut s'écrire comme la somme de quatre carrés.*

$$X \geq 0 \iff \exists (x_1, x_2, x_3, x_4) \in \mathbb{Z}^4 \text{ tel que } X = x_1^2 + x_2^2 + x_3^2 + x_4^2. \qquad \diamondsuit$$

Bien que cette caractérisation semble plus complexe que la première, l'algorithme de J.Rabin et J.Shallit [RS85] permet d'obtenir une décomposition en somme de quatre carrés de manière efficace. Elle est donc aisément utilisable en pratique.

Enfin, J.Groth a proposé dans [Gro05] une variante plus efficace utilisant une décomposition du secret en somme de trois carrés lorsque le secret à une certaine forme. Un théorème classique de théorie des nombres énonce que tout entier positif est la somme de trois carrés si et seulement s'il n'est pas de la forme $4^n(8k+7)$. Or il n'existe pas de $X = 4x+1$ de la forme $4^n(8k+7)$ et pour tout x entier, $4x+1$ est négatif si et seulement si x est négatif. J.Groth propose donc de poser $X_a = 4x_a+1$ (resp. $X_b = 4x_b+1$) et de prouver que X_a (resp. X_b) est positif en montrant qu'il s'agit d'une somme de trois carrés. La caractérisation correspondante est la suivante.

Lemme 3. *Soit X un entier différent de $4^n(8k+7)$ pour tout $n, k \in \mathbb{N}$. X est positif ou nul si, et seulement si, il peut s'écrire comme la somme de trois carrés.*

$$X \geq 0 \iff \exists (x_1, x_2, x_3) \in \mathbb{Z}^3 \text{ tel que } X = x_1^2 + x_2^2 + x_3^2. \qquad \diamondsuit$$

Lorsqu'une telle décomposition existe, l'algorithme de J.Rabin et J.Shallit [RS85] permet d'obtenir une telle décomposition en somme de trois carrés de manière efficace.

Pour ces trois caractérisations, la preuve de connaissance sans divulgation d'un secret x appartenant à un intervalle $[a, b]$ est alors composée de deux preuves, une prouvant que $x_a \geq 0$ et une seconde prouvant que $x_b \geq 0$. Chacune de ces deux preuves est elle-même composée d'une preuve de connaissance $\mathcal{ZKPK}$ des valeurs (x_1, x_2), (x_1, x_2, x_3, x_4) ou (x_1, x_2, x_3) et d'une seconde preuve $\mathcal{ZKPK}$ montrant qu'en utilisant les valeurs engagées pour la preuve précédente, il est possible de reconstituer le secret x_a (resp. x_b).

Remarque de sécurité. C'est trois caractéristiques sont utilisés dans les preuves correspondantes au niveau de l'exposant c'est à dire modulo l'ordre $|\mathbb{G}|$ du groupe $\mathbb{G}$ dans lequel se déroule le protocole.

Elles ne peuvent donc pas être appliquées dans un groupe d'ordre connu. En effet, si l'ordre est connu, il est possible d'obtenir la décomposition en somme de quatre carrés pour un entier x négatif. La décomposition étant faite modulo $|\mathbb{G}|$, il suffit alors de considérer $x + |\mathbb{G}| = x \bmod |\mathbb{G}|$. Or, $x + |\mathbb{G}|$ étant un entier positif, il est possible d'en trouver une décomposition en somme de carrés. En conséquence, il est possible de prouver que $x \bmod |\mathbb{G}|$ est positif si l'on connait $|\mathbb{G}|$ alors que x est négatif. De manière plus générale, ceci permet de prouver qu'un secret $x \notin [a, b]$ appartient à l'intervalle $[a, b]$.

Exemple.

Pour cet exemple, nous nous plaçons dans un groupe $\mathbb{G}$ d'ordre q.

Soit $0 < x < a$ un entier, le prouveur s'engage sur $x \notin [a, b]$. Les calculs étant effectués modulo q, il peut s'intéresser à $y = x - a + q$ plutôt qu'à $x - a$. Or $y = x - a + q$ est positif, il génère donc une décomposition en quatre carrés (y_1, y_2, y_3, y_4) de $y = x - a + q$ et donc une décomposition en quatre carrés de $x - a$ modulo q.

Le prouveur peut prouver qu'il connait (y_1, y_2, y_3, y_4) tel que $x-a = y_1^2+y_2^2+y_3^2+y_4^2 \bmod q$. Et donc prouver que $x > a$ alors que $0 < x < a$. •

Ainsi la proposition de T.H.Yuen, Q.Huang, Y.Mu, W.Susilo, D.Wong et G.Yang [YHM+09], qui utilise la caractérisation du Lemme 2, nécessiterait de se placer dans un groupe $\mathbb{G}$ d'ordre inconnu. Or la sécurité de leur proposition repose sur le problème SD (voir Définition 19), ils proposent donc de se placer dans un groupe $\mathbb{G}$ d'ordre connu. Malheureusement, dans ce cas, l'attaque que nous venons de voir fonctionne et la preuve qu'ils proposent ne peut donc pas prouver l'appartenance à un intervalle.

4.1.2 La décomposition en base u

Il est connu que n'importe quel entier peut être décomposé en n'importe quelle base u. Nous avons donc le Lemme suivant.

Lemme 4 (Décomposition en base u). *Soient x et u deux entiers avec u positif.*

$$x \in [0, u^\ell[\Longleftrightarrow x = \sum_{i=0}^{\ell-1} x_i u^i \text{ avec } \forall i \in [0, \ell[, x_i \in [0, u[$$

Nous notons $x = [x_0, \ldots, x_{\ell-1}]_u$. ♢

Cette décomposition a été utilisée pour le cas binaire, c'est-à-dire lorsque $u = 2$, par M.Bellare et S.Goldwasser [BG97]. Leur solution propose ainsi de décomposer le secret et les bornes de l'intervalle de cette façon puis de les comparer bit par bit. Cependant cette solution ne fonctionne qu'avec des intervalles de la forme $[0, 2^k[$.

4.1.3 La double représentation en base u

B.Schoenmakers décrit pour sa solution [Sch01] la caractérisation donnée par le lemme suivant avec $u = 2$. J.Camenisch, R.Chaabouni et A.Shelat ont ensuite proposé dans [CCS08] la version générique.

Lemme 5. *Soient a, b, x, u quatre entiers appartenant à $\mathbb{Z}_q^*$ avec u un entier positif. Nous notons $B = b - a + 1$ et $X = x - a$. Soit k l'unique entier tel que $u^k \leq B \leq u^{k+1}$. Nous notons $B_0 = u^{k+1} - B$ et $Y = x - a + B_0$.*

$$x \in [a, b] \Longleftrightarrow X \in [0, u^{k+1}[\text{ et } Y \in [0, u^{k+1}[. \quad ♢$$

Démonstration. (Idée de la preuve) Nous écrivons $x \in [a, b]$ sous la forme $X \in [0, B[$ avec $B = b - a + 1$ et $X = x - a$. Soit k l'unique entier tel que $u^k \leq B \leq u^{k+1}$. Nous notons $B_0 = u^{k+1} - B$, nous avons donc $X \in [0, u^{k+1}[\cap [-B_0, B[$. Or $B + B_0 = u^{k+1}$, donc nous pouvons réécrire cette équation de la façon suivante : $(X \in [0, u^{k+1}[) \wedge (X + B_0 \in [0, u^{k+1}[)$, ce qui est équivalent à $(X \in [0, u^{k+1}[) \wedge (Y \in [0, u^{k+1}[)$. □

À partir de [BG97] et du lemme 5, B.Schoenmakers [Sch01, Sch05] propose une méthode pour prouver l'appartenance d'un secret x à un intervalle $[a, b]$ quelconque. Le principe est de poser $B = b - a + 1$, $X = x - a$, k l'unique entier tel que $u^k \leq B \leq u^{k+1}$. On génère ensuite $B_0 = u^{k+1} - B$ et on obtient $Y = x - a + B_0$. En utilisant le lemme 5 avec $u = 2$, on obtient $x \in [a, b]$ si et seulement si $X \in [0, 2^{k+1}[$ et $Y \in [0, 2^{k+1}[$. On utilise ensuite la preuve de [BG97] qu'un secret appartient à un intervalle de la forme $[0, 2^{k+1}[$ a deux reprises : une première fois pour X puis une seconde pour Y. On obtient ainsi une preuve de connaissance sans divulgation que le secret x appartient à l'intervalle $[a, b]$.

4.1.4 Décomposition Multi-Bases

H.Lipmaa, N.Asokan et V.Niemi proposent dans [LAN02] une preuve de connaissance d'un secret appartenant à un intervalle de la forme $[0, v]$. Pour cela ils introduisent une nouvelle décomposition sur une base particulière.

Lemme 6 (Décomposition Multi-bases). *Soient v et x deux entiers de $\mathbb{Z}_q^*$ et $\ell = \lfloor \log_2 v \rfloor$.*

$$x \in [0, v] \Longleftrightarrow x = \sum_{i=0}^{\ell} v_i x_i \text{ avec } \forall i \in [0, \ell], x_i \in \{0, 1\}, v_i = \lfloor (v + 2^{\ell - i})/2^{\ell - i + 1} \rfloor. \quad \diamondsuit$$

Nous notons $x = [\![x_0, \ldots, x_\ell]\!]_v$.

Cette décomposition peut être considérée comme une généralisation de la décomposition binaire vue au Lemme 4 lorsque $x \in \mathbb{Z}_q^*$. En effet, pour prouver que $x \in [0, 2^k[$ en utilisant la décomposition multi-bases, nous avons $v = 2^k - 1$ et donc, pour tout $i \in [0, k-1], v_i = 2^i$. Nous avons alors la même décomposition dans les deux bases :

$$x = [\![x_0, \ldots, x_\ell]\!]_{2^k - 1} = [x_0, \ldots, x_{k-1}]_2$$

La preuve de connaissance d'un secret x appartenant à un intervalle $[0, v]$ de [LAN02] utilise cette caractérisation en engageant d'abord chaque x_i puis en utilisant les valeurs publiques v_i afin de prouver la relation $x = \sum_{i=0}^{\ell} v_i x_i$.

Cas général Comme expliqué par H.Lipmaa, N.Asokan et V.Niemi dans [LAN02], il est possible d'utiliser le même type de méthode pour prouver qu'un secret appartient à un intervalle de la forme $[a, b]$. Cela peut se faire en utilisant le Lemme suivant.

Lemme 7 (Cas général de la décomposition multi-bases). *Soient a, b et x trois entiers de $\mathbb{Z}_p^*$ et $\ell = \lfloor \log_2 b \rfloor$.*

$$x \in [a, b] \Longleftrightarrow x = \sum_{i=-1}^{\ell} b_i x_i \text{ avec } \begin{cases} \forall i \in [-1, \ell] & x_i \in \{0, 1\}, b_{-1} = a \\ \forall i \in [0, \ell] & b_i = \lfloor (b - a + 2^{\ell - i})/2^{\ell - i + 1} \rfloor \end{cases} \quad \diamondsuit$$

4.1.5 Caractérisation bit à bit

La caractérisation que nous allons présenter maintenant a été proposée par M.Fischlin [Fis01] pour le cas d'une décomposition binaire classique. L'idée de la solution reliée à cette caractérisation est de comparer le secret à chaque borne en comparant leurs représentations. Je décris cette caractérisation dans le cas de la décomposition multi-bases dont la décomposition binaire est un cas particulier pour $x \in \mathbb{Z}_q$ (voir Section précédente).

Lemme 8 (Caractérisation bit à bit). *Soient a et x deux entiers dont les décompositions multi-bases en base v sont notées $a = [\![a_0, \cdots, a_\ell]\!]_v$ et $x = [\![x_0, \cdots, x_\ell]\!]_v$.*

$$a < x \Longleftrightarrow \exists i' \in [0, \ell] \text{ tel que } a_{i'} = 0, x_{i'} = 1 \text{ et } \forall j > i', a_j = x_j$$

$\diamondsuit$

4.1.6 Caractérisation basée sur les signatures

Les travaux les plus récents sur le sujet sont basés sur l'utilisation de signatures. J.Camenisch, R.Chaabouni et A.Shelat proposent ainsi dans [CCS08] d'approfondir les travaux de I.Teranishi et K.Sako [TS06] dans ce domaine. L'idée est de désigner une autorité pour quelle génère une signature pour chaque élément de l'intervalle $[a, b]$. La preuve de connaissance d'un secret x dans un intervalle $[a, b]$ devient une preuve de connaissance d'une des signatures publiées sans révéler laquelle afin de cacher le secret connu. J.Camenisch, R.Chaabouni et A.Shelat affinent cette méthode. Pour prouver que $x \in [a, b]$ ils utilisent d'abord le Lemme 5 et se ramènent à deux

preuves qu'un secret X (resp. Y) appartient à un intervalle de la forme $[0, u^\ell]$. Ils utilisent alors le Lemme 4 pour représenter le secret X (resp. Y) en base u. Nous avons donc $X = [X_0, \ldots, X_{\ell-1}]_u$ (resp. $Y = [Y_0, \ldots, Y_{\ell-1}]_u$). Enfin, ils prouvent par l'utilisation de signatures que chaque X_i (resp. Y_i) de la représentation en base u du secret X (resp. Y) est dans l'intervalle $[0, u[$. Avec cette méthode, l'autorité désignée n'a plus que u signatures à publier, et celles-ci pourront être utilisées pour les deux preuves à effectuer, la preuve pour X et celle pour Y. Le principal problème de cette méthode réside dans la taille de la clé publique et dans la complexité en temps de l'initialisation.

De manière plus formelle, ce type de solution utilise la caractérisation suivante.

Lemme 9. *Soient a, b, x trois entiers. Pour tout $k \in [a, b]$, soit $\sigma_k = \textsc{Sign}(k)$ une signature de l'autorité désignée sur la valeur k. Soit Σ l'ensemble de toutes les signatures σ_k.*

$$x \in [a, b] \text{ si et seulement si } \exists \sigma \in \Sigma \text{ tel que } \sigma = \textsc{Sign}(x). \qquad \diamondsuit$$

Les auteurs de [TS06, CCS08] proposent d'utiliser les signatures courtes de D.Boneh et X.Boyen [BB08]. Ces signatures permettent de prouver la connaissance d'un message et de sa signature sans les révéler.

4.2 Nouvelle preuve d'appartenance d'un secret à un intervalle

Nous avons proposé avec S.Canard, I.Coisel et J.Traoré dans [CCJT], une nouvelle preuve sans divulgation qu'un secret appartient à un intervalle. L'idée initiale est d'utiliser la décomposition en multi-basse du secret vu au Lemme 6 puis d'appliquer la caractérisation bit par bit entre le secret et chacune des bornes de l'intervalle $[a, b]$ (voir Lemme 8). Pour cela, nous aurons besoin du Lemme suivant qui découle de la décomposition multi-bases (Lemme 6).

Lemme 10. $v = [\![1, \ldots, 1]\!]_v$

Démonstration. Nous prouvons le lemme par contradiction. Nous supposons que $v \neq [\![1, \ldots, 1]\!]_v$. Nous notons $u = [\![1, \ldots, 1]\!]_v \in [0, v[$ $(u \neq v)$. Comme $v \in [0, v]$ alors il existe $\tilde{v}_0, \ldots, \tilde{v}_\ell$ tels que $v = [\![\tilde{v}_0, \ldots, \tilde{v}_\ell]\!]_v$ et $\exists i_0 \in [0, \ell] / \tilde{v}_{i_0} = 0$. Donc nous avons $u = \sum_{i=0}^{\ell} v_i > \sum_{i=0}^{\ell} v_i \tilde{v}_i = v$. Ce qui contredit le fait que $u \in [0, v[$. □

Cependant, pour une valeur v, la décomposition multi-bases d'un entier x n'est pas unique. Par exemple pour $v = 10$, nous avons $v_3 = 5, v_2 = 3, v_1 = 1$ et $v_0 = 1$. L'entier $x = 6$ peut ainsi s'écrire $[\![0, 1, 0, 1]\!]_v$ ou $[\![1, 0, 0, 1]\!]_v$. Or pour une comparaison bit-à-bit nous avons besoin de l'unicité de la décomposition. Pour cela nous définissons un algorithme de décomposition (Algorithme 1) qui retourne une décomposition unique pour un entier donné.

Algorithme 1: $\textsc{MBDec}(x, v_0, \ldots, v_\ell)$

```
m = x;
pour i = ℓ à 0 par pas de −1 faire
    si m ≥ v_i alors
        x_i := 1;
        m := m − v_i
    sinon x_i := 0
retourner x = [[x_0, ..., x_ℓ]]_v;
```

Le principe de base de notre solution est ainsi de décomposer le secret dans une multi-base, en utilisant l'algorithme 1, puis de comparer bit-à-bit le secret et les bornes de l'intervalle. Cette solution naïve peut être affinée en remarquant que les bornes de l'intervalle sont des éléments publics. Nous pouvons ainsi améliorer cette solution en notant les points suivants.

1. Dans le cas $x \leq b$, si on applique le Lemme 6 en prenant $v = b$, il ne reste plus qu'à prouver que $x = [\![x_0, \ldots, x_\ell]\!]_b$ pour $\ell = \lfloor log_2 b \rfloor$. On obtient la preuve suivante.
   (a) **On engage chaque $x_i \in \{0,1\}$.** Pour cela, le prouveur choisit $r_0, \ldots, r_\ell$ aléatoirement dans $\mathbb{Z}_q$ et calcule $C_i = g^{x_i} h^{r_i}$ pour tout $i \in [0, \ell]$. Il envoie ensuite les engagements $\{C_i\}_{i \in [0,\ell]}$ au vérifieur.
   (b) **On prouve ensuite que $\forall i \in [0,\ell], x_i \in \{0,1\}$ et que $x = \sum_{i=0}^{\ell} b_i x_i$.** Pour cela , on remarque que le prédicat $L = (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \ldots \wedge (C_\ell = h^{r_\ell} \vee C_\ell/g = h^{r_\ell})$ permet de prouver la première partie et que, en posant $t = \sum_{i=0}^{\ell} b_i r_i$ et $\tilde{C} = \prod_{i=0}^{\ell} C_i^{b_i}$, prouver la seconde partie revient à prouver que $\tilde{C} = g^x h^t$. On a donc la preuve (intermédiaire) suivante.

   $$\textsc{Pok}(x, t, r_0, \ldots, r_\ell : (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \ldots \wedge (C_\ell = h^{r_\ell} \vee C_\ell/g = h^{r_\ell}) \wedge \tilde{C} = g^x h^t)$$

2. Dans le cas $a \leq x$, si on représente a en utilisant le Lemme 6, on obtient $a = [\![a_0, \ldots, a_\ell]\!]_b$ (puisque $a < b$). On peut à présent utiliser les représentations de a et x en multi-bases b pour les comparer.
   Pour cela, nous aurons besoin du Lemme suivant.

   Lemme 11. $\exists! i_0 \in [0,\ell] / a_{i_0} = 0 \wedge (\forall i > i_0, a_i = 1)$ ♢

   Démonstration. L'existence est donnée par le fait que $a \neq b$ et l'unicité est obtenue grâce à l'utilisation de l'Algorithme 1. □

   Le lemme 8 indique $\exists i' \in [0,\ell] / a_{i'} = 0$, $x_{i'} = 1$ et $\forall j > i', a_j = x_j$. Par simple déduction, nous avons $i_0 \geq i'$ donc pour tout $j > i_0$ nous avons $a_j = x_j = 1$. La borne inférieure de l'intervalle a est publique et i_0 ne dépend que de a donc nous ne divulguons aucune information sur le secret x en indiquant que pour tout $j > i_0$ nous avons $x_j = 1$.
   Nous pouvons ainsi nous dispenser de prouver par une preuve $ZKPK$ que pour tout $j > i_0, x_j = 1$ en ouvrant simplement les engagements correspondant $C_{i_0+1}, \ldots, C_\ell$.
3. Toujours dans le cas $a \leq x$, nous devons encore observer, et prouver, que les x_i pour tout $i \leq i_0$ sont compatibles avec l'assertion $a \leq x$. On peut simplifier la preuve restante en observant que le prouveur connait a, ce qui implique que pour le $i$ème bit, nous avons les possibilités suivantes.
   – Soit nous avons $a_i = 0$.
     Si $x_i = 1$, nous devons prouver que c'est le cas, puis nous pouvons nous arrêter car cela est suffisant pour attester que $a \leq x$.
     Sinon $x_i = 0$, nous devons prouver que c'est le cas, puis continuer avec les bits restant à comparer. Nous notons B le prédicat concernant la comparaison restant à effectuer.

   Comme nous sommes dans le contexte des preuves sans divulgation nous devons établir qu'il s'agit de l'un ou l'autre cas sans révéler lequel, nous devons donc prouver le prédicat suivant.

   $$(x_i = 1) \vee \big((x_i = 0) \wedge B\big)$$

– Soit nous avons $a_i = 1$ alors par construction $x_1 = 1$. Il faut donc l'établir avant de passer à la suite.

Plus formellement, la solution fonctionne de la façon suivante.

Construction 15 (Nouvelle solution sur la décomposition multi-base).
Soient b, $a = [\![a_0, \cdots, a_\ell]\!]_b$ et $x = [\![x_0, \cdots, x_\ell]\!]_b$ trois entiers tels que $a < b$ et $x \in [a, b]$.

– Init. L'initialisation consiste en la détermination de i_0 tel que $a_{i_0} = 0$ et $\forall i > i_0, a_i = 1$. Puis en l'exécution de l'Algorithme 2 nommé FGreat afin d'obtenir le prédicat sur les bits 0 à $i_0 - 1$ en utilisant comme entrées $\{a_0, \cdots, a_{i_0-1}\}$

Algorithme 2: FGreat$(\{a_0, \cdots, a_i\})$

si $(a_i = \ldots = a_0 = 0)$ **alors** $L := \emptyset$;
sinon
  si $(a_i = 1)$ **alors**
    si $(i = 0)$ **alors** $L := (x_0 = 1)$;
    sinon $L := (x_i = 1) \wedge [\text{FGreat}(\{a_0, \cdots, a_{i-1}\})]$;
  sinon $L := (x_i = 1) \vee [\text{FGreat}(\{a_0, \cdots, a_{i-1}\})]$;
retourner L;

– Prouve. Le protocole de preuve d'intervalle en lui-même correspond aux étapes suivantes entre le prouveur et le vérifieur.

1. Le prouveur choisit aléatoirement $r_0, \ldots, r_\ell \in \mathbb{Z}_q$ et calcule $C_i = g^{x_i} h^{r_i}$ pour tout $i \in [0, \ell]$. Il envoie ensuite l'ensemble $\{C_i\}_{i \in [0,\ell]}$ au vérifieur ainsi que les r_i pour tout $i \in [i_0 + 1, \ell]$.
2. Si et seulement si les x_i correspondent à la décomposition en multi-bases de x, le prouveur et le vérifieur peuvent tout deux reconstituer le même engagement $\tilde{C}$ sur le secret x . Le prouveur calcule $t = \sum_{i=0}^{\ell} b_i r_i$ et obtient $\tilde{C} = g^x h^t$. Tandis que le vérifieur calcule $\tilde{C} = \prod_{i=0}^{\ell} C_i^{b_i}$ avec les b_i de la décomposition multi-base.
3. Le prouveur et le vérifieur effectuent ensuite la preuve de connaissance sans divulgation suivante.

$$\begin{aligned} U_f \quad = \quad & \text{Pok}\Big(x, t, r_0, \ldots, r_{i_0} : \tilde{C} = g^x h^t \wedge (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \ldots \wedge \\ & (C_{i_0-1} = h^{r_{i_0-1}} \vee C_{i_0-1}/g = h^{r_{i_0-1}}) \wedge \big(C_{i_0}/g = h^{r_{i_0}} \vee (C_{i_0} = h^{r_{i_0}} \wedge L)\big)\Big) \end{aligned}$$

4.3 Comparaison de l'efficacité des solutions

Nous allons à présent comparer l'efficacité des diverses propositions que nous avons vues dans ce chapitre. Ce travail a été effectué en collaboration avec S.Canard, I.Coisel et J.Traoré pour l'article [CCJT]. Nous considérons que l'engagement sur le secret x a déjà été effectué, cette étape n'est donc pas intégrée aux résultats d'efficacité que nous avons obtenus.

Pour notre étude, nous nous plaçons, autant que possible, sur les courbes elliptiques. En effet, dans ce cadre, pour un même niveau de sécurité, les clés secrètes nécessaires sont plus courtes et les calculs plus légers car manipulant des éléments plus court. Nous utilisons alors la

multiplication modulaire dans $\mathbb{G}$, le groupe d'ordre q, comme opération élémentaire. Cependant, certaines solutions nécessitent de se placer sur $\mathbb{Z}_n$ avec n un module RSA (par exemple [Bou00, Gro05] ou [Lip03]). Nous avons donc utilisé, pour ces cas, des comparaisons entre les temps d'exécution des différentes opérations afin de les ramener à notre unité : le temps d'exécution d'une multiplication modulaire dans $\mathbb{G}$. Pour cela, nous nous sommes principalement basés sur les travaux de C.Arene, T.Lange, M.Naehrig et C.Ritzenthaler [ALNR09] sur certaines courbes elliptiques particulières (les courbes d'Edward), sur l'article [Möl01] de B.Möller étudiant les algorithmes de multi-exponentiation, sur la thèse de B.Lynn sur l'implémentation des couplages [Lyn07] et enfin sur des tests effectués avec la libraire Bouncycastle.

L'obtention des résultats de complexité étant particulièrement calculatoire, je n'en préciserai pas les détails. Ce qui nous intéresse ici est de comparer ces résultats afin de pouvoir observer quelles solutions sont les plus efficaces en fonction de la taille du secret et des bornes. Les résultats obtenus pour la complexité de chaque solution sont résumés dans la Table 4.1.

Méthode	Complexité en temps		Complexité en espace (en bits)	Taille de la clé publique (en bits)
	Prouver (mul. mod. sur $\mathbb{G}$)	Vérifieur (mul. mod. sur $\mathbb{G}$)		
Décomposition en somme de carrés [Lip03]	$(30.2l_{\mathbb{Z}_n} + 16.2l_e + 34.1l_s + 2\ell)(\frac{l_{\mathbb{Z}_n}}{\lvert q\rvert})^2$	$(17.1l_{\mathbb{Z}_n} + 17.1l_e + 21.1l_s + 2\ell + 6)(\frac{l_{\mathbb{Z}_n}}{\lvert q\rvert})^2$	$11\lvert\mathbb{Z}_n\rvert + 10l_{\mathbb{Z}_n} + 15l_e + 19l_s + \frac{5}{2}\ell$	$2\lvert\mathbb{Z}_n\rvert + l_{\mathbb{Z}_n} + l_s + l_e$
Décomposition en somme de carrés [Gro05]	$(8.75l_{\mathbb{Z}_n} + 16.7l_e + 16.7l_s + 7.9\ell + 7.9)(\frac{l_{\mathbb{Z}_n}}{\lvert q\rvert})^2$	$(9.4l_{\mathbb{Z}_n} + 16.3l_e + 13.3l_s + 4\ell + 6)(\frac{l_{\mathbb{Z}_n}}{\lvert q\rvert})^2$	$7\lvert\mathbb{Z}_n\rvert + 6l_{\mathbb{Z}_n} + 11l_e + 10l_s + 4\ell$	$7\lvert\mathbb{Z}_n\rvert + \ell_{\mathbb{Z}_n} + \ell_s + \ell_e$
Double représentation en base u [Sch01, Sch05]	$\frac{293\lvert q\rvert+12}{6}\ell + \frac{359\lvert q\rvert+3}{3}$	$\frac{175\lvert q\rvert+6}{3}\ell + \frac{125\lvert q\rvert}{2} + \frac{25}{3}\frac{2^{\ell+2}-1}{2^\ell}\lvert q\rvert$	$(6\lvert\mathbb{G}\rvert + 8\lvert q\rvert)\ell + (6\lvert\mathbb{G}\rvert + 4\lvert q\rvert)$	$4(\lvert\mathbb{G}\rvert + \lvert q\rvert)$
Décomposition Multi-Bases [LAN02]	$\frac{75\lvert q\rvert+1}{2}\ell + \frac{325\lvert q\rvert+2}{4}$	$\frac{175\lvert q\rvert+6}{6}\ell + \frac{1775\lvert q\rvert+48}{24} + \frac{25}{3}\frac{2^{\ell+3}-1}{2^{\ell+2}}\lvert q\rvert$	$(3\lvert\mathbb{G}\rvert + 4\lvert q\rvert)\ell + (5\lvert\mathbb{G}\rvert + 5\lvert q\rvert)$	$2\lvert\mathbb{G}\rvert + (\ell + 5)\lvert q\rvert$
Caractérisation basée sur les signatures [CCS08]	$1250\lvert q\rvert + 66.7k\lvert q\rvert - 1148$	$583.8\lvert q\rvert + 12.5u\lvert q\rvert + 10u + 1181k - 1150$	$(u + k + 4)\lvert\mathbb{G}\rvert + (2k + 2)\lvert\mathbb{G}_T\rvert + (4k + 8)q$	$4\lvert\mathbb{G}\rvert + (k + 3)q$
Notre solution Multi-Bases	$\left(\frac{175\lvert q\rvert+12}{96}\right)\ell^2 + \left(\frac{5275\lvert q\rvert+156}{96}\right)\ell + \left(\frac{1025\lvert q\rvert+12}{48}\right)$	$\left(\frac{525\lvert q\rvert+24}{12}\right)\ell - \left(\frac{75\lvert q\rvert+6}{12}\right) + \left(\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell+1}}\right)\lvert q\rvert$	$(4\lvert\mathbb{G}\rvert + 6\lvert q\rvert)\ell - 3\lvert q\rvert$	$2\lvert\mathbb{G}\rvert + \ell + \lvert q\rvert$

TABLE 4.1 – Comparaison de l'efficacité des solutions présentées

Afin de pouvoir comparer plus facilement les solutions, nous nous plaçons à un niveau de sécurité permettant une protection à long-terme. Cela correspond, en suivant les recommandations 2010 du réseau d'excellence ECRYPT II, à 128 bits de sécurité et donc aux valeurs suivantes : $|q| = 256$, $|\mathbb{G}| = 257$, $l_{\mathbb{Z}_n} = 3248$, $l_e = l_s = 160$. Nous obtenons ainsi la Table 4.2.

Notre hypothèse de travail implique que, lorsque cela est possible, la borne supérieure de l'intervalle b est un élément de $\mathbb{Z}_q$. C'est pourquoi je comparerai ici les différentes méthodes pour des tailles de borne supérieure $\ell = \log_2(b)$ comprises entre 1 et 250. Pour tous les graphiques de comparaison nous utilisons la même légende, définie en Figure 4.1.

Nous allons d'abord comparer les solutions en fonction de leur efficacité en vérification et en preuve avant de considérer la taille des données échangées et publiées.

4.3.1 Comparaison en fonction des capacités du vérifieur

L'efficacité des différentes solutions pour la vérification d'une preuve est représentée en Figure 4.2. Trois solutions ressortent en fonction de la taille de b. Pour $\ell > 25$, la meilleure solution est [CCS08]. Pour de petites valeurs, $\ell \leq 5$, la solution idéale est notre solution tandis que pour $5 < \ell \leq 25$, la solution [LAN02] est la plus efficace. Les solutions à base de décomposition en somme de carrés [Gro05, Lip03] sont peu efficaces par rapport aux autres solutions. Elles sont

Méthode	Complexité en temps		Complexité en espace (en bits)	Taille de la clé publique (en bits)
	Prouveur (mul. mod. sur $\mathbb{G}$)	Vérifieur (mul. mod. sur $\mathbb{G}$)		
Décomposition en somme de carrés [Lip03]	$(106099 + \frac{63}{32}\ell)(\frac{203}{16})^2$	$(123263 + \frac{127}{65}\ell)(\frac{203}{16})^2$	$73648 + \frac{5}{2}\ell$	10064
Décomposition en somme de carrés [Gro05]	$(\frac{540287+127\ell}{16})(\frac{203}{16})^2$	$(\frac{1126591+127\ell}{16})(\frac{203}{16})^2$	$35840 + 4\ell$	26304
Double représentation en base u [Sch01, Sch05]	$\frac{37510}{3}\ell + \frac{66307}{3}$	$\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell-8}} + \frac{44806}{3}\ell + 16000$	$3590\ell + 2566$	2052
Décomposition Multi-Bases [LAN02]	$\frac{19201}{2}\ell + \frac{45313}{2}$	$\frac{25}{3}\frac{2^{\ell+3}-1}{2^{\ell-2}} + \frac{22403}{2}\ell + \frac{56806}{3}$	$1795\ell + 2565$	$256\ell + 1794$
Caractérisation basée sur les signatures [CCS08]	$318852 + \frac{51200}{3}k$	$3210 * 2^{\ell/k} + \frac{7087}{6}k + 148290$	$257 * 2^{\ell/k} + 775k + 1807$	$257k + 1799$
Notre solution Multi-Bases	$\frac{11203}{24}\ell^2 + \frac{337639}{24}\ell + \frac{65603}{12}$	$\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell-3}} + 11202\ell + \frac{3201}{2}$	$1536\ell + 260$	$\ell + 770$

TABLE 4.2 – Comparaison de l'efficacité des solutions présentées pour 128 bits de sécurité

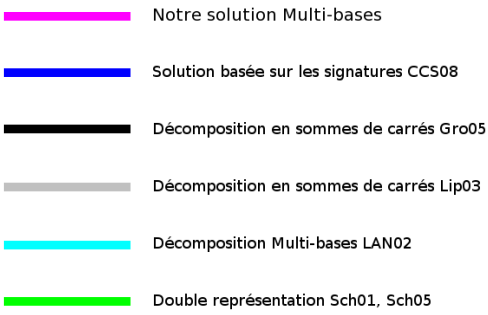

FIGURE 4.1 – Légende

en effet handicapées par la taille du groupe nécessaire à leur application.

4.3.2 Comparaison en fonction des capacités du prouveur

Nous allons à présent nous intéresser à la complexité côté prouveur, en nous référant à la Figures 4.3. Nous retrouvons dans ce cas la même distribution que lors de la comparaison précédente. La solution basée sur les signatures de [CCS08] est la plus intéressante pour une borne supérieure de grande taille, la limite se situant, ici, à 60 bits. La solution de [LAN02] étant, dans ce cas, optimale entre 60 et 3 bits. Enfin, notre solution est la plus efficace pour une très petite valeur, $\ell \leq 3$.

Les solutions à base de décomposition en sommes de carrés sont à nouveau mises à mal par la taille du groupe qu'elles impliquent.

On notera que l'on retrouve les mêmes solutions que pour l'efficacité en vérification, cependant les limites ont glissées. Ainsi entre 25 et 60 bits, notamment, un choix devra être effectué entre la solution la plus efficace pour prouver ou celle optimale pour vérifier en fonction de la puissance des parties en présence.

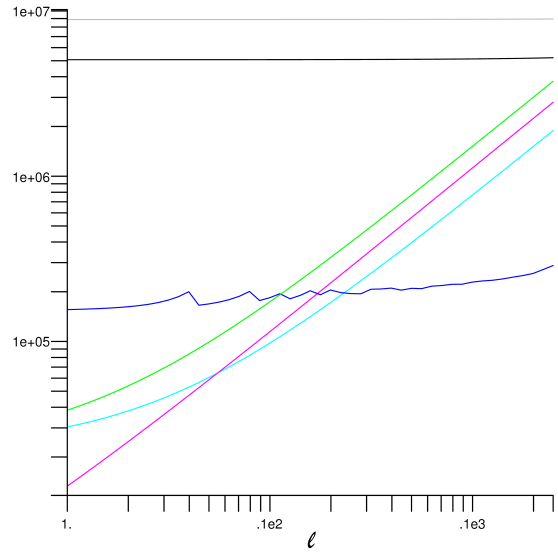

FIGURE 4.2 – Complexités du vérifieur pour $\ell \in [1, 250]$

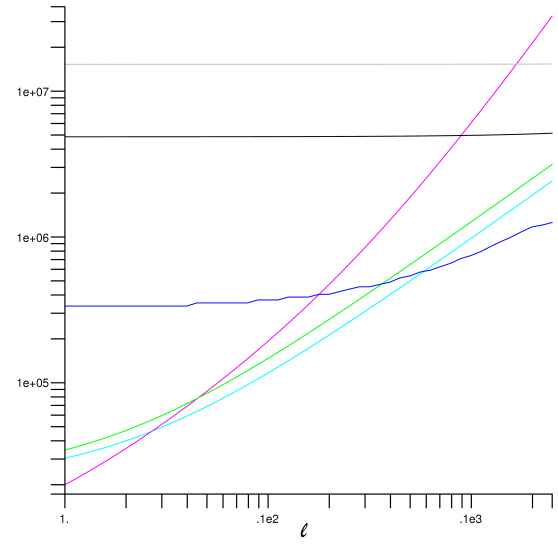

FIGURE 4.3 – Complexités du prouveur pour $\ell \in [1, 250]$

4.3.3 Complexité en espace des communications

Nous nous plaçons à présent du point de vue de la taille des éléments échangés entre le prouveur et le vérifieur, nous nous référons ici à la Figure 4.4.

Nous retrouvons à nouveau les trois solutions qui se sont distinguées en efficacité, à savoir notre solution pour de petites valeurs $\ell \leq 5$, [LAN02] pour des tailles intermédiaires comprises ici entre 5 et 14 bits, et en enfin [CCS08] pour de plus grandes valeurs. Cependant, ici la solution [CCS08] n'est plus la solution idéale pour toute taille supérieure à 14 bits. Au-dessus de 24 bits, la solution [Gro05] est à privilégier.

4.3.4 Taille des éléments publics

Enfin, nous nous plaçons du point de vue de la taille des éléments publics en nous référant à la Figure 4.5.

Les solutions [CCS08] et [LAN02] qui se sont distinguées par leur efficacité impliquent de publier une quantité relativement grande de valeurs publiques. Ainsi sur ce critère, notre solution est la plus efficace pour tout ℓ. Cependant cette solution, bien qu'idéale pour de petites valeurs, n'est pas conçue pour une borne supérieure de grande taille. En effet, la complexité de cette solution est polynomiale en ℓ et non linéaire comme cela est le cas pour les autres solutions. De ce fait, à partir de 90 bits, cette solution est la moins efficace de toutes pour la génération de preuves. Pour une borne supérieure de grande taille, il peut être préférable d'utiliser la double représentation [Sch01, Sch05]. Cette solution requiert la publication d'une plus grande quantité de données et est moins efficace que notre solution en vérification mais elle est du même ordre que [LAN02] pour l'efficacité côté prouveur.

4.3.5 Conclusion de cette comparaison

En conclusion, pour une valeur de borne supérieure petite ($\ell \leq 5$), notre solution est la plus indiquée, et cela quelle que soit la priorité. En ce qui concerne les valeurs plus grandes, un choix doit être effectué entre efficacité du protocole et quantité de valeurs à échanger ou stocker.

Si la priorité est l'efficacité, la solution [LAN02] sera à privilégier pour une borne supérieure entre 5 et 25 bits, et la solution [CCS08] pour une borne b de plus de 60 bits. Entre 25 et 60 bits, l'une ou l'autre de ces deux solutions pourra être choisie en fonction de l'accent souhaité entre efficacité de la génération ou de la vérification des preuves.

Si la complexité en communication est une priorité, la solution [LAN02] reste idéale pour une taille comprise entre 5 et 14 bits. Entre 14 et 25 bits, la solution [CCS08] sera optimale pour ce critère, tandis qu'au-dessus, la solution [Gro05] sera la moins coûteuse en échanges.

Enfin, si la quantité de valeurs publiques à stocker est un facteur prépondérant, la solution

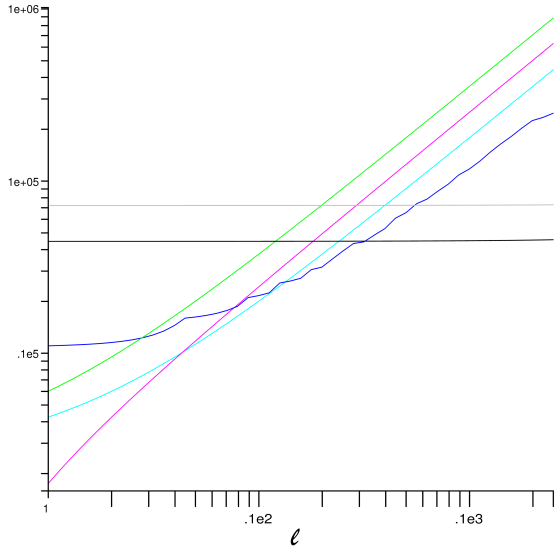

FIGURE 4.4 – Complexité en espace des communications pour $\ell \in [1, 250]$

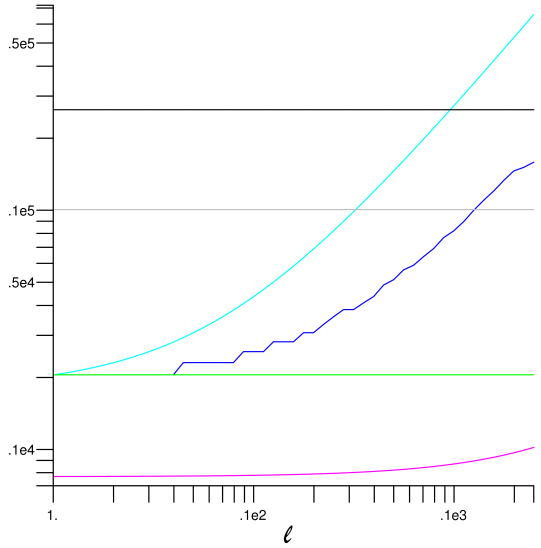

FIGURE 4.5 – Taille de la clé publique pour $\ell \in [1, 250]$

décrite en section 4.2 et, éventuellement, les solutions [Sch01, Sch05], pour un compromis sur l'efficacité du prouveur, sont à privilégier.

Remarque. Les solutions [Lip03] et [Gro05] sont moins efficaces ici du fait de la taille du groupe qu'elles requièrent. Cependant, si l'on reste au même niveau de sécurité mais que la taille de la borne supérieure dépasse $|q|$ bits, la taille des groupes utilisée dans les autres solutions devra être adaptée afin que le groupe comprenne toujours la borne supérieure. Ainsi, en supposant que la taille du groupe choisie est toujours la plus petit possible, et en considérant le niveau de sécurité de 128 bits dans lequel nous nous sommes placés, la solution [Gro05] sera la solution la plus efficace en preuve lorsque la borne supérieure dépassera 700 bits, cette solution restera cependant moins efficace que [CCS08] en ce qui concerne la vérification.

Conclusion

J'ai décrit dans ce chapitre les résultats que j'ai produit en collaboration avec S.Canard, I.Coisel et J. Traoré sur les preuves de connaissances dans un intervalle. Ceux-ci comprennent une nouvelle construction de preuve de connaissance dans un intervalle ainsi que la première étude comparative des différentes constructions de la littérature. En effet, il était supposé jusqu'ici que les constructions les plus récentes étaient plus adaptées dans tous les cas. Nous avons montré que la construction la mieux adaptée dépendait du système considéré, c'est-à-dire de la taille de l'intervalle, des capacités du vérifieur et du prouveur ainsi que des possibilités de stockage pour la clé publique. Nous avons enfin mis en avant les différentes constructions, dont la notre, à privilégier en fonction du contexte.

Nous allons à présent nous intéresser à un autre type de preuve de connaissance qui va nous permettre de prouver la connaissance d'un ensemble de secrets vérifiant une équation de couplage : les preuves dites Groth-Sahaï du nom de leurs inventeurs J.Groth et A.Sahaï.

Chapitre 5

Preuve de connaissance non-interactive et équations de couplages : Groth-Sahai

Sommaire

Les preuves de connaissance non-interactives proposées par J.Groth et A.Sahai dans [GS08, GS07], et révisées par E. Ghadafi, N.P. Smart et B. Warinschi dans [EGW09], sont principalement utilisées pour prouver la connaissance d'un ou plusieurs secrets qui respectent une équation de couplages. Cette solution permet de générer des preuves de connaissance non-interactives à divulgation nulle de connaissance mais aussi des preuves de connaissance non-interactives à témoin indistinguable. Pour chacune de ses possibilités, trois propositions sont développées dans l'article. Chaque proposition permet de reposer sur un des problèmes difficiles les plus courants de la cryptographie à base de couplage : SD, SXDH ou DLIN (voir Chapitre 1 Section 1.2.2).

En premier lieu, je donnerai une description succincte de ce type de preuve de connaissance. Nous nous intéresserons ensuite aux travaux que j'ai effectué en collaboration avec O.Blazy, G.Fuchsbauer, M.Izabachène, H.Sibert et D.Vergnaud [BFI$^+$10a, BFI$^+$10b]. Dans ce cadre, nous avons étudié l'utilisation de méthodes de rassemblement d'équations de couplage dans l'objectif d'améliorer l'efficacité de la vérification de telles preuves. Dans une dernière section, je donnerais un exemple pratique de signature de groupe utilisant les preuves Groth-Sahai afin de mieux identifier l'impact sur les solutions pratiques des méthodes que nous avons proposées dans [BFI$^+$10a, BFI$^+$10b].

5.1 Preuves de connaissance non-interactives Groth-Sahai

Je vais à présent décrire le principe des preuves de connaissance non-interactives Groth-Sahai : la procédure d'initialisation du schéma, la procédure de génération d'une preuve Groth-

Sahai et enfin la procédure de vérification d'une telle preuve.

5.1.1 Procédure d'initialisation

Durant l'initialisation, on détermine le problème difficile sur lequel notre protocole va reposer (SD, SXDH ou DLIN) afin de définir le couplage que l'on utilisera par la suite. On choisit ensuite, en fonction de l'architecture nécessaire au problème, le couplage qui y correspond et donc les groupes $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$, des générateurs pour chaque groupe g_1, g_2 et g_t ainsi qu'une forme bilinéaire $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

Chaque preuve $\mathcal{NIPK}$ utilise un ensemble de valeurs publiques spécifiques qui dépend du problème difficile sur lequel va reposer la preuve (SD, SXDH ou DLIN) et du souhait d'une mise en gage parfaitement **Indistinguable** (et calculatoirement **résistante aux collisions**) ou d'une mise en gage parfaitement **résistante aux collisions** (et calculatoirement **Indistinguable**). Pour rappel, un engagement ne peut être à la fois parfaitement **Indistinguable** et parfaitement **résistant aux collisions** (voir Chapitre 1 Section 1.3.5). Le Schéma de preuves Groth-Sahai permet d'atteindre une mise en gage parfaitement **Indistinguable** ou parfaitement **résistante aux collisions**, cependant il requiert que le choix soit effectué à l'initialisation car celui-ci détermine la valeur de certaines valeurs publiques utilisées non seulement pour les engagements mais aussi pour la génération et la vérification de preuve.

5.1.2 Procédure de génération

Par définition une preuve $\mathcal{NIPK}$ permet de prouver la connaissance de secrets de manière non-interactive. Ici, la ou les valeurs secrètes peuvent être de trois types : scalaires, éléments de $\mathbb{G}_1$ ou éléments de $\mathbb{G}_2$. Celles-ci sont regroupées en fonction de leur type et notées sous forme de vecteur de la façon suivante.

- Le vecteur $\vec{X}$ regroupe les éléments secrets de $\mathbb{G}_1$,
- le vecteur $\vec{Y}$ regroupe les éléments secrets de $\mathbb{G}_2$,
- le vecteur $\vec{x}$ regroupe les scalaires qui seront engagés dans $\mathbb{G}_1$,
- enfin, le vecteur $\vec{y}$ regroupe les scalaires qui seront engagés dans $\mathbb{G}_2$.

Le prouveur détermine ensuite le type le plus adapté à chaque équation qu'il souhaite prouver. Les trois principaux types sont les suivants, du plus efficace mais très restrictif au moins efficace mais général. On choisira pour chaque équation le type le plus spécifique qui peut lui convenir afin d'avoir une solution la plus efficace possible.

- Les *équations Quadratiques*, notées Q, sont les équations dont toutes les variables (exceptés les générateurs g_1, g_2) sont des scalaires.

  Exemple (1).
  Nous souhaitons prouver que le bit b est égal à 0 ou à 1 sans donner sa valeur. Pour cela nous prévoyons d'engager le bit b une fois dans $\mathbb{G}_1$, nous notons $x = b$ la variable à engager dans $\mathbb{G}_1$, et une fois dans $\mathbb{G}_2$, nous notons $y = b$ la variable à engager dans $\mathbb{G}_2$. Nous prouvons ensuite que les deux équations suivantes sont vérifiées.

  $$\mathsf{e}(\mathsf{g}_1^x, \mathsf{g}_2^{1-y}) = 1_T \quad \text{et} \quad \mathsf{e}(\mathsf{g}_1^{1-x}, \mathsf{g}_2^y) = 1_T \quad \text{avec } 1_T \text{ l'élément neutre de } \mathbb{G}_T \tag{5.1}$$

  Les variables à mettre en gage sont des scalaires : $x = b$ et $y = b$. Nous notons $\vec{x} = x(= b)$ dans $\mathbb{G}_1$ et $\vec{y} = y(= b)$ dans $\mathbb{G}_2$. Nous remarquons que les deux équations (5.1) peuvent aussi s'écrire de la manière suivante.

  $$\mathsf{e}(\mathsf{g}_1^x, \mathsf{g}_2^{1-y}) = \mathsf{e}(\mathsf{g}_1, \mathsf{g}_2)^0 \quad \text{et} \quad \mathsf{e}(\mathsf{g}_1^{1-x}, \mathsf{g}_2^y) = \mathsf{e}(\mathsf{g}_1, \mathsf{g}_2)^0$$

Elles n'utilisent alors que des scalaires en dehors des générateurs g_1 et g_2, il s'agit d'une *équation quadratique.* •

– Les *équations de multiplications Multi-Scalaires*, notées MS, sont les équations qui peuvent être ramenées à une équation dans un seul groupe $\mathbb{G}_1$ ou $\mathbb{G}_2$. C'est-à-dire les équations dont tous les éléments de groupe, secrets ou non, sont soit tous dans $\mathbb{G}_1$ soit tous dans $\mathbb{G}_2$. Ce type d'équation est moins efficace que les équations quadratiques. Ainsi, pour des équations ne contenant que des scalaires, il est préférable de choisir le type Q plutôt que le type MS, bien que les deux soient applicables.

  Exemple (2).
  Nous connaissons un certificat (A, y_1, y_2) tel que $A = \mathsf{g}_1^{\frac{1}{y_1+y_2}}$. Nous souhaitons prouver que notre certificat est correct sans révéler les valeurs y_1 et y_2. Cela revient à prouver que y_1 et y_2 respectent bien l'équation suivante.

$$\mathsf{e}(A, \mathsf{g}_2^{y_1} \cdot g_2^{y_2}) = \mathsf{e}(\mathsf{g}_1, \mathsf{g}_2) \text{ avec } A = \mathsf{g}_1^{\frac{1}{y_1+y_2}} \in \mathbb{G}_1 \tag{5.2}$$

  Les valeurs mises en gage sont les scalaires y_1 et y_2. Nous notons $\vec{y} = (y_1, y_2)$ les scalaires mis en gages dans $\mathbb{G}_2$. L'équation n'utilise qu'un seul élément de groupe autre que les générateurs : $A \in \mathbb{G}_1$. Il s'agit d'une *équation de multiplications Multi-Scalaires* dans $\mathbb{G}_1$.

  •

– Les *équations de Produits de Couplage*, notées PC, représente le type par défaut. Il est moins efficace que les autres, il ne sera donc utilisé que lorsque l'on ne peut pas appliquer un autre type. Cependant il s'agit du cas le plus général qui permet ainsi de répondre aux cas les plus divers.

  Exemple (3).
  Nous connaissons un certificat (A, y_1, Y) tel que $A = \mathsf{g}_1^{\frac{1}{y_1+\gamma}} \in \mathbb{G}_1$ avec $Y = \mathsf{g}_2^{\gamma} \in \mathbb{G}_2$. Nous souhaitons prouver que notre certificat est correct sans révéler les valeurs y_1 et Y ou notre certificat A. Nous souhaitons donc prouver l'équation suivante.

$$\mathsf{e}(A, \mathsf{g}_2^{x})\mathsf{e}(A, Y) = \mathsf{e}(\mathsf{g}_1, \mathsf{g}_2) \text{ avec } A = \mathsf{g}_1^{\frac{1}{x+\gamma}} \in \mathbb{G}_1 \text{ et } Y = \mathsf{g}_2^{\gamma} \in \mathbb{G}_2 \tag{5.3}$$

  L'équation utilise deux éléments de groupe en plus des générateurs (A et Y) n'appartenant pas au même groupe. On ne peut donc appliquer ici qu'une *équation de produits de couplage.* Les valeurs mises en gage sont les éléments de groupe Y et A ainsi que le scalaire y_1. Nous notons $\vec{X} = (A)$, l'élément de groupe mis en gage appartenant à $\mathbb{G}_1$ et $\vec{Y} = (Y, g_2^{y_1})$ le vecteur des éléments engagés dans $\mathbb{G}_2$. On note que, dans le cas PC, les scalaires mis en gages sont inclus comme des éléments de groupe (ici $g_2^{y_1}$). •

Pour chaque type d'équation et chaque problème difficile (SD, SXDH ou DLIN), J.Groth et A.Sahai proposent dans [GS08, GS07] une équation permettant d'obtenir la preuve de connaissance non-interactive Groth-Sahai π_{gs}. Je ne donne pas ici le détail de l'obtention de ces preuves mais seulement les principes de base nécessaires à leur utilisation. Dans un premier temps, le prouveur prépare ses équations.

1. Les parties de l'équation n'interférant pas avec les valeurs engagées sont isolées d'un coté de l'égalité, arbitrairement nous considérons qu'il s'agit du membre de droite. En fonction du type d'équation, nous notons : t la partie scalaire du membre de droite pour les équations

Quadratiques, $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) la partie dans $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) du membre de droite pour les équations Multi-Scalaire dans $\mathbb{G}_1$ (resp. $\mathbb{G}_2$), et enfin t_T la partie dans $\mathbb{G}_T$ dans le cas le plus général des équations de Produit de Couplage.

2. Le prouveur adapte les équations qu'il souhaite prouver aux formats posés par [GS08] en fonction du type d'équation. Il définit ainsi la valeur de la matrice Γ des variables γ_{ij} qui sert de variable d'ajustement pour la mise en forme.

De manière plus formelle, la mise en forme exigée pour chaque type d'équation que nous venons de voir est la suivante.

– *Équations Quadratiques.* Seuls des scalaires sont considérés, nous pouvons donc poser l'équation en termes d'équations d'exposants.

$$\sum_{i=1}^{n} a_i y_i + \sum_{i=1}^{m} x_i b_i + \sum_{j=1}^{n}\sum_{i=1}^{m} \gamma_{ij} x_i y_j = t \tag{5.4}$$

Exemple (1).
Si nous ne considérons que les exposants des deux équations (5.1), nous obtenons.

$$x_1(1 - y_1) = 1 \quad \text{et} \quad y_1(1 - x_1) = 1$$

Nous pouvons ensuite noter t le membre de droite et effectuer la mise en forme.

$$-y_1 + x_1 - x_1 y_1 = t \quad \text{et} \quad y_1 - x_1 - x_1 y_1 = t$$

Nous obtenons ainsi la mise en forme de l'équation (5.4) avec $t = 1$, $a_1 = -1$ et $\gamma_{11} = -1$. •

– *Équations de multiplications Multi-Scalaires.* Pour une équation qui se déroule uniquement dans $\mathbb{G}_1$, c'est-à-dire une équation dont les éléments de groupe sont dans $\mathbb{G}_1$, nous pouvons poser l'équation directement dans $\mathbb{G}_1$.

$$\prod_{i=1}^{n} A_i^{y_i} \prod_{i=1}^{m} X_i^{b_i} \prod_{j=1}^{n}\prod_{i=1}^{m} X_i^{\gamma_{ij} y_j} = \mathcal{T}_1 \tag{5.5}$$

Le cas où l'équation peut se dérouler uniquement dans $\mathbb{G}_2$ se déduit par symétrie.

$$\prod_{i=1}^{n} Y_i^{a_i} \prod_{i=1}^{m} B_i^{x_i} \prod_{j=1}^{n}\prod_{i=1}^{m} Y_j^{\gamma_{ij} x_i} = \mathcal{T}_2 \tag{5.6}$$

Exemple (2).
Notre seul élément de groupe A étant dans $\mathbb{G}_1$, nous utilisons la première version. En nous concentrant sur $\mathbb{G}_1$, nous pouvons résumer l'équation 5.2 par l'équation suivante.

$$A^{y_1} \cdot A^{y_2} = \mathsf{g}_1$$

On peut ensuite noter $\mathcal{T}_1$ le membre de droite (g_1), poser les vecteurs $\vec{A} = (A, A)$ et $\vec{y} = (y_1, y_2)$ et nous obtenons la bonne mise en forme Groth-Sahai. •

– *Équations de Produit de Couplage.* Pour cette équation, nous considérons que les scalaires engagés l'ont été comme des éléments de groupe. Nous pouvons ensuite appliquer la mise en forme suivante.

$$\prod_{i=1}^{n} \mathsf{e}(A_i, Y_i) \cdot \prod_{i=1}^{m} \mathsf{e}(X_i, B_i) \cdot \prod_{j=1}^{n}\prod_{i=1}^{m} \mathsf{e}(X_i, Y_j)^{\gamma_{ij}} = t_T \tag{5.7}$$

Exemple (3).
Il est trivial, une fois les scalaires mis en gages comme des éléments de groupe, de transformer l'équation 5.3. Nous rappelons que dans cet exemple A est secret contrairement à l'exemple 2. Nous avons l'équation suivante.

$$\mathsf{e}(A, Y) \cdot \mathsf{e}(A, \mathsf{g}_2^x) = \mathsf{e}(\mathsf{g}_1, \mathsf{g}_2)$$

On prend à présent la formalisation Groth-Sahai. Le seul élément de groupe de $\mathbb{G}_1$ à mettre en gage est A, on note $\vec{X} = (A)$. Dans $\mathbb{G}_2$ nous avons Y et g_2^x, nous notons $\vec{Y} = (Y, \mathsf{g}_2^x)$. Il reste ensuite à noter $t_T = \mathsf{e}(\mathsf{g}_1, \mathsf{g}_2)$ et nous obtenons l'équation suivante.

$$\mathsf{e}(X_1, Y_1) \cdot \mathsf{e}(X_1, Y_2) = t_T$$ •

Une fois l'équation et les variables formalisées en fonction de leurs types. Le prouveur rentre alors dans une phase calculatoire. Il engage les vecteurs de secrets en fonction du style d'engagement qu'il souhaite (parfaitement **Indistinguable** ou parfaitement **résistant aux collisions**). Puis il applique aux équations mises en formes les équations de preuve $\mathcal{NIPK}$ correspondantes au problème difficile choisit.

Afin de ne pas complexifier inutilement ce mémoire, nous ne donnons pas ici ces équations. Le lecteur pourra les retrouver dans la version complète de l'article original [GS07] ou dans la version revisitée [EGW09]. Pour information, nous donnons le tableau récapitulatif du nombre d'éléments que chaque variable ou équation ajoute à la taille d'une preuve $\mathcal{NIPK}$ Groth-Sahai.

	SD	SXDH	DLIN
Variable - élément de groupe	1	2	3
Variable - scalaire	1	2	3
Produit de couplage	1	8	9
Multiplications Multi-scalaires dans $\mathbb{G}_1$ ou $\mathbb{G}_2$	1	6	9
Équations quadratiques	1	4	6

TABLE 5.1 – Nombre d'éléments de groupe que chaque variable ou équation ajoute à la taille d'une preuve de connaissance non interactive Groth-Sahaï

5.1.3 Procédure de vérification de preuve

La vérification d'une preuve Groth-Sahai correspond à la vérification d'une relation booléenne sur les éléments de la preuve. Cette équation dépend du type d'équation prouvée et du problème difficile choisi. De manière générique, il s'agit d'une relation de la forme :

$$\prod_{i=1}^{k} \mathsf{e}(f_i, h_i)^{c_i} \stackrel{?}{=} A$$

avec $k \in \mathbb{N}$, f_i des éléments de $\mathbb{G}_1$, h_i des éléments de $\mathbb{G}_2$, c_i des scalaires et A un élément de $\mathbb{G}_T$. Le vérifieur retourne vrai si la relation est vérifiée et faux sinon.

La description que nous venons de voir nous permet de mieux comprendre le fonctionnement des preuves de type Groth-Sahai et, en particulier, à quoi correspond la vérification d'une preuve.

Nous allons, à présent, nous intéresser à une méthode permettant de rendre plus efficace cette étape de vérification : le rassemblement de preuves.

5.2 Rassemblement de preuve $\mathcal{NIPK}$ "Groth-Sahai"

La vérification d'une preuve de connaissance non-interactive Groth-Sahai est relativement longue, du fait de nombreuses évaluations de couplages. Nous nous sommes donc intéressés avec O.Blazy, G.Fuchsbauer, M.Izabachène, H.Sibert et D.Vergnaud [BFI+10a, BFI+10b] à une méthode permettant de vérifier un ensemble de sous-preuves de connaissance de même type en une seule fois de manière plus efficace qu'une vérification de chaque sous-preuve une par une. Nous avons ensuite étendu ce résultat à la vérification d'un ensemble de preuves de connaissance non-interactives de même type. La vérification de ces preuves (ou sous-preuves) étant ralentie par l'évaluation de nombreux couplages, nous avons souhaité minimiser ces évaluations d'une part en optimisant les équations de vérification proposées par J.Groth et A.Sahai et d'autre part en utilisant des méthodes de rassemblement d'équation de couplage. Pour cela nous nous sommes intéressés aux travaux de A.L.Ferrara, M.Green, S.Hohenberger et M.O.Pedersen [FGHOP09] décrivant des méthodes permettant de rassembler un ensemble d'équations de couplage tout en gardant un même niveau de sécurité.

Nos travaux permettent ainsi d'obtenir une réponse rapide à la question : "Toutes les preuves (et sous-preuves) de connaissances de l'ensemble sont-elles valides ?" Si la réponse est positive, chaque preuve de connaissance non-interactive de l'ensemble est valide avec une probabilité de faux positifs dépendante des paramètres choisis. Si la réponse est négative alors au moins une preuve est invalide. Dans le cas d'une vérification d'un ensemble de preuves de connaissances, nous devons identifier la (ou les) preuve(s) invalides. Pour cela on peut utiliser une méthode de type "diviser pour conquérir". Dans ce cas, il suffit de diviser l'ensemble des preuves vérifiées en deux sous-ensembles puis de tester chaque sous-ensemble séparément. Si la réponse est positive pour un sous-ensemble, nous avons l'assurance que toutes les preuves de ce sous-ensemble sont valides, sinon nous divisons à nouveau le sous-ensemble en deux et recommençons. Cette méthode se termine lorsque chaque preuve de connaissance invalide a été identifiée.

Notre solution permettant de vérifier un ensemble de preuve de connaissance Groth-Sahai en une seule fois est donc à utiliser, de préférence, dans les cas où l'on soupçonne que toutes, ou au moins l'écrasante majorité, des preuves à vérifier sont valides.

5.2.1 Principe du rassemblement de preuve $\mathcal{NIPK}$ "Groth-Sahai"

Afin de pouvoir vérifier efficacement un ensemble de m relations de vérification, nous devons trouver un moyen de les combiner. Pour ce faire nous utilisons le test du petit exposant proposé par M.Bellare, J.A.Garay et T.Rabin dans [BGR98]. Celui-ci fonctionne de la manière suivante. Supposons que nous ayons m relations de la forme $\mathsf{e}(f_j, h_j)^{c_j} = A_j$. On choisit aléatoirement un ensemble de petits exposants $r_1, \cdots, r_m$ et on vérifie si $\prod_{j=1}^{m} \mathsf{e}(f_j, h_j)^{c_j r_j} = \prod_{j=1}^{m} A_j^{r_j}$. Si c'est le cas les m équations sont vérifiées avec une probabilité de faux positifs $Pr = 2^{-l_r}$ pour l_r la taille des r_i . Sinon au moins une est fausse.

Afin de réduire la complexité de cette technique, nous utilisons astucieusement les trois techniques suivantes.

Déplacer l'exposant dans le couplage. Les exponentiations dans $\mathbb{G}_t$ étant plus couteuses en complexité que dans $\mathbb{G}_1$ (ou dans $\mathbb{G}_2$ pour les couplages qui ne sont pas de type 2),

nous souhaitons les éviter autant que possible. Pour cela, nous pouvons déplacer les exponentiations de $\mathbb{G}_T$ dans $\mathbb{G}_1$. Comme nous travaillons sur des couplages, cette action est réversible si nécessaire.

$$\mathsf{e}(f,h)^c \leftrightarrow \mathsf{e}(f^c,h)$$

Déplacer un produit dans le couplage. Lorsque deux couplages ont un élément commun, ils peuvent être combinés afin de réduire le nombre d'évaluations de couplages.

$$\prod_{j=1}^{m} \mathsf{e}(f_j^{r_j},h) \leftrightarrow \mathsf{e}(\prod_{j=1}^{m} f_j^{r_j},h)$$

Échanger deux produits. Dans certains cas, une amélioration de l'efficacité peut être obtenue en déplaçant un produit d'une composante du couplage à l'autre.

$$\prod_{i=1}^{k} \mathsf{e}(\prod_{j=1}^{m} f_j^{r_{i,j}},h_i) \leftrightarrow \prod_{j=1}^{m} \mathsf{e}(f_j,\prod_{i=1}^{k} h_i^{r_{i,j}})$$

L'application de ces techniques aux relations booléennes de vérification étant particulièrement calculatoire, je ne donnerai ici que l'efficacité des relations booléennes que nous avons obtenues. Le lecteur pourra se référer à l'article paru à ACNS 2010 [BFI+10a], ou à la version longue de l'article [BFI+10b] pour les résultats détaillés.

En ce qui concerne le problème SD nous ne pouvons gagner qu'un facteur n qui ne compense pas l'utilisation du test du petit exposant. Il n'apparaît donc pas dans le tableau récapitulatif. Pour les deux autres problèmes, nous comparons la vérification classique, c'est-à-dire la version sans technique d'assemblage, et notre version.

	Vérification classique	Vérification [BFI+10a, BFI+10b]
SXDH		
Produit de couplage	$5m+3n+16$	$m+2n+8$
Multiplications Multi-scalaires dans $\mathbb{G}_1$	$8m+2n+14$	$\min(2n+9, 2m+n+7)$
Multiplications Multi-scalaires dans $\mathbb{G}_2$	$8n+2m+14$	$\min(2m+9, 2n+m+7)$
Équations quadratiques	$8m+8n+12$	$2\min(m,n)+8$
DLIN		
Produit de couplage	$12n+27$	$3n+6$
Multiplications Multi-scalaires	$9n+12m+27$	$3n+3m+6$
Équations quadratiques	$18n+24$	$3n+6$

TABLE 5.2 – Nombre d'évaluations de couplages pour une vérification où n et m représentent les nombres d'élément de groupe mis en gage dans $\mathbb{G}_1$ et dans $\mathbb{G}_2$.

5.2.2 Application aux signatures de groupe de J.Groth

Nous allons maintenant voir l'impact de ces techniques sur les signatures de groupe de J.Groth [Gro07], c'est-à-dire sur une des signatures de groupe les plus efficaces dans le modèle standard. J.Groth propose une méthode transformant une signature certifiée [BFPW07]

qui respecte une certaine structure en une signature de groupe en utilisant les preuves non-interactives à témoin indistinguable (cf. Définition 36) de [GS08, GS07]. Pour les détails sur l'obtention de ces résultats, le lecteur pourra se référer à l'article paru à la conférence ACNS 2010 [BFI$^+$10a], ou à la version longue de l'article [BFI$^+$10b].

Le schéma de signature de groupe de J.Groth fonctionne de la façon suivante.

- Un utilisateur choisit une paire de clés pour le schéma de signature certifiée et demande au gestionnaire de groupe de certifier sa clé de vérification.
- Pour générer une signature de groupe, l'utilisateur crée une signature certifiée, la chiffre et construit une preuve de connaissance à témoin indistinguable montrant que le chiffré contient une signature certifiée valide.

Afin de simplifier l'illustration de notre amélioration nous ne donnons pas de description formelle du schéma mais un simple aperçu. La version que nous allons utiliser comme exemple est la solution **CPA**-anonyme de [Gro07].

Dans cette version, la clé publique du gestionnaire de groupe GG est le triplet $\mathsf{gpk} = (f, h, T) \in \mathbb{G}^2 \times \mathbb{G}_T$, sa clé privée étant $\mathsf{ggsk} \in \mathbb{G}$ avec $\mathsf{e}(f, \mathsf{ggsk}) = T$. Les utilisateurs sont munis d'une paire de clés $(\mathsf{psk}, \mathsf{ppk})$ telle que $\mathsf{ppk} = g^{\mathsf{psk}} \in \mathbb{G}$. Le certificat A d'appartenance au groupe d'un membre ayant la clé publique $\mathsf{ppk} = g^{\mathsf{psk}} \in \mathbb{G}$ est un couple $A = (A_1, A_2)$ tel que $\mathsf{e}(A_1, \mathsf{ppk} \cdot h)\,\mathsf{e}(f, A_2) = T$.

Pour signer un message $m \in \mathbb{Z}_p$, l'utilisateur calcule une signature faible Boneh-Boyen [BB08] $\sigma = g^{1/(\mathsf{psk}+m)}$ en utilisant sa clé privée psk. Il forme ensuite les mises en gages des éléments de groupes ppk, A_2 et σ que nous notons b_{ppk}, $b_{\mathcal{A}_2}$ et b_σ. Il lui suffit ensuite de générer la preuve que les engagements satisfont les équations suivantes.

$$\mathsf{e}(A_1, \mathsf{ppk} \cdot h)\,\mathsf{e}(f, A_2) = T \qquad\qquad \mathsf{e}(\sigma, g^m \mathsf{ppk}) = \mathsf{e}(g, g) \tag{5.8}$$

Le fait que A_1 soit donné en clair n'est pas problématique du fait de la malléabilité du certificat. En effet, cela permet aux membres du groupe de renouveler l'aléa à chaque nouvelle signature. Si on note $\pi_{GS} = (\phi_1, \phi_2)$ la preuve Groth-Sahai correspondant aux deux équations (5.8), alors la signature de groupe obtenue est :

$$\Sigma = (A_1, b_{\mathsf{ppk}}, b_{\mathcal{A}_2}, b_\sigma, \pi_{GS})$$

En appliquant nos techniques d'assemblages présentées plus haut aux équations de vérifications (5.8) et en prenant en compte le fait que la première équation est un cas simplifié des produits de couplages, nous obtenons les résultats résumés en Table 5.2.

	Version classique [Gro07]	Version avec [BFI$^+$10a, BFI$^+$10b]
Vérification de la sous-preuve ϕ_1	13	–
Vérification de la sous-preuve ϕ_2	55	–
Vérification de la preuve $\pi_{GS} = (\phi_1, \phi_2)$	68	11
Vérification de n preuves	$68n$	$4n + 7$

TABLE 5.3 – Efficacité de notre solution par rapport à la version classique

La version classique vérifie chacune des sous-preuves séparément, le coût de vérification d'une preuve est donc la somme des coûts de vérification de chaque sous-preuve. Notre solution, pour sa part, vérifie les deux sous-preuves en même temps, il n'est donc pas nécessaire d'indiquer de coûts de vérifications pour les sous-preuves.

Conclusion

Nous avons étudié dans ce chapitre le fonctionnement des preuves de connaissances non-interactives Groth-Sahai et comment améliorer l'efficacité de la vérification de telles preuves. Dans cet objectif, j'ai décrit les travaux auxquels j'ai participé avec O.Blazy, G.Fuchsbauer, M.Izabachène, H.Sibert et D.Vergnaud, sur les techniques de rassemblement de telles preuves. Cette étude a permis de rendre plus efficace leur vérification, en particulier lorsqu'un vérifieur a de nombreuses preuves à vérifier. Nos résultats, publiés à la conférence ACNS 2010 [BFI+10a], permettent de diviser jusqu'à 10 fois le nombre d'évaluations de couplages, l'opération dominante, nécessaires à une vérification.

Troisième partie

Signatures Caméléons

Chapitre 6

Signatures caméléons

Sommaire

Les signatures caméléons ont été introduites par H.Krawczyk et T.Rabin dans [KR00]. L'idée est de permettre à un signataire de choisir un délégué qui sera capable de modifier certaines parties des messages signés tout en gardant une signature valide du signataire sur le message modifié. Les schémas de signature caméléon sont désignés en anglais sous le terme de *Chameleon Signature Schemes* ou *Sanitizable Signature Schemes*. Il est ici important de préciser que le second terme regroupe non seulement les signatures caméléons auxquelles nous allons nous intéresser mais aussi un autre type de signature dans lequel le délégué n'est capable que de supprimer certaines parties du message et non de le modifier. Ce second type de schéma est aussi connu sous le nom de *Content Extraction Signatures* [SBZ01] et de *Redactable Signature Schemes* [CLX09].

Dans ce chapitre, nous commencerons par présenter le modèle de sécurité défini par C.Brzuska *et al.* dans [BFF+09] pour les signatures caméléons, nous nous intéresserons ensuite à la sécurité des constructions de l'état de l'art. Enfin, nous décrirons la nouvelle signature caméléon que nous avons présentée à la conférence CT-RSA 2010 avec S. Canard [CJ10a].

6.1 Définitions et état de l'art

6.1.1 Définitions

Un schéma de signature caméléon est un schéma de signature permettant au signataire d'un message de désigner un délégué qui pourra par la suite modifier certaines parties du message signé tout en gardant une signature valide du signataire sur le nouveau message.

Définition 40 (Message/Signature original(e)).
On appelle message original un message dont la signature a été émise par le signataire. De même on appelle signature originale une signature caméléon émise par le signataire. ∗

Définition 41 (Message/Signature modifié(e)).
On appelle message modifié un message dont la signature a été obtenue par modification d'une signature originale. On appelle signature modifiée une signature caméléon émise par le délégué par modification d'une signature originale. ∗

Définition 42 (Instance).
Une instance de signature caméléon est l'ensemble des paires message-signature comprenant une paire originale ainsi que toutes les paires modifiées issues de celle-ci. Deux messages sont dits de la même instance s'ils ont en commun le même message original ou si l'un des deux messages est l'original dont l'autre est une modification. ∗

6.1.2 État de l'art

L'idée à l'initiative des signatures caméléons provient de l'article [KR00] de H.Krawczyk et T.Rabin. Ils y proposent, comme application de leur fonction de hachage caméléon (voir Définition 22), de créer une signature permettant à un tiers de confiance (le délégué) de modifier tout le message signé tout en gardant une signature valide du signataire. X.Chen, F.Zhang et K.Kim proposent dans [CZK04] un nouveau schéma suivant la même idée. Mais ce n'est qu'en 2005 que le premier schéma de signature caméléon, selon la définition actuelle, est proposé par G.Ateniese, D.H.Chou, B.De Medeiros et G.Tsudik [ACdT05]. Ils permettent ainsi au signataire de choisir les parties modifiables du message qu'il signe et esquissent les notions de sécurité nécessaires à un tel schéma.

Cet article a motivé le premier schéma non falsifiable de signature caméléon par S.Canard, F.Laguillaumie et M.Milhau [CLM08] ainsi que le modèle proposé par C.Brzuska, M.Fischlin, T.Freudenreich, A.Lehmann, M.Page, J.Schelbert, D.Schröder et F.Volk dans [BFF+09], que nous prendrons comme référence.

Je vais à présent décrire le modèle [BFF+09]. La description détaillée des différents schémas que nous venons de voir ainsi que leur sécurité dans ce modèle est donnée en Section 6.3.

6.2 Modèle de Brzuska *et al.*

Dans [ACdT05], Ateniese *et al.* proposent le premier jet d'un modèle de sécurité pour ce type de schéma. C'est à partir de cette esquisse que Brzuska *et al.* [BFF+09] ont construit le modèle que nous allons maintenant présenter.
Nous décrirons d'abord la définition formelle d'un schéma de signature caméléon dans ce modèle avant de nous intéresser aux propriétés de sécurité qu'il exige et enfin nous comparerons la sécurité des différents algorithmes de l'état de l'art dans ce modèle.

Le signataire souhaitant effectuer une signature caméléon sur un message m doit découper ce message en parties afin de séparer les parties modifiables par le délégué des parties qui devront rester inchangées. La description initiale du modèle par Brzuska *et al.* considère que le signataire découpe son message en parties égales. Cependant une telle mise en forme implique un grand nombre de parties dont la taille sera, dans le meilleur des cas, égale à la plus petite chaîne de caractères consécutifs de même type (tous modifiables ou tous fixes). J'ai choisi de modifier le modèle en permettant aux parties d'avoir des tailles différentes. Nous verrons par la suite que cette modification n'a qu'un impact mineur sur le reste du modèle. Dans la suite, la taille du message m en bits sera notée ℓ, le nombre de parties du message sera noté t, et enfin la taille de la $i$ème partie m_i du message sera notées ℓ_i.

6.2.1 Définition formelle d'un schéma de signature caméléon

Avant de donner une définition formelle d'un schéma de signature caméléon dans ce modèle, nous devons définir deux variables :

Définition 43 (ADM).
La variable ADM définit les modifications autorisées (admissible) sur un message. Elle est composée des indices des parties modifiables par le délégué ainsi que de la taille en bits de chaque partie du message.

$$\mathsf{ADM} = \left(\{i | \mathsf{m}_i \text{ est modifiable }\}, \{\ell_i\}_{i\in[1,\mathsf{t}]}\right) \qquad *$$

Par abus de langage, on notera $|\mathsf{ADM}|$ le nombre de parties indiquées comme modifiables par le signataire.

Définition 44 (MOD).
La variable MOD définit les modifications que le délégué souhaite faire sur un message donné. Plus formellement, MOD est l'ensemble des paires (i, m'_i) telles que m_i sera remplacé par m'_i durant $\mathcal{SC}.\textsc{Modifie}$.

$$\mathsf{MOD} = \{(i, \mathsf{m}'_i) | \mathsf{m}_i \text{ sera remplacé par } \mathsf{m}'_i \text{ durant } \mathcal{SC}.\textsc{Modifie}\} \qquad *$$

Par abus de langage, nous dirons que $i \in \mathsf{MOD}$ s'il existe m'_i tel que $(i, \mathsf{m}'_i) \in \mathsf{MOD}$. Pour finir nous dirons que MOD *correspond à* ADM si $\forall i \in \mathsf{MOD}, i \in \mathsf{ADM}$.

Définition 45 (Signature Caméléon).
Un schéma de signature caméléon $\mathcal{SC}$ est un schéma de signature défini par les sept algorithmes suivants.

- $\mathcal{SC}.\textsc{Init}$ prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système comprenant la description du groupe et les paramètres d'initialisations des schémas utilisés. On notera ces paramètres $\mathcal{SC}.\mathsf{param}$. Nous considérerons dans la suite que λ est inclus dans les paramètres.

$$\mathcal{SC}.\mathsf{param} \leftarrow \mathcal{SC}.\textsc{Init}(1^\lambda)$$

- $\mathcal{SC}.\textsc{SigGénClé}$ (respectivement $\mathcal{SC}.\textsc{SanGénClé}$) génère la paire de clés du signataire $(\mathsf{spk}, \mathsf{ssk})$ (respectivement du délégué $(\mathsf{dpk}, \mathsf{dsk})$) en fonction des paramètres du système $\mathcal{SC}.\mathsf{param}$.

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SC}.\textsc{SigGénClé}(\mathcal{SC}.\mathsf{param}) \text{ et } (\mathsf{dpk}, \mathsf{dsk}) \leftarrow \mathcal{SC}.\textsc{SanGénClé}(\mathcal{SC}.\mathsf{param})$$

– $\mathcal{SC}$.Signe est l'algorithme de signature. Il prend en entrée un message m de taille ℓ qui est composé de t parties, la clé secrète du signataire ssk et la clé publique du délégué dpk. Enfin, il prend en entrée une variable ADM qui permet au signataire d'indiquer les parties qui pourront être modifiées par le délégué ainsi que la taille des différentes parties du message (cf. Définition 43). L'algorithme retourne une signature Σ sur le message m (ou $\perp$ en cas d'erreur). Nous considérons dans la suite que la variable ADM est incluse dans la signature Σ.

$$\Sigma \leftarrow \mathcal{SC}.\textsc{Signe}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM})$$

– $\mathcal{SC}$.Modifie utilise un message m et sa signature caméléon Σ, la clé publique du signataire spk, la clé secrète du délégué dsk ainsi qu'une variable MOD décrivant les modifications voulues sur le message m (cf. Définition 44). L'algorithme retourne une nouvelle signature Σ' sur le message modifié m' (ou $\perp$ en cas d'erreur).

$$(\Sigma', \mathsf{m}') \leftarrow \mathcal{SC}.\textsc{Modifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD})$$

– $\mathcal{SC}$.Vérifie permet de s'assurer de la validité d'une signature Σ sur un message m en utilisant les clés publiques du signataire spk et du délégué dpk. Il retourne vrai si la signature est valide et faux si elle ne l'est pas.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SC}.\textsc{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk})$$

– $\mathcal{SC}$.Prouve prend en entrées une signature Σ sur un message m, la clé secrète du signataire ssk, la clé publique du délégué dpk ainsi qu'un ensemble de couples message-signature ayant été produit avec ces clés $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k=[1,\mathsf{q}]}$. Il retourne une preuve $\pi_{\mathsf{or/mod}}$ sur la signature Σ du message m.

$$\pi_{\mathsf{or/mod}} \leftarrow \mathcal{SC}.\textsc{Prouve}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, (\mathsf{m}_k, \Sigma_k)_{k\in[1,\mathsf{q}]})$$

– $\mathcal{SC}$.Juge est un algorithme public permettant de vérifier, à l'aide d'une preuve générée par le signataire, si une paire message-signature est originale ou non. Pour cela, il prend en entrée la paire concernée (m, Σ), la preuve $\pi_{\mathsf{or/mod}}$ provenant de l'algorithme $\mathcal{SC}$.Prouve ainsi que les clés publiques du signataire spk et du délégué dpk. L'algorithme retourne l'origine du couple (m, Σ), c'est à dire $\mathsf{Signataire}$ si le couple est un original ou $\mathsf{Délégué}$ s'il s'agit d'une paire modifiée.

$$\{\mathsf{Signataire}, \mathsf{Délégué}\} \leftarrow \mathcal{SC}.\textsc{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$$

*

Dans ce modèle la consistance d'un schéma de signature caméléon correspond aux trois sous-propriétés suivantes.

– **La consistance par rapport au signataire** : Si une signature sur un message à été émise par $\mathcal{SC}$.Signe en utilisant une clé secrète générée par $\mathcal{SC}$.SigGénClé, alors elle doit être acceptée avec une probabilité écrasante par l'algorithme de vérification $\mathcal{SC}$.Vérifie.
– **La consistance par rapport au délégué** : Si une signature sur un message a été générée honnêtement par $\mathcal{SC}$.Modifie à partir d'une signature valide et d'une clé secrète de délégué générée par $\mathcal{SC}$.SanGénClé, alors elle doit être acceptée avec une probabilité écrasante par l'algorithme de vérification $\mathcal{SC}$.Vérifie.
– **La consistance de la preuve** : Pour n'importe quel message modifié, le signataire doit être capable d'exhiber une preuve $\pi_{\mathsf{or/mod}}$ grâce à l'algorithme $\mathcal{SC}$.Prouve telle que l'algorithme $\mathcal{SC}$.Juge retourne $\mathsf{Délégué}$.

6.2.2 Propriétés de Sécurité

Brzuska *et al.* ont défini cinq propriétés de sécurité requises pour un schéma de signature caméléon. De manière informelle ces propriétés sont les suivantes :

- **Infalsifiabilité**. Il doit être impossible de produire, pour une personne autre que le signataire et le délégué, une signature originale ou une signature modifiée. Cette propriété est similaire à la propriété d'**Infalsifiabilité** d'un schéma de signature classique.
- **Immuabilité**. Il doit être impossible, même pour le délégué, de modifier une partie du message désignée comme non-modifiable par le signataire, tout en gardant une signature caméléon valide de celui-ci.
- **Intimité**. Personne ne doit être capable de restaurer le message original à partir de messages modifiés.
- **Transparence**. Seuls le signataire et le délégué doivent pouvoir distinguer une paire message-signature originale d'une paire modifiée.
- **Responsabilité**. En cas de problème sur l'origine d'une signature, le Juge doit être capable d'arbitrer correctement qui est à l'origine de la signature. Plus formellement, cette propriété ce décompose en deux sous-propriétés.
  - Délégué-**Responsabilité** : Si un message n'a pas été signé par le signataire, alors même un délégué corrompu doit être incapable de faire accuser le signataire par le Juge.
  - Signataire-**Responsabilité** : Si un message et sa signature n'ont pas été modifiés par le délégué, alors même un signataire corrompu ne doit pas pouvoir faire accuser le délégué par le Juge.

Une grande similitude existe ici avec les propriétés esquissées par Ateniese *et al.* Les deux principales modifications résident dans l'ajout de la propriété d'**Infalsifiabilité** et dans la modification de la **Responsabilité**. Ateniese *et al.* ne considérant qu'une Faible-**Responsabilité** qui correspond à la Signataire-**Responsabilité**.

Dans [BFF+09], Brzuska *et al.* montrent que la **Transparence** implique l'**Intimité** et que la **Responsabilité** implique l'**Infalsifiabilité**. Nous ne donnerons ici que l'idée de ces preuves. Le lecteur trouvera les preuves détaillées ainsi que les preuves d'indépendance des trois propriétés restantes (**Transparence**, **Immuabilité** et **Responsabilité**) dans l'article original [BFF+09]. Les relations entre les différentes propriétés sont résumées en figure 6.1.

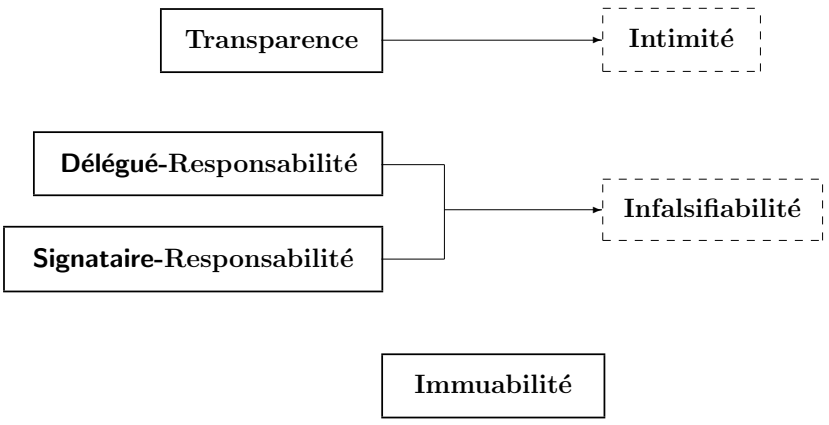

FIGURE 6.1 – Simplification des propriétés

Intuition des implications :

- **Transparence** ⇒ **Intimité**. Si un adversaire contre l'**Intimité** existe, il est capable de retrouver le message original à partir de messages modifiés. En conséquence celui-ci est capable de distinguer le message original d'un message modifié. Et donc il sera possible d'utiliser cet adversaire pour construire un adversaire contre la **Transparence**.
- **Responsabilité** ⇒ **Infalsifiabilité**. La **Responsabilité** est composée de deux sous propriétés la Signataire-**Responsabilité** et la Délégué-**Responsabilité**. En conséquence nous avons l'implication suivante :

$$(\text{Signataire} - \textbf{Responsabilité} \text{ et } \text{Délégué} - \textbf{Responsabilité}) \Rightarrow \textbf{Infalsifiabilité}.$$

  Le principe est alors de supposer un adversaire contre l'**Infalsifiabilité** et de prouver que celui-ci peut être utilisé pour construire un adversaire contre la Signataire-**Responsabilité** ou la Délégué-**Responsabilité**. Or s'il existe un adversaire contre l'**Infalsifiabilité**, alors celui-ci est capable de construire une signature originale ou une signature modifiée. Dans le premier cas la signature originale peut-être utilisée pour gagner l'expérience contre la Signataire-**Responsabilité** dans le second cas la signature modifiée pourra être utilisée pour gagner l'expérience contre la Délégué-**Responsabilité**.

Pour rappel nous avons légèrement modifié le modèle de Brzuska *et al.* en permettant aux parties du message d'avoir des tailles différentes. On remarque ici que dans les deux implications le fait que les différentes parties soient toutes de même taille ou non n'a aucun impact. Nous utiliserons donc ces implications dans la suite et ne prouverons que trois propriétés de sécurité : **Transparence**, **Immuabilité** et **Responsabilité**. En conséquence, nous nous intéressons seulement aux définitions formelles des trois propriétés restantes.

Les Oracles

Les définitions formelles de propriétés de sécurité définissent un adversaire polynomial $\mathcal{A}$ qui cherche à gagner une expérience contre un challengeur $\mathcal{C}$. Dans les expériences que nous allons voir l'adversaire pourra utiliser certains des quatre oracles suivants.

- L'oracle $\mathcal{O}$.Signe permet à l'adversaire d'obtenir une signature originale du signataire sur un message m de son choix pour la clé secrète qui correspondant à la clé publique de délégué dpk et la variable ADM que l'adversaire a définie.

$$\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$$

- Réciproquement, l'oracle $\mathcal{O}$.Modifie permet à l'adversaire de faire modifier une paire message-signature (m, Σ) de son choix par le délégué en lui précisant les modifications qu'il souhaite MOD. L'oracle retourne une signature modifiée valide si les modifications sont admissibles et $\perp$ en cas d'erreur.

$$\mathcal{O}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD})$$

- $\mathcal{O}$.Prouve est l'oracle correspondant à l'algorithme de preuve d'origine. Il permet à l'adversaire d'obtenir une preuve de l'origine d'une paire message-signature (m, Σ) de son choix en fonction d'une base de données $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k \in [1,\mathsf{q}]}$ qu'il fournit.

$$\mathcal{O}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{BD})$$

– Enfin l'oracle $\mathcal{O}.\text{MODIF/SIGNE}_b$ prend en entrée un message m, ADM et MOD. L'oracle retourne $\perp$ si MOD ne correspond pas à ADM. On note m' le message correspondant au message m modifié selon MOD. L'oracle retourne une signature originale du signataire sur m' si $b = 0$. Et si $b = 1$, il effectue une signature originale du signataire sur le message m selon ADM puis modifie celle-ci en fonction de MOD comme le ferait le délégué. Enfin, il retourne cette signature modifiée sur le message m'.

$$\mathcal{O}.\text{MODIF/SIGNE}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$$

Pour des raisons de lisibilité des expériences, les entrées choisies par l'adversaire seront omises dans les oracles, de même le $\mathcal{O}.$ sera sous-entendu lorsqu'il est évident. Par exemple, l'oracle $\mathcal{O}.\text{SIGNE}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$ sera abusivement noté SIGNE dans les expériences.

Définitions des propriétés de sécurité

Nous définissons à présent de manière formelle les propriétés de sécurité pour un schéma de signature caméléon sûr.

Définition 46 (Immuabilité).
Soient λ un paramètre de sécurité, $\mathcal{SC}.\text{INIT}(1^\lambda)$ un algorithme d'initialisation et $\mathcal{SC}.\text{SIGGÉNCLÉ}$ un algorithme de génération de clé de signataire $(\mathsf{spk}, \mathsf{ssk})$. Soit $\mathcal{A}$ un attaquant ayant accès à l'oracle de signature $\mathcal{O}.\text{SIGNE}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$ et à l'oracle de preuve $\mathcal{O}.\text{PROUVE}(\mathsf{m}, \Sigma, \mathsf{BD})$ décris plus haut. Il renvoie un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$. Nous définissons l'expérience contre l'**Immuabilité** comme suit.

$\text{EXP}^{\mathcal{SC}}_{\mathbf{Imm},\mathcal{A}}(\lambda)$:
$\mathcal{SC}.\mathsf{param} \leftarrow \mathcal{SC}.\text{INIT}(1^\lambda)$
$(\mathsf{spk}, \mathsf{ssk}) \longleftarrow \mathcal{SC}.\text{SIGGÉNCLÉ}(\mathcal{SC}.\mathsf{param})$
$(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*) \longleftarrow \mathcal{A}^{\text{SIGNE,PROUVE}}(\mathcal{SC}.\mathsf{param}, \mathsf{spk})$
Soient $\{(\mathsf{m}_k, \mathsf{ADM}_k, \mathsf{dpk}_k)\}_{k\in[1,\mathsf{n}]}$ les requêtes à l'oracle SIGNE et Σ_k les réponses correspondantes.
Retourne 1 si $\mathcal{SC}.\text{VÉRIFIE}(\mathcal{SC}.\mathsf{param}, \mathsf{m}^*, \Sigma^*, \mathsf{spk}, \mathsf{dpk}^*) = \mathsf{vrai}$ et,
pour tout $k \in [1, \mathsf{n}]$,
$\mathsf{dpk}^* \neq \mathsf{dpk}_i$ ou
$\exists j \notin \mathsf{ADM}_k$ tel que $\mathsf{m}^*_j \neq \mathsf{m}_{k_j}$
sinon retourne 0.

Le succès de l'adversaire polynomial $\mathcal{A}$ dans l'expérience $\text{EXP}^{\mathcal{SC}}_{\mathbf{Imm},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathbf{Imm},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{EXP}^{\mathcal{SC}}_{\mathbf{Imm},\mathcal{A}}(\lambda)].$$

Un schéma de signature caméléon $\mathcal{SC}$ est dit **Immuable** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SC}}_{\mathbf{Imm},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathbf{Imm},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

Définition 47 (Transparence).
Soient λ un paramètre de sécurité, $\mathcal{SC}.\textsc{Init}(1^\lambda)$ un algorithme d'initialisation, $\mathcal{SC}.\textsc{SigGénClé}$ un algorithme de génération de clé de signataire $(\mathsf{spk}, \mathsf{ssk})$, $\mathcal{SC}.\textsc{SanGénClé}$ un algorithme de génération de clés de délégué $(\mathsf{dpk}, \mathsf{dsk})$ et b un bit choisit aléatoirement. Soit $\mathcal{A}$ un attaquant ayant accès aux quatre oracles que nous avons présentés : l'oracle de signature $\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$, l'oracle de modification $\mathcal{O}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD})$, l'oracle de preuve $\mathcal{O}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{BD})$ et enfin l'oracle de challenge $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$. Ce dernier retourne une signature originale si $b = 0$ ou une signature modifiée si $b = 1$. Notons que l'oracle preuve ne peut être utilisé sur les signatures demandées à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b$. À la fin de l'expérience, l'attaquant retourne un bit b'. Nous définissons l'expérience contre la **Transparence** comme suit.

$\textsc{Exp}^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda)$:
$\mathcal{SC}.\mathsf{param} \leftarrow \mathcal{SC}.\textsc{Init}(1^\lambda)$
$(\mathsf{spk}, \mathsf{ssk}) \longleftarrow \mathcal{SC}.\textsc{SigGénClé}(\mathcal{SC}.\mathsf{param})$
$(\mathsf{dpk}, \mathsf{dsk}) \longleftarrow \mathcal{SC}.\textsc{SanGénClé}(\mathcal{SC}.\mathsf{param})$
$b \longleftarrow \{0, 1\}$
$b^* \longleftarrow \mathcal{A}^{\textsc{Signe},\textsc{Modifie},\textsc{Prouve},\textsc{Modif}/\textsc{Signe}_b}(\mathcal{SC}.\mathsf{param}, \mathsf{spk}, \mathsf{dpk})$
Retourne 1 si $b^* = b$ sinon retourne 0.

L'avantage de l'adversaire $\mathcal{A}$ dans l'expérience $\textsc{Exp}^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Adv}^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda) = |Pr[1 \longleftarrow \textsc{Exp}^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda)] - 1/2|.$$

Un schéma de signature caméléon $\mathcal{SC}$ est dit **Transparent** si quelque soit l'adversaire polynomial $\mathcal{A}$ son avantage $\mathsf{Adv}^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Adv}^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

Définition 48 (Délégué-Responsabilité).
Soient λ un paramètre de sécurité, $\mathcal{SC}.\textsc{Init}(1^\lambda)$ un algorithme d'initialisation et $\mathcal{SC}.\textsc{SigGénClé}$ un algorithme de génération de clé de signataire $(\mathsf{spk}, \mathsf{ssk})$. Soit $\mathcal{A}$ un attaquant ayant accès à l'oracle de signature $\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$ et à l'oracle de preuve $\mathcal{O}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{BD})$ que nous avons défini. Il renvoie un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$. Nous définissons l'expérience contre la Délégué-**Responsabilité** comme suit.

$\text{Exp}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda)$:

$\mathcal{SC}.\mathsf{param} \leftarrow \mathcal{SC}.\text{Init}(1^\lambda)$

$(\mathsf{spk}, \mathsf{ssk}) \longleftarrow \mathcal{SC}.\text{SigGénClé}(\mathcal{SC}.\mathsf{param})$

$(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*) \longleftarrow \mathcal{A}^{\text{Signe,Prouve}}(\mathcal{SC}.\mathsf{param}, \mathsf{spk})$

Soit $(\mathsf{m}_k, \mathsf{ADM}_k, \mathsf{dpk}_k)_{k\in[1,\mathsf{n}]}$ les requêtes à l'oracle $\mathcal{SC}.\text{Signe}$ et Σ_i les réponses correspondantes.

$\pi_{\mathsf{or/mod}} \longleftarrow \mathcal{SC}.\text{Prouve}(\mathcal{SC}.\mathsf{param}, \mathsf{ssk}, \mathsf{m}^*, \Sigma^*, \mathsf{BD}, \mathsf{dpk}^*)$ avec $\mathsf{BD} = (\mathsf{m}_k, \Sigma_k)_{k\in[1,\mathsf{n}]}$

Retourne 1 si :

$\mathcal{SC}.\text{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}^*, \Sigma^*, \mathsf{spk}, \mathsf{dpk}^*) = \mathsf{vrai}$;

$(\mathsf{dpk}^*, \mathsf{m}^*) \neq (\mathsf{dpk}_k, \mathsf{m}_k)$ pour tout $k \in [1, \mathsf{n}]$;

et $\mathcal{SC}.\text{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}^*, \Sigma^*, \mathsf{spk}, \mathsf{dpk}^*, \pi_{\mathsf{or/mod}}) = \mathsf{Signataire}$.

Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda)].$$

Un schéma de signature caméléon $\mathcal{SC}$ est dit Délégué- **Responsable** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

Définition 49 (Signataire-Responsabilité).
Soient λ un paramètre de sécurité, $\mathcal{SC}.\text{Init}(1^\lambda)$ un algorithme d'initialisation et $\mathcal{SC}.\text{SanGénClé}$ un algorithme de génération de clé de délégué $(\mathsf{dpk}, \mathsf{dsk})$. Soit $\mathcal{A}$ un attaquant ayant accès à l'oracle de modification $\mathcal{O}.\text{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD})$ et qui renvoie en fin d'expérience un quadruplet $(\mathsf{spk}^*, \pi^*_{\mathsf{or/mod}}, \mathsf{m}^*, \Sigma^*)$. Nous définissons l'expérience contre la Signataire-**Responsabilité** comme suit.

$\text{Exp}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp},\mathcal{A}}(\lambda)$:

$\mathcal{SC}.\mathsf{param} \leftarrow \mathcal{SC}.\text{Init}(1^\lambda)$

$(\mathsf{dpk}, \mathsf{dsk}) \longleftarrow \mathcal{SC}.\text{SanGénClé}(\mathcal{SC}.\mathsf{param})$

$(\mathsf{spk}^*, \pi^*_{\mathsf{or/mod}}, \mathsf{m}^*\Sigma^*) \longleftarrow \mathcal{A}^{\text{Modifie}}(\mathcal{SC}.\mathsf{param}, \mathsf{dpk})$

Soit $(\mathsf{m}'_k, \Sigma'_k)_{k\in[1,\mathsf{n}]}$ les réponses de l'oracle Modifie.

Retourne 1 si :

$\mathcal{SC}.\text{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}^*, \Sigma^*, \mathsf{spk}^*, \mathsf{dpk}) = \mathsf{vrai}$;

$(\mathsf{spk}^*, \mathsf{m}^*) \neq (\mathsf{spk}_k, \mathsf{m}'_k)$ pour tout $k \in [1, \mathsf{n}]$;

$\mathcal{SC}.\text{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}^*, \Sigma^*, \mathsf{spk}^*, \mathsf{dpk}, \pi^*_{\mathsf{or/mod}}) = \mathsf{Délégué}$.

Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SC}}_{\mathsf{sig-Resp},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathsf{sig-Resp},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{SC}}_{\mathsf{sig-Resp},\mathcal{A}}(\lambda)].$$

Un schéma de signature caméléon $\mathcal{SC}$ est dit Signataire- **Responsable** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SC}}_{\mathsf{sig-Resp},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathsf{sig-Resp},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

6.3 La sécurité des constructions existantes

Nous nous plaçons à présent dans le modèle [BFF+09] et nous nous intéressons à la sécurité des cinq schémas de signature caméléon de la littérature.

6.3.1 Solution de H.Krawczyk et T.Rabin

À NDSS 2000, H.Krawczyk et T.Rabin s'intéressent aux fonctions de hachage caméléons (cf. Section 22). A cette occasion ils proposent, comme application de leur fonction, un nouveau concept : une signature sur un message qui sera modifiable par un tiers de confiance. Ce proto-schéma de signature caméléon considère que tout le message est modifiable par le tiers désigné (le délégué). Le schéma proposé utilise un schéma de signature **EU − CMA** sûr, noté $\mathcal{S}$ (cf. Section 29) ainsi que le schéma de fonction de hachage caméléon décrit dans l'article que nous noterons $\mathcal{HC}$. Le schéma de signature caméléon fonctionne de la manière suivante.

Construction 16.
Signature Caméléon de H.Krawczyk et T.Rabin

$\mathcal{SC}$.Init(1^λ) Soit $\mathbb{G}$ un groupe d'ordre p premier, g un générateur du groupe choisi en fonction du paramètre de sécurité λ. D'autre part, le schéma de signature est initialisé $\mathcal{S}.\mathsf{param} = \mathcal{S}.\text{Init}(\lambda)$.

$$\mathcal{SC}.\mathsf{param} = \{\mathbb{G}, g, p, \mathcal{S}.\mathsf{param}\} \leftarrow \mathcal{SC}.\text{Init}(1^\lambda)$$

$\mathcal{SC}$.SigGénClé($\mathcal{SC}.\mathsf{param}$) La paire de clés du signataire correspond à une paire de clés du schéma de signature $\mathcal{S}$. Il génère donc sa paire de clés en utilisant l'algorithme dédié du schéma $\mathcal{S}$: $(\mathsf{ssk}, \mathsf{spk}) = \mathcal{S}.\text{GénClé}(\mathcal{S}.\mathsf{param})$.

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SC}.\text{SigGénClé}(\mathcal{SC}.\mathsf{param}).$$

$\mathcal{SC}$.SanGénClé($\mathcal{SC}.\mathsf{param}$) La clé secrète du délégué dsk est choisie aléatoirement dans $\{0,1\}^\lambda$. Le délégué calcule ensuite sa clé publique $\mathsf{dpk} = g^{\mathsf{dsk}}$. Il obtient la paire $(\mathsf{dsk}, \mathsf{dpk})$ qui correspond à une paire de clés de la fonction de hachage caméléon de [KR00].

$$(\mathsf{dpk}, \mathsf{dsk}) \leftarrow \mathcal{SC}.\text{SanGénClé}(\mathcal{SC}.\mathsf{param}).$$

$\mathcal{SC}$.Signe($\mathcal{SC}.\mathsf{param}, \mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}$) Le message est modifiable dans son intégralité, nous notons $\mathsf{ADM} = \{1, \ell\}$. Le signataire choisit aléatoirement r dans $\{0,1\}^\lambda$ et calcule

$$\mathsf{h} = \mathcal{HC}.\text{Hache}(\mathsf{dpk}, \mathsf{m}, \mathsf{r}) = g^{\mathsf{m}}\mathsf{dpk}^{\mathsf{r}} = g^{\mathsf{m}+\mathsf{dsk}\mathsf{r}}.$$

Il signe ensuite le condensé avec la signature $\mathcal{S}$: $\sigma = \mathcal{S}.\text{Signe}(\mathsf{spk}, \mathsf{ssk}, \mathsf{h})$ et obtient la signature caméléon $\Sigma = (\sigma, \mathsf{r})$.

$$\Sigma = (\sigma, \mathsf{r}) \leftarrow \mathcal{SC}.\text{Signe}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}).$$

$\mathcal{SC}$.MODIFIE($\mathcal{SC}$.param, m, Σ, spk, dsk, MOD) Ici MOD correspond directement au message modifié voulu car le message est modifiable en entier. Le délégué souhaite remplacer le message m par le message $\mathsf{m}' = \mathsf{MOD}$ dans la signature Σ, il calcule une collision de la manière suivante.

$$\mathsf{r}' = \mathcal{HC}.\text{FORGE}(\mathsf{dsk}, \mathsf{m}', \mathsf{m}, \mathsf{r}, \mathsf{h}) = \frac{\mathsf{m} - \mathsf{m}' + \mathsf{rdsk}}{\mathsf{dsk}}$$

Le délégué obtient la signature $\Sigma' = (\sigma, \mathsf{r}')$.

$$(\Sigma' = (\sigma, \mathsf{r}'), \mathsf{m}') \leftarrow \mathcal{SC}.\text{MODIFIE}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}).$$

$\mathcal{SC}$.VÉRIFIE($\mathcal{SC}$.param, m, Σ, spk, dpk) La vérification du schéma s'effectue en deux étapes. Le vérifieur reconstitue d'abord le condensé $\mathsf{h} = \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, \mathsf{m}, \mathsf{r}) = g^{\mathsf{m}}\mathsf{dpk}^{\mathsf{r}}$, puis retourne la sortie de la vérification de la signature sur celui-ci $\mathcal{S}$.VÉRIFIE(spk, σ, h).

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SC}.\text{VÉRIFIE}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk}).$$

Analyse de la solution.

Les algorithmes $\mathcal{SC}$.PROUVE et $\mathcal{SC}$.JUGE n'existent pas pour ce schéma, la **Responsabilité** n'y est pas prévue. Une signature originale est indistinguable d'une signature modifiée, le signataire ne peut pas prouver s'il a (ou n'a pas) généré une paire message-signature. Il n'existe donc pas de solution triviale pour établir la **Responsabilité** dans ce schéma.
D'autre part, ce schéma considère que l'intégralité du message peut être modifié, la propriété d'**Immuabilité** n'a donc pas de raison d'être.
Enfin, G.Ateniese et B.de Medeiros montrent dans [Ad04] qu'un problème de sécurité dans la fonction caméléon permet de retrouver la clé de délégation à partir de deux paires message-signature d'une même instance. Le principe de l'attaque proposée est le suivant.

> Supposons qu'un attaquant connaisse deux messages signés m_a et m_b d'une même instance. Il connaît donc r_a et r_b tels que $\mathcal{HC}.\text{HACHE}(\mathsf{dpk}, \mathsf{m}_a, \mathsf{r}_a) = \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, \mathsf{m}_b, \mathsf{r}_b)$. Il peut alors retrouver la clé sécrète du délégué : $\mathsf{dsk} = \frac{\mathsf{m}_a - \mathsf{m}_b}{\mathsf{r}_b - \mathsf{r}_a}$.

6.3.2 Solution de X.Chen, F.Zhang et K.Kim

Quatre ans après [KR00], à l'occasion d'ISC 2004, X.Chen, F.Zhang et K.Kim proposent une nouvelle solution pour le même type de proto-schéma de signature caméléon c'est-à-dire le cas où le délégué peut modifier tout le message. Ce schéma utilise, lui aussi, un schéma de signature **EU − CMA** sûr, noté $\mathcal{S}$ (cf. Section 29), ainsi que le schéma de fonction de hachage caméléon qu'ils décrivent, que nous noterons $\mathcal{HC}$. Il fonctionne de la manière suivante.

Construction 17.
Signature Caméléon de X.Chen, F.Zhang et K.Kim

$\mathcal{SC}$.INIT(1^λ) Soit $\mathbb{G}$ un groupe d'ordre p premier dans lequel le problème CDH est difficile et le problème DDH facile (cf. Définition 12 et 13). Soient g un générateur de $\mathbb{G}$ et H une fonction de hachage. De plus, le schéma de signature $\mathcal{S}$ est initialisé en fonction du paramètre de sécurité λ : $\mathcal{S}.\mathsf{param} = \mathcal{S}.\text{INIT}(\lambda)$.

$$\mathcal{SC}.\mathsf{param} = \{\mathbb{G}, g, p, H, \mathcal{S}.\mathsf{param}\} \leftarrow \mathcal{SC}.\text{INIT}(1^\lambda)$$

$\mathcal{SC}$.SigGénClé($\mathcal{SC}$.param) La paire de clés du signataire correspond à une paire de clés du schéma de signature $\mathcal{S}$. Il génère donc sa paire de clés en utilisant l'algorithme de génération de clés de $\mathcal{S}$: $(\mathsf{ssk}, \mathsf{spk}) = \mathcal{S}$.GénClé($\mathcal{S}$.param).

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SC}.\textsc{SigGénClé}(\mathcal{SC}.\mathsf{param}).$$

$\mathcal{SC}$.SanGénClé($\mathcal{SC}$.param) La clé secrète du délègue dsk est choisie aléatoirement dans $\{0,1\}^\lambda$. Le délégué calcule ensuite sa clé publique $\mathsf{dpk} = g^{\mathsf{dsk}}$. Il obtient la paire $(\mathsf{dsk}, \mathsf{dpk})$ qui correspond à une paire de clés de la fonction de hachage caméléon de [CZK04].

$$(\mathsf{dpk}, \mathsf{dsk}) \leftarrow \mathcal{SC}.\textsc{SanGénClé}(\mathcal{SC}.\mathsf{param}).$$

$\mathcal{SC}$.Signe($\mathcal{SC}$.param, m, ssk, dpk, ADM) Le message est modifiable dans son intégralité donc $\mathsf{ADM} = (1, \ell)$. Le signataire construit un identifiant de session $Id = H(Id_s \| Id_d \| Id_t)$ avec Id_s l'identifiant du signataire, Id_d l'identifiant du délégué et Id_t un identifiant de transaction. Il choisit aléatoirement r dans $\{0,1\}^\lambda$ et calcule g^{r}, $\mathsf{dpk}^{\mathsf{r}}$ puis le condensé sur le message :

$$\mathsf{h} = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}, \mathsf{dpk}^{\mathsf{r}}, Id) = (g \cdot Id)^{\mathsf{m}} \mathsf{dpk}^{\mathsf{r}}.$$

Il signe ensuite le condensé avec la signature $\mathcal{S}$: $\sigma = \mathcal{S}$.Signe(spk, ssk, h) et obtient la signature caméléon $\Sigma = (Id, g^{\mathsf{r}}, \mathsf{dpk}^{\mathsf{r}}, \sigma)$.

$$\Sigma = (Id, g^{\mathsf{r}}, \mathsf{dpk}^{\mathsf{r}}, \sigma) \leftarrow \mathcal{SC}.\textsc{Signe}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}).$$

$\mathcal{SC}$.Modifie($\mathcal{SC}$.param, m, Σ, spk, dsk, MOD) Ici MOD correspond directement au message modifié voulu car le message est modifiable en entier. Le délégué souhaite remplacer le message m par le message $\mathsf{m}' = \mathsf{MOD}$ dans la signature Σ, il calcule $A = g^{\mathsf{r}}(g \cdot Id)^{\mathsf{dsk}^{-1}(\mathsf{m}-\mathsf{m}')}$ et $B = \mathsf{dpk}^{\mathsf{r}}(g \cdot Id)^{(\mathsf{m}-\mathsf{m}')}$. Et obtient ainsi, pour un r' qu'il ne connait pas, les valeurs $g^{\mathsf{r}'} = A$ et $\mathsf{dpk}^{\mathsf{r}'} = B$ et ainsi la signature $\Sigma' = (Id, g^{\mathsf{r}'}, \mathsf{dpk}^{\mathsf{r}'}, \sigma)$.

$$(\Sigma' = (Id, g^{\mathsf{r}'}, \mathsf{dpk}^{\mathsf{r}'}, \sigma), \mathsf{m}') \leftarrow \mathcal{SC}.\textsc{Modifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}).$$

$\mathcal{SC}$.Vérifie($\mathcal{SC}$.param, m, Σ, spk, dpk) La vérification de la signature $\Sigma = (Id, g^{\mathsf{r}}, \mathsf{dpk}^{\mathsf{r}}, \sigma)$ s'effectue en trois étapes. Le vérifieur reconstitue d'abord le condensé.

$$\mathsf{h} = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}, \mathsf{dpk}^{\mathsf{r}}, Id) = (g \cdot Id)^{\mathsf{m}} \mathsf{dpk}^{\mathsf{r}}$$

Il vérifie ensuite la signature sur celui-ci. Si $\mathcal{S}$.Vérifie(spk, σ, h) retourne faux le vérifieur retourne faux. Sinon il vérifie que $(g, \mathsf{dpk}, g^{\mathsf{r}}, \mathsf{dpk}^{\mathsf{r}})$ est un quadruplet Diffie-Hellman et retourne vrai si c'est le cas et faux sinon.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SC}.\textsc{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk}).$$

Observations.

Les algorithmes $\mathcal{SC}$.Prouve et $\mathcal{SC}$.Juge n'existent pas dans le schéma initial, la **Responsabilité** n'y est pas prévue. Cependant, bien qu'une signature originale soit indistinguable d'une signature modifiée, le signataire peut prouver qu'il a généré une paire message-signature et donc obtenir la propriété de Délégué – **Responsabilité**. Pour cela je suppose que le signataire

conserve dans une base de données BD toutes les paires message-signature qu'il signe ainsi que les aléas r correspondants $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k, \mathsf{r}_k)\}_{k\in[1,\mathsf{q}]}$. Notons que je n'essaye pas d'atteindre la $\mathsf{Signataire}-$**Responsabilité**, en effet, nous verrons par la suite qu'elle n'est pas atteignable. L'algorithme $\mathcal{SC}.\textsc{Juge}$ que je propose ne juge ici que la responsabilité du signataire sur le message et donc retourne soit $\mathsf{Signataire}$ soit le symbole d'erreur $\perp$.

$\mathcal{SC}.\textsc{Prouve}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, \mathsf{BD})$ Le signataire cherche un indice j dans la base de données $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k, \mathsf{r}_k)\}_{k\in[1,\mathsf{q}]}$ tel que la paire (m, Σ) soit de la même instance que (m_j, Σ_j). La preuve qu'il n'a pas généré la paire (m, Σ) est $\pi_{\mathsf{or/mod}} = (\mathsf{m}_j, \Sigma_j, \mathsf{r}_j)$.

$$\pi_{\mathsf{or/mod}} = (\mathsf{m}_j, \Sigma_j, \mathsf{r}_j) \leftarrow \mathcal{SC}.\textsc{Prouve}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, \mathsf{BD})$$

$\mathcal{SC}.\textsc{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$ Le juge vérifie la validité de la signature $\Sigma = (Id, g^{\mathsf{r}}, \mathsf{dpk}^{\mathsf{r}}, \sigma)$ sur le message m. Si elle n'est pas valide, il retourne $\perp$. On note $\pi_{\mathsf{or/mod}} = (\mathsf{m}_j, \Sigma_j, \mathsf{r}_j)$ avec $\Sigma_j = (Id_j, A_j, B_j, \sigma_i)$ et le juge vérifie si :
- $Id = Id_j$,
- $\mathsf{vrai} = \mathcal{SC}.\textsc{Vérifie}(\mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$,
- $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}, g^{\mathsf{r}}, Id) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}_j, A_j, Id_j)$,
- et enfin si $A_j = g^{\mathsf{r}_j}$.

Si toutes les conditions sont vérifiées alors il retourne $\perp$ sinon il retourne $\mathsf{Signataire}$.

$$\{\mathsf{Signataire}, \perp\} \leftarrow \mathcal{SC}.\textsc{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$$

Analyse de la solution.

Ce schéma résiste à l'attaque d'exposition de clé proposée dans [Ad04], cependant une autre attaque permet à un attaquant de construire une paire modifiée de son choix à partir de deux paires message-signature valides d'une même instance (cf. Section 6.3). Cette attaque fonctionne de la manière suivante.

Supposons qu'un attaquant connaisse deux paires message-signature m_a et m_b d'une même instance. Il connait donc r_a et r_b tels que $(g \cdot Id)^{\mathsf{m}_a}\mathsf{dpk}^{\mathsf{r}_a} = (g \cdot Id)^{\mathsf{m}_b}\mathsf{dpk}^{\mathsf{r}_b}$. Il peut calculer une trappe $T = (\frac{g^{\mathsf{r}_b}}{g^{\mathsf{r}_a}})^{(\mathsf{m}_a - \mathsf{m}_b)^{-1}} \pmod p = (g \cdot Id)^{\mathsf{dsk}^{-1}}$.
Ainsi, pour tout message m^*, l'attaquant peut calculer :

$$A^* = g^{\mathsf{r}}T^{m-m^*} = g^{\mathsf{r}}(g \cdot Id)^{\mathsf{dsk}^{-1}(m-m^*)} \text{ et } B^* = \mathsf{dpk}^{\mathsf{r}}(g \cdot Id)^{m-m^*}$$

tels que $A^* = g^{\mathsf{r}^*}$ et $B^* = \mathsf{dpk}^{\mathsf{r}^*}$ pour un r^* qu'il ignore. Il obtient ainsi une signature modifiée $\Sigma^* = (Id, g^{\mathsf{r}^*}, \mathsf{dpk}^{\mathsf{r}^*}, \sigma)$ sur le message m^*.

Cette attaque permet non seulement de faire tomber l'**Infalsifiabilité** du schéma mais aussi montre l'impossibilité de la $\mathsf{Délégué}-$**Responsabilité**. En effet, le signataire peut utiliser cette attaque pour générer des paires message-signature modifiées à l'insu du délégué.
Enfin, le message est ici toujours modifiable dans son intégralité, la propriété d'**Immuabilité** n'est donc pas applicable.

6.3.3 Solution de G.Ateniese, D.H.Chou, B.De Medeiros et G.Tsudik

À ESORICS 2005, G.Ateniese, D.H.Chou, B.De Medeiros et G.Tsudik [ACdT05] proposent le premier schéma de signature caméléon selon la définition actuelle : Le signataire découpe le message en t parties égales et désigne certaines comme modifiables par le délégué, les autres parties restant non-falsifiables même par le délégué. La description du découpage et des parties

désignées comme modifiable ou non est décrite dans ADM (cf. Définition 43). Le cas où le message est entièrement modifiable n'est ici plus qu'un cas particulier. Ce schéma utilise, un schéma de signature $\mathbf{EU-CMA}$, noté $\mathcal{S}$ (voir Définition 29) ainsi qu'un schéma de fonction de hachage caméléon basée sur l'identité **résistant aux collisions** et **uniforme** $\mathcal{HC}$ (voir Définition 22). Il fonctionne de la manière suivante.

Construction 18.
Solution de G.Ateniese, D.H.Chou, B.De Medeiros et G.Tsudik

$\mathcal{SC}$.Init(1^λ) Cette étape initialise le schéma de signature $\mathcal{S}.\mathsf{param} = \mathcal{S}.\textsc{Init}(\lambda)$ et la fonction de hachage caméléon $\mathcal{HC}.\mathsf{param} = \mathcal{HC}.\textsc{Init}(\lambda)$ en fonction du paramètre de sécurité λ.

$$\mathcal{SC}.\mathsf{param} = \{\mathcal{S}.\mathsf{param}, \mathcal{HC}.\mathsf{param}\} \leftarrow \mathcal{SC}.\textsc{Init}(1^\lambda)$$

$\mathcal{SC}$.SigGénClé($\mathcal{SC}$.param) La paire de clés du signataire correspond à une paire de clés de signataire du schéma de signature choisi $(\mathsf{spk}, \mathsf{ssk}) = \mathcal{S}.\textsc{GénClé}(\mathcal{S}.\mathsf{param})$.

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SC}.\textsc{SigGénClé}(\mathcal{SC}.\mathsf{param})$$

$\mathcal{SC}$.SanGénClé($\mathcal{SC}$.param La paire de clés du délégué correspond à une paire de clés d'une fonction caméléon sûre $(\mathsf{dsk}, \mathsf{dpk}) = \mathcal{HC}.\textsc{GénClé}(\mathcal{HC}.\mathsf{param})$.

$$(\mathsf{dpk}, \mathsf{dsk}) \leftarrow \mathcal{SC}.\textsc{SanGénClé}(\mathcal{SC}.\mathsf{param})$$

$\mathcal{SC}$.Signe($\mathcal{SC}$.param, m, ssk, dpk, ADM) Le signataire choisit aléatoirement dans $\{0,1\}^\lambda$ un identifiant unique Id_m. Le message m est décomposé en fonction de ADM : $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_\mathsf{t}$. Le signataire choisit aléatoirement un r_i pour chaque partie modifiable, on note $\mathsf{R} = \{\mathsf{r}_i, \forall i \in \mathsf{ADM}\}$. Le signataire calcule ensuite un message intermédiaire de la façon suivante.

$$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, Id_\mathsf{m} \| i \| \mathsf{m}_i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i || i & \text{sinon} \end{cases}.$$

Enfin, le signataire génère une signature : $\sigma = \mathcal{S}.\textsc{Signe}(\mathsf{spk}, \mathsf{ssk}, Id_\mathsf{m} \| \mathsf{t} \| \mathsf{dpk} \| \tilde{\mathsf{m}})$. La signature caméléon sur le message m est $\Sigma = (\sigma, \mathsf{R}, \mathsf{ADM}, Id_\mathsf{m}, \mathsf{dpk})$ avec $\mathsf{R} = \{\mathsf{r}_i | i \in \mathsf{ADM}\}$. Le signataire ajoute la signature et le message dans sa base de données BD.

$$\Sigma = (\sigma, \mathsf{R}, \mathsf{ADM}, Id_\mathsf{m}, \mathsf{dpk}) \leftarrow \mathcal{SC}.\textsc{Signe}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM})$$

$\mathcal{SC}$.Modifie($\mathcal{SC}$.param, m, Σ, spk, dsk, MOD) Le délégué souhaite à présent transformer le message $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_\mathsf{t}$ en fonction de MOD, c'est-à-dire en le message $\mathsf{m}' = \mathsf{m}'_1 \| \cdots \| \mathsf{m}'_\mathsf{t}$. Pour chaque partie m_j qu'il souhaite modifier, il calcule :

$$\mathsf{r}'_j = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, Id_\mathsf{m} \| j \| \mathsf{m}_j, Id_\mathsf{m} \| j \| \mathsf{m}'_j, \mathsf{r}_j, \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, Id_\mathsf{m} \| j \| \mathsf{m}_j, \mathsf{r}_j)).$$

La nouvelle signature est $\Sigma = (\sigma, \mathsf{R}', \mathsf{ADM}, Id_\mathsf{m}, \mathsf{dpk})$ avec $\mathsf{R}' = \{\mathsf{r}'_i, i \in \mathsf{ADM}\}$ en considérant que $\mathsf{r}'_i = \mathsf{r}_i$ si m_i n'a pas été modifié.

$$(\Sigma' = (\sigma, \mathsf{R}', \mathsf{ADM}, Id_\mathsf{m}, \mathsf{dpk}), \mathsf{m}') \leftarrow \mathcal{SC}.\textsc{Modifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD})$$

$\mathcal{SC}$.VÉRIFIE($\mathcal{SC}$.param, m, Σ, spk, dpk) Le vérifieur reconstruit le message intermédiaire.

$$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, Id_{\mathsf{m}} \| i \| \mathsf{m}_i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i || i & \text{sinon} \end{cases}$$

Puis vérifie la signature $\mathcal{S}$.VÉRIFIE(spk, $Id_{\mathsf{m}} \| \mathsf{t} \| \mathsf{dpk} \| \tilde{\mathsf{m}}, \sigma$). Il retourne vrai si la signature est valide et faux si elle ne l'est pas.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SC}.\text{VÉRIFIE}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk})$$

$\mathcal{SC}$.PROUVE($\mathcal{SC}$.param, m, Σ, ssk, dpk, BD) En cas de problème, le signataire prouve qu'il n'a pas généré une paire valide (m, Σ) en divulguant au Juge le message original et sa signature sur celui-ci. Pour cela il cherche dans $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k \in [1,\mathsf{q}]}$ une paire (m_j, Σ_j) telle que :
– $Id_{\mathsf{m}_j} = Id_{\mathsf{m}}$,
– $\forall i \in [1,\mathsf{t}]$:

$$\begin{cases} \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, Id_{\mathsf{m}_j} \| i \| \mathsf{m}_{j_i}, \mathsf{r}_{j_i}) = \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, Id_{\mathsf{m}} \| i \| \mathsf{m}_i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_{j_i} = \mathsf{m}_i & \text{sinon} \end{cases}$$

– et $\sigma_j = \sigma$.
Si une telle paire existe alors $\pi_{\mathsf{or/mod}} = (\mathsf{m}_j, \Sigma_j)$ sinon $\pi_{\mathsf{or/mod}} = \perp$.

$$\pi_{\mathsf{or/mod}} \leftarrow \mathcal{SC}.\text{PROUVE}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, \mathsf{BD})$$

$\mathcal{SC}$.JUGE($\mathcal{SC}$.param, m, Σ, $\pi_{\mathsf{or/mod}}$) Si la preuve contient $\perp$ alors le juge retourne Signataire, sinon il vérifie la preuve, c'est-à-dire, en notant $\pi_{\mathsf{or/mod}} = (\mathsf{m}_j, \Sigma_j)$, si :
– vrai = $\mathcal{SC}$.VÉRIFIE($\mathcal{SC}$.param, m_j, Σ_j, spk, dpk),
– $Id_{\mathsf{m}_j} = Id_{\mathsf{m}}$,
– $\forall i \in [1,\mathsf{t}]$:

$$\begin{cases} \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, Id_{\mathsf{m}_j} \| i \| \mathsf{m}_{j_i}, \mathsf{r}_{j_i}) = \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, Id_{\mathsf{m}} \| i \| \mathsf{m}_i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_{j_i} = \mathsf{m}_i & \text{sinon} \end{cases}$$

– et $\sigma_j = \sigma$.
Si c'est le cas, il retourne Délégué sinon il retourne Signataire.

$$\{\mathsf{Signataire}, \mathsf{Délégué}\} \leftarrow \mathcal{SC}.\text{JUGE}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$$

Analyse de la solution.

Il existe une attaque contre l'**Infalsifiabilité** de ce schéma qui permet, à partir d'une combinaison de messages d'une même instance, de générer une signature modifiée valide sur un nouveau message. Cette attaque fonctionne de la manière suivante.

Supposons qu'un attaquant connaisse deux messages signés $\mathsf{m}_a = \mathsf{m}_1 \| \mathsf{m}_{a2} \| \mathsf{m}_3 \| \mathsf{m}_{a4}$ et $\mathsf{m}_b = \mathsf{m}_1 \| \mathsf{m}_{b2} \| \mathsf{m}_3 \| \mathsf{m}_{b4}$ d'une même instance tels que les parties 2 et 4 soient modifiables et différentes ($\mathsf{m}_{a2} \neq \mathsf{m}_{b2}$ et $\mathsf{m}_{a4} \neq \mathsf{m}_{b4}$). Il connait donc les signatures correspondantes : $\Sigma_a = (\sigma, \{\mathsf{r}_{a_2}, \mathsf{r}_{a_4}\}, \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk})$ et $\Sigma_b = (\sigma, \{\mathsf{r}_{b2}, \mathsf{r}_{b4}\}, \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk})$. Il peut alors construire une signature sur le message $\mathsf{m}^* = \mathsf{m}_1 \| \mathsf{m}_{a2} \| \mathsf{m}_3 \| \mathsf{m}_{b4}$: $\Sigma^* = (\sigma, \{\mathsf{r}_{a2}, \mathsf{r}_{b4}\}, \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk})$.

De plus, un signataire corrompu peut faire accuser le délégué par le Juge à partir du moment où celui-ci a généré au moins une paire message-signature modifiée. Pour cela, il lui suffit de retourner comme preuve la paire modifiée au lieu de la paire originale. Le schéma n'est donc pas Signataire-**Responsable**.
Notons, que le délégué n'est jamais capable de faire accuser le signataire. Ce schéma est donc Délégué-**Responsable**.

Il n'existe pas d'attaque connue contre les deux derniers schémas. Tout deux sont basés sur la solution de [ACdT05].

6.3.4 Solution de S.Canard, F.Laguillaumie et M.Milhau

S.Canard, F.Laguillaumie et M.Milhau [CLM08] proposent un schéma respectant un sous-ensemble des propriétés esquissées dans [ACdT05]. Ils mettent de coté l'**Intimité** et la **Transparence** et se concentrent sur la propriété de **Responsabilité**. Cette propriété est atteinte en utilisant le message original comme un identifiant d'instance.

[CLM08] se base sur [ACdT05] et contre l'attaque proposée sur l'**Infalsifiabilité** en ajoutant une partie supplémentaire au message intermédiaire. Cette partie est un condensé caméléon du message courant. De plus, il utilise le message original comme identifiant d'instance ce qui permet d'obtenir la **Responsabilité**. Cependant ce choix d'identifiant implique d'abandonner l'**Intimité** et donc la **Transparence**.

6.3.5 Solution de C.Brzuska *et al.*

Enfin, C.Brzuska, M.Fischlin, T.Freudenreich, A.Lehmann, M.Page, J.Schelbert, D.Schröder et F.Volk proposent dans [BFF+09] un schéma de signature caméléon prouvé sûr pour toutes les propriétés bien que peu efficace pour le délégué.

[BFF+09] se base lui aussi sur [ACdT05] mais propose une autre méthode pour résoudre les problèmes de sécurité du schéma initial. L'identifiant d'instance est remplacé par un identifiant de message TAG qui sera construit de façon différente par le signataire et le délégué. Le signataire construit TAG à partir d'une clé secrète κ et d'un aléa Nonce de la façon suivante : $\mathsf{TAG} = \mathcal{PRG}(x)$ avec $x = \mathcal{PRF}(\kappa, \mathsf{Nonce})$. Le délégué choisit le TAG aléatoirement. Les deux constructions sont indistinguables, cela n'affecte donc pas la **Transparence** du schéma. Par contre, le signataire peut maintenant prouver qu'il a construit une paire message-signature en exhibant le x correspondant au TAG utilisé dans la signature originale. Le TAG étant lié à un message, l'attaque sur l'**Infalsifiabilité** de [ACdT05] ne fonctionne plus.

On notera cependant que, l'identifiant de message étant inséré dans chaque partie modifiable, le délégué devra générer une collision pour chaque partie modifiable par le délégué quelque soit le nombre de parties de m qu'il souhaitait changer. Cette solution ne permet donc pas une méthode de modification efficace.

En conclusion, nous pouvons résumer les propriétés de sécurité atteintes ✓ ou non ✗ par les différents schémas dans le tableau suivant (Table 6.1).

6.4 Notre nouvelle construction

Nous allons voir à présent la construction que nous avons proposée avec Sébastien Canard à CT-RSA 2010 [CJ10a]. Il s'agit d'un schéma de signature caméléon sûr dans le modèle de Brzuska *et al.* et dont la complexité pour le délégué est linéaire en le nombre de parties qu'il

	Immuable	**Transparent**	**Intime.**	**Responsable** sig-**Resp**	del-**Resp**	**Infal.**
[KR00]	Non applicable	✓	✓	Non applicable		✗
[CZK04]	Non applicable	✓	✓	✗	✓	✗
[ACdT05]	✓	✓	✓	✗	✓	✗
[CLM08]	✓	✗	✗	✓	✓	✓
[BFF+09]	✓	✓	✓	✓	✓	✓

TABLE 6.1 – Sécurité des schémas de l'état de l'art.

modifie. Notre solution est ainsi plus efficace que la solution [BFF+09], notamment pour les modifications. D'autre part, notre solution est plus modulable et a été conçue afin d'envisager des extensions (cf Chapitre 7).

Dans cette section, je décrirai d'abord notre construction puis je prouverai sa sécurité.

6.4.1 Le schéma

Le schéma est construit sur la même base que [ACdT05]. Nous pallions les problèmes de sécurité de ce schéma en combinant deux procédés. Premièrement nous utilisons l'idée de [CLM08] en ajoutant une partie finale au message intermédiaire qui correspond à un condensé caméléon du message complet. Ceci permet de contrer l'attaque contre l'**Infalsifiabilité** du schéma [ACdT05].

D'autre part, nous utilisons l'idée de [BFF+09] pour obtenir la responsabilité. Nous utilisons leur concept de TAG généré de manière pseudo-aléatoire (et prouvable) par le signataire, et de manière aléatoire par le délégué.

Cependant, contrairement à [BFF+09], nous n'insérons pas ce TAG dans chaque partie modifiable mais nous utilisons la partie additionnelle pour cela. Ainsi nous pouvons concilier le niveau de sécurité du schéma [BFF+09] tout en permettant au délégué de ne pas calculer une collision pour chaque partie modifiable mais seulement sur chaque partie qu'il souhaite modifier plus une sur le message intégral. Plus formellement, la construction est la suivante.

Construction 19.
Notre solution Le schéma utilise un schéma de signature $\mathbf{EU-CMA}$ que nous notons $\mathcal{S}$ (cf. Définition 29), un schéma de fonction de hachage caméléon $\mathcal{HC}$ **uniforme** et **résistant aux collisions** (cf. Définition 22), un schéma de fonction pseudo aléatoire $\mathcal{PRF}$ (cf. Définition 24) et un schéma de générateur pseudo-aléatoire $\mathcal{PRG}$ (cf. Définition 23). Le schéma fonctionne de la manière suivante.

$\mathcal{SC}$.INIT(1^λ). Cet algorithme prend en entrée 1^λ avec λ un paramètre de sécurité et initialise le système. Ce qui correspond ici à l'exécution de $\mathcal{HC}$.INIT(λ) pour obtenir les paramètres de la fonction caméléon $\mathcal{HC}.\mathsf{param}$ et de $\mathcal{S}$.INIT(λ) pour les paramètres du schéma de signature $\mathcal{S}.\mathsf{param}$. On note $\mathcal{SC}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, \mathcal{S}.\mathsf{param}\}$.

$$\mathcal{SC}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, \mathcal{S}.\mathsf{param}\} \leftarrow \mathcal{SC}.\text{INIT}(1^\lambda).$$

$\mathcal{SC}$.SIGGÉNCLÉ($\mathcal{SC}.\mathsf{param}$). Cet algorithme exécute l'algorithme de génération des clés du schéma de signature $\mathcal{S}$.GÉNCLÉ(1^λ) et obtient $(\mathsf{pk}, \mathsf{sk})$. Il choisit ensuite κ aléatoirement dans $\{0,1\}^\lambda$.

$$(\mathsf{spk} = \mathsf{pk}, \mathsf{ssk} = \{\mathsf{sk}, \kappa\}) \leftarrow \mathcal{SC}.\text{SIGGÉNCLÉ}(\mathcal{SC}.\mathsf{param})$$

$\mathcal{SC}$.SanGénClé($\mathcal{SC}$.param). L'algorithme exécute $\mathcal{HC}$.GénClé(1^λ) et utilise la paire de clés obtenue comme clés de délégué (dsk, dpk).

$$(\mathsf{dsk}, \mathsf{dpk}) \leftarrow \mathcal{SC}.\textsc{SanGénClé}(\mathcal{SC}.\mathsf{param})$$

$\mathcal{SC}$.Signe($\mathcal{SC}$.param, m, ssk, dpk, ADM). Le signataire génère d'abord un TAG pour son message. Il choisit aléatoirement $\mathsf{Nonce} \in \{0,1\}^\lambda$, calcule $\mathsf{x} = \mathcal{PRF}(\kappa, \mathsf{Nonce})$ et obtient ainsi $\mathsf{TAG} = \mathcal{PRG}(\mathsf{x})$. Il choisit alors $|\mathsf{ADM}| + 1$ aléas dans $\{0,1\}^\lambda$ que nous notons $\mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}, \mathsf{r}_c\}$. Le signataire exécute ensuite une *procédure de reconstruction* sur le message m.

> *Procédure de reconstruction* :
> Cette procédure publique utilise TAG, les aléas R, la clé publique du délégué dpk et celle du signataire spk :
>
> 1. Pour chaque partie, on calcule une valeur intermédiaire $\tilde{\mathsf{m}}_i$.
>
> $$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}_i||i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i||i & \text{sinon} \end{cases}$$
>
> 2. Puis on calcule une partie finale supplémentaire.
>
> $$\mathsf{h}_c = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c)$$
>
> Le message reconstitué est $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1\| \cdots \|\tilde{\mathsf{m}}_t\|\mathsf{h}_c\|\mathsf{dpk}$.

Le signataire signe le message reconstitué grâce au schéma de signature choisit : $\sigma = \mathcal{S}.\textsc{Signe}(\mathsf{spk}, \mathsf{ssk}, \tilde{\mathsf{m}})$. Et il obtient la signature caméléon suivante.

$$\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}) \text{ avec } \mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}, \mathsf{r}_c\}$$

La signature et le message correspondant sont ajoutés à la base de données du signataire BD.

$$\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}) \leftarrow \mathcal{SC}.\textsc{Signe}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}).$$

$\mathcal{SC}$.Modifie($\mathcal{SC}$.param, m, Σ, spk, dsk, MOD). Le délégué vérifie que MOD correspond à ADM, si ce n'est pas le cas il s'arrête en retournant $\perp$. Il met à jour les tailles ℓ_i des parties qui vont être modifiées puis choisit aléatoirement Nonce′ et TAG′ dans $\{0,1\}^\lambda$. Le délégué effectue la *procédure de reconstruction* comme décrit précédemment et obtient $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1\| \cdots \|\tilde{\mathsf{m}}_t\|\mathsf{h}_c\|\mathsf{dpk}$ et donc les valeurs h_c et $\tilde{\mathsf{m}}_i = \mathsf{h}_i$ pour tout $i \in \mathsf{MOD}$. Il construit les collisions sur celles-ci grâce à sa clé secrète de délégué et à l'algorithme $\mathcal{HC}$.Forge de la manière suivante.

– Pour tout $i \in \mathsf{MOD}$ le délégué calcule $\mathsf{r}'_i = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, \mathsf{m}_i\|i, \mathsf{m}'_i\|i, \mathsf{r}_i, \tilde{\mathsf{m}}_i)$,
– puis $\mathsf{r}'_c = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{TAG}'\|\mathsf{m}'\|\mathsf{spk}, \mathsf{r}_c, \mathsf{h}_c)$.

Enfin, il met à jour l'ensemble $\mathsf{R}' = \{\mathsf{r}'_1, \cdots, \mathsf{r}'_u, \mathsf{r}'_c\}$ de la manière suivante $\mathsf{r}'_i = \mathsf{r}_i$ si $i \notin \mathsf{MOD}$, sinon $\mathsf{r}'_i = \mathsf{r}'_i$. La signature modifiée obtenue est $\Sigma' = (\sigma, \mathsf{TAG}', \mathsf{Nonce}', \mathsf{R}', \mathsf{ADM}')$.

$$(\Sigma' = (\sigma, \mathsf{TAG}', \mathsf{Nonce}', \mathsf{R}', \mathsf{ADM}'), \mathsf{m}') \leftarrow \mathcal{SC}.\textsc{Modifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}).$$

$\mathcal{SC}$.Vérifie($\mathcal{SC}$.param, m, Σ, spk, dpk). Le vérifieur utilise la *procédure de reconstruction* décrite plus haut afin d'obtenir le message intermédiaire $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1|| \cdots ||\tilde{\mathsf{m}}_t||\mathsf{h}_c||\mathsf{dpk}$. Il retourne ensuite la sortie de l'algorithme de vérification sur le message $\mathcal{S}.\textsc{Vérifie}(\mathsf{spk}, \sigma, \tilde{\mathsf{m}})$.

$\mathcal{SC}$.Prouve($\mathcal{SC}$.param, m, Σ, ssk, dpk, BD $= (\mathsf{m}_k, \Sigma_k)_{k\in[1,\mathsf{q}]}$). Le signataire cherche un indice $j \in [1,\mathsf{q}]$ dans sa base de données de couples signature-message $\mathsf{BD} = (\mathsf{m}_k, \Sigma_k)_{k\in[1,\mathsf{q}]}$ tel que :

– vrai $= \mathcal{SC}.\textsc{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;

– $\mathsf{m}_j \neq \mathsf{m}$;
– $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}_j\|\mathsf{m}_j\|\mathsf{spk}, \mathsf{r}_{c_j})$
avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.
S'il existe un tel j alors $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ avec $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j)$ et $\mathsf{r}_{c_j} \in \mathsf{R}_j$ sinon $\pi_{\mathsf{or/mod}} = \perp$.

$$\pi_{\mathsf{or/mod}} \leftarrow \mathcal{SC}.\textsc{Prouve}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, (\mathsf{m}_k, \Sigma_k)_{k\in[1,\mathsf{q}]})$$

$\mathcal{SC}.\textsc{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$. L'algorithme vérifie d'abord si Σ est une signature valide de m, si $\mathcal{SC}.\textsc{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk})$ retourne faux alors il retourne $\perp$. Sinon, il s'intéresse à $\pi_{\mathsf{or/mod}}$.
Si $\pi_{\mathsf{or/mod}} = \perp$ alors il retourne $\mathsf{Signataire}$ sinon il vérifie que $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ respecte les propriétés suivantes :
– $\mathsf{vrai} = \mathcal{SC}.\textsc{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
On peut donc poser $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j)$ avec $\mathsf{R}_j = \{\mathsf{r}_{1_j}, \cdots, \mathsf{r}_{u_j}, \mathsf{r}_{c_j}\}$.
– $\mathsf{m}_j \neq \mathsf{m}$;
– $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}_j\|\mathsf{m}_j\|\mathsf{spk}, \mathsf{r}_{c_j})$
avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.
Si la preuve $\pi_{\mathsf{or/mod}}$ respecte les trois propriétés, il retourne $\mathsf{Délégué}$ sinon il retourne $\mathsf{Signataire}$.

6.4.2 Preuve de sécurité du schéma

Théorème 7. *Sous l'hypothèse que le schéma de signature $\mathcal{S}$ soit* $\boldsymbol{EU-CMA}$ *sûr, que les fonctions $\mathcal{PRF}$ et PRG choisies soient pseudo-aléatoires et que la fonction de hachage caméléon $\mathcal{HC}$ soit* ***uniforme*** *et* ***résistante aux collisions****, le nouveau schéma de signature caméléon est un schéma de signature caméléon sûr dans le modèle de Brzuska* et al.

Démonstration. Nous allons à présent démontrer le théorème de sécurité, pour cela nous devons montrer que le schéma est **Immuable**, **Transparent** et **Responsable** :

Immuabilité. Nous utilisons le fait que la signature utilisée $\mathcal{S}$ est $\mathbf{EU-CMA}$ sûre et que les parties non modifiables du message sont signées avec $\mathcal{S}$.
Plus formellement, nous montrons que s'il existe un adversaire contre l'**Immuabilité** du schéma, que nous notons $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$, alors nous pouvons l'utiliser pour construire un adversaire, noté $\mathcal{A}^{\mathcal{S}}_{\mathbf{EU-CMA}}$, contre la **Falsification universelle** du schéma de signature.
Au début de l'expérience, le simulateur reçoit du challengeur la clé publique pk qui sera utilisée pour les signatures de l'oracle de signature $\mathcal{S}.\textsc{Signe}$. Le simulateur fournit à l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ cette clé publique pk comme clé publique du signataire des futures signatures caméléons ($\mathsf{spk} = \mathsf{pk}$). Le simulateur choisit ensuite aléatoirement une clé secrète κ dans $\{0,1\}^\lambda$ pour la $\mathcal{PRF}$ du schéma. Il peut ainsi simuler les oracles nécessaires à $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ de la manière suivante :
– $\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$. Supposons que $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ demande une signature caméléon sur un message m avec la clé publique de délégué dpk et la variable ADM. Le simulateur choisit aléatoirement Nonce dans $\{0,1\}^\lambda$ puis génère un $\mathsf{TAG} = \mathcal{PRG}(x)$ avec $x = \mathcal{PRF}(\kappa, \mathsf{Nonce})$. Il choisit ensuite aléatoirement un aléa par partie modifiable décrite dans ADM : $\mathsf{R} = \{\mathsf{r}_i, \forall i \in \mathsf{ADM}\}$. Puis il reconstruit les parties intermédiaires :

$$\forall i, \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}_i||i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i||i & \text{sinon} \end{cases}$$

Le simulateur calcule ensuite $\mathsf{h}_c = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c)$ et obtient un message intermédiaire $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1||\cdots||\tilde{\mathsf{m}}_t||\mathsf{h}_c||\mathsf{dpk}$. Il utilise enfin l'oracle de signature du challengeur sur le message $\tilde{\mathsf{m}}$ afin d'obtenir la signature σ. Il retourne à l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ la signature caméléon correspondant à sa requête : $\Sigma = \{\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}\}$ avec $\mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_u, \mathsf{r}_c\}$. Pour finir, le simulateur intègre la paire (m, Σ) dans une base de données, notée BD.

– $\mathcal{O}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{BD})$. Supposons que l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ demande une preuve sur un couple message-signature (m, Σ) avec pour entrées une clé publique de délégué dpk et une base de données de couple message signature $\mathsf{BD}' = (\mathsf{m}_k, \Sigma_k)_{k=[1,\mathsf{q}]}$. Le simulateur cherche dans cette base de données une entrée (m_j, Σ_j) telle que :
  – $\mathsf{vrai} = \mathcal{SC}.\textsc{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
  – $\mathsf{m}_j \neq \mathsf{m}$;
  – $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}_j\|\mathsf{m}_j\|\mathsf{spk}, \mathsf{r}_{c_j})$ avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.

  S'il trouve une telle entrée il retourne la preuve $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ avec $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j)$ et $\mathsf{r}_{c_j} \in \mathsf{R}_j$ sinon il retourne $\perp$.

A la fin de l'expérience, $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ retourne un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$. Supposons que Σ^* est une signature caméléon valide de m^* avec la clé publique de délégation dpk^*, l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ gagne l'expérience si pour toutes les requêtes faites aux oracles :
– soit dpk^* est une clé publique différentes de celles utilisées pour le message m^*,
– soit il existe une partie m^*_j différente de la $j$ème partie du message de la requête alors que $j$ème partie du message n'était pas modifiable.

Le simulateur calcule le message intermédiaire $\tilde{\mathsf{m}}^* = \tilde{\mathsf{m}}^*_1||\cdots||\tilde{\mathsf{m}}^*_t||{\mathsf{h}^*}_c||\mathsf{dpk}^*$. Si l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ a gagné l'expérience alors le message $\tilde{\mathsf{m}}^*$ est différent de tous les messages demandées au challengeur car pour chaque requête soit la partie contenant dpk^* était différente soit la partie $\tilde{\mathsf{m}}^*_j$ était différente. Le simulateur gagne donc l'expérience $\mathbf{EU-CMA}$ en retournant la signature Σ^* sur le message $\tilde{\mathsf{m}}^*$.

En conclusion, la probabilité de succès d'un adversaire contre l'**Immuabilité** du schéma est :

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathbf{Imm},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU-CMA},\mathcal{A}}(\lambda).$$

□

Transparence. Nous utilisons le fait que les seules différences entre une signature originale et une signature modifiée résident dans la construction du TAG et des aléas de R. Plus formellement, nous utilisons une preuve par jeu. Les échanges entre l'adversaire $\mathcal{A}$ et les oracles sont simulés par un distingueur $\mathcal{D}$ qui cherche à casser l'**uniformité** de la fonction caméléon $\mathcal{HC}$ ou le fait que le générateur $\mathcal{PRG}$ soit **pseudo-aléatoire** :

Jeu 0 : Le Jeu 0 correspond au jeu joué par l'adversaire dans l'expérience originale. Dans cette attaque nous nous intéressons à la forme du message obtenu grâce à la requête de challenge à l'oracle $\mathcal{O}.\textsc{Modif/Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$ qui dépend du bit b : si $b = 0$, $\mathcal{D}$ retourne une paire signature originale Σ_0 avec $\mathsf{TAG}_0 = \mathcal{PRG}(x_0)$ et R_0 un ensemble d'aléas, si $b = 1$ il retourne une signature modifiée Σ_1 avec TAG_1 un aléa et R_1 les sorties de l'algorithme $\mathcal{HC}.\textsc{Forge}$.

À la fin de son attaque, l'adversaire retourne un bit b'. On note S_0 l'événement $b = b'$ dans le Jeu 0. On obtient :

$$\mathsf{Adv}^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda) = 2|Pr[S_0] - 1/2|.$$

Jeu 1 : $\mathcal{D}$ modifie sa réponse à la requête à $\mathcal{O}.\textsc{Modif/Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$. Dans le

cas $b = 0$, le distingueur effectue une signature caméléon classique sur le message challenge mais en utilisant un x_0 choisit aléatoirement. Il retourne donc une signature Σ_0 avec TAG_0 construit à partir d'un aléa et R_0 un ensemble d'aléas. Le cas $b = 1$ reste inchangé.

La modification effectuée est indistinguable pour l'adversaire s'il n'est pas capable de distinguer un aléa d'une sortie de la fonction pseudo-aléatoire $\mathcal{PRF}$ tout en connaissant la graine Nonce_0. Si la modification est distinguable par l'adversaire, alors $\mathcal{D}$ peut utiliser cet adversaire contre la propriété de fonction **pseudo-aléatoire** de la $\mathcal{PRF}$. La différence entre les deux jeux est donc bornée de la façon suivante :

$$|Pr[S_1] - Pr[S_0]| \leqslant \mathsf{Adv}^{\mathcal{PRF}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda).$$

Jeu 2 : Dans le Jeu 2, le distingueur modifie à nouveau sa réponse à la requête à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$. Dans le cas $b = 0$, $\mathcal{D}$ effectue une signature caméléon classique sur le message challenge mais en utilisant cette fois un TAG_0 choisit aléatoirement. Il retourne donc une signature Σ_0 avec TAG_0 un aléa et R_0 un ensemble d'aléas. Le cas $b = 1$ reste inchangé.
La modification effectuée est indistinguable pour l'adversaire s'il n'est pas capable de distinguer un aléa d'une sortie du générateur pseudo-aléatoire $\mathcal{PRG}$. Si la modification est distinguable par l'adversaire, alors $\mathcal{D}$ peut utiliser cet adversaire contre la propriété de générateur **pseudo-aléatoire** du schéma $\mathcal{PRG}$. La différence entre les deux jeux est donc bornée de la façon suivante :

$$|Pr[S_2] - Pr[S_1]| \leqslant \mathsf{Adv}^{\mathcal{PRG}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda).$$

Jeu 3 : Dans ce jeu, le distingueur modifie à nouveau sa réponse à la requête à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$: Dans le cas $b = 1$, le distingueur calcule une signature caméléon originale sur le message challenge en utilisant un TAG_1 choisit aléatoirement. Il retourne donc une signature Σ_1 avec TAG_1 un aléa et R_1 un ensemble d'aléas. Le cas $b = 0$ reste inchangé.
La modification effectuée est indistinguable par l'adversaire s'il n'est pas capable de distinguer un aléa d'une sortie de $\mathcal{HC}.\textsc{Forge}$. Si la modification est distinguable par l'adversaire, alors $\mathcal{D}$ peut utiliser cet adversaire contre l'**uniformité** du schéma de hachage caméléon $\mathcal{HC}$. La différence entre les deux jeux est donc bornée de la façon suivante :

$$|Pr[S_3] - Pr[S_2]| \leqslant \mathsf{Adv}^{\mathcal{HC}}_{\mathbf{uni},\mathcal{A}}(\lambda).$$

Or dans ce dernier jeu les réponses sont identiques quelque soit la valeur de b. Donc, nous avons :

$$Pr[S_3] = 1/2.$$

Il est à présent possible d'évaluer l'avantage de l'adversaire contre la **Transparence** du schéma. Cet avantage est donné par :

$$\begin{aligned} Adv^{\mathcal{SC}}_{\mathbf{Transp},\mathcal{A}}(\lambda) &= 2|Pr[S_0] - 1/2| \\ &= 2|Pr[S_0] - Pr[S_3]| \\ &\leqslant 2(|Pr[S_1] - Pr[S_0]| + |Pr[S_2] - Pr[S_1]| + |Pr[S_3] - Pr[S_2]|) \\ &\leqslant 2\mathsf{Adv}^{\mathcal{PRF}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) + 2\mathsf{Adv}^{\mathcal{PRG}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) + 2\mathsf{Adv}^{\mathcal{HC}}_{\mathbf{uni},\mathcal{A}}(\lambda). \end{aligned}$$

□

Délégué-**Responsabilité**. Nous utilisons le fait que le schéma de signature utilisé $\mathcal{S}$ est **EU−CMA** sûr et que si la preuve honnêtement construite sur le message challenge retourne Signataire alors l'attaquant a, au minimum, falsifié une des deux parties supplémentaire du message intermédiaire.

Plus formellement, nous montrons que s'il existe un adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ contre la Délégué-**Responsabilité** du schéma alors nous pouvons l'utiliser pour construire un adversaire $\mathcal{A}^{\mathcal{S}}_{\mathbf{EU-CMA}}$ contre le schéma de signature.

Au début du jeu, le simulateur reçoit du challengeur la clé publique pk qui sera utilisée pour les signatures de l'oracle de signature $\mathcal{S}$.Signe. Le simulateur fournit à l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ cette clé publique pk comme clé publique des futures signatures caméléons ($\mathsf{spk} = \mathsf{pk}$). Le simulateur choisit ensuite aléatoirement une clé secrète κ dans $\{0,1\}^\lambda$ pour la $\mathcal{PRF}$ du schéma. Il peut ainsi simuler les oracles nécessaires à $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ de la manière suivante :

– $\mathcal{O}$.Signe($\mathsf{m}, \mathsf{dpk}, \mathsf{ADM}$). Supposons que $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ demande une signature caméléon sur un message m utilisant la clé publique de délégué dpk et la variable ADM. Le simulateur choisit aléatoirement Nonce dans $\{0,1\}^\lambda$ puis génère un $\mathsf{TAG} = \mathcal{PRG}(x)$ avec $x = \mathcal{PRF}(\kappa, \mathsf{Nonce})$. Il choisit ensuite aléatoirement un aléa par partie modifiable décrite dans ADM : $\mathsf{R} = \{\mathsf{r}_i, \forall i \in \mathsf{ADM}\}$. Il reconstruit les parties intermédiaires :
$$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = \mathcal{HC}.\text{Hache}(\mathsf{dpk}, \mathsf{m}_i||i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i||i & \text{sinon} \end{cases}$$
Le simulateur calcule ensuite $\mathsf{h}_c = \mathcal{HC}.\text{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c)$ et obtient le message intermédiaire $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1||\cdots||\tilde{\mathsf{m}}_t||\mathsf{h}_c||\mathsf{dpk}$. Il utilise enfin l'oracle de signature du challengeur sur le message $\tilde{\mathsf{m}}$ afin d'obtenir la signature σ. Il retourne à l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathbf{Imm}}$ la signature caméléon correspondant à sa requête : $\Sigma = \{\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}\}$ avec $\mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_u, \mathsf{r}_c\}$. Pour finir, le simulateur intègre la paire (m, Σ) dans une base de données, notée BD.

– $\mathcal{O}$.Prouve($\mathsf{m}, \Sigma, \mathsf{BD}$). Supposons que $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ demande une preuve sur un couple message-signature (m, Σ) avec pour entrées une clé publique de délégué dpk et une base de données de couple message signature $\mathsf{BD} = (\mathsf{m}_k, \Sigma_k)_{k=[1,\mathsf{q}]}$. Le simulateur cherche dans la base de données une entrée (m_j, Σ_j) telle que :
  – $\mathsf{vrai} = \mathcal{SC}.\text{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
  – $\mathsf{m}_j \neq \mathsf{m}$;
  – $\mathcal{HC}.\text{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\text{Hache}(\mathsf{dpk}, \mathsf{TAG}_j\|\mathsf{m}_j\|\mathsf{spk}, \mathsf{r}_{c_j})$
    avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.

  S'il trouve une telle entrée il retourne la preuve $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ avec $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j)$ et $\mathsf{r}_{c_j} \in \mathsf{R}_j$ sinon il retourne $\perp$.

A la fin de l'expérience, $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ retourne un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$. Le simulateur calcule la preuve $\pi^*_{\mathsf{or/mod}}$ correspondant à la paire (m^*, Σ^*) de la même façon que pour la simulation de l'oracle $\mathcal{SC}$.Prouve mais avec sa propre base de données BD. L'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ gagne l'expérience si :

– Σ^* est une signature caméléon valide de m^* avec la clé publique de délégation dpk^* ;
– $\mathcal{SC}.\text{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}^*, \Sigma^*, \mathsf{spk}, \mathsf{dpk}^*, \pi_{\mathsf{or/mod}}) = \mathsf{Signataire}$;
– Et si pour toutes les requêtes $k \in [1,\mathsf{n}]$ faites aux oracles $(\mathsf{dpk}^*, \mathsf{m}^*) \neq (\mathsf{dpk}_k, \mathsf{m}_k)$.

Le simulateur calcule le message intermédiaire $\tilde{\mathsf{m}}^* = \tilde{\mathsf{m}}^*_1||\cdots||\tilde{\mathsf{m}}^*_t||\mathsf{h}^*{}_c||\mathsf{dpk}^*$. Supposons que l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del-Resp}}$ a gagné son expérience, cela sous-entend que la preuve $\pi_{\mathsf{or/mod}}$ a accusé le signataire et donc que le simulateur n'a pas trouvé dans sa base de données de signature originale correspondant au message intermédiaire $\tilde{\mathsf{m}}^*$ avec la clé de délégué dpk^* et un message différent de m^*. Or pour toutes les requêtes $k \in [1,\mathsf{n}]$,

$(\mathsf{dpk}^*, \mathsf{m}^*) \neq (\mathsf{dpk}_k, \mathsf{m}_k)$. Donc le simulateur n'a jamais fait signer $\tilde{\mathsf{m}}^*$. Le simulateur gagne donc l'expérience $\mathbf{EU-CMA}$ en retournant la signature Σ^* sur le message $\tilde{\mathsf{m}}^*$.
En conclusion, la probabilité de succès d'un adversaire contre la Délégué-**Responsabilité** du schéma est :

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU}-\mathbf{CMA},\mathcal{A}}(\lambda).$$

□

Signataire-**Responsabilité**. Nous utilisons d'une part la **résistance aux collisions** de la fonction caméléon et d'autre part le fait que $\mathcal{PRG}$ soit une fonction à sens unique.
Plus formellement, nous montrons que s'il existe un adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp}}$ contre la Signataire-**Responsabilité** du schéma alors nous pouvons l'utiliser pour construire un adversaire $\mathcal{A}^{\mathcal{HC}}_{\mathbf{collRes}}$ contre la **résistance aux collisions** de la fonction caméléon ou pour construire un adversaire $\mathcal{A}^{\mathcal{PRG}}_{\mathbf{OW}}$ contre le fait que $\mathcal{PRG}$ soit une fonction à sens unique.
Au début de l'expérience, le simulateur reçoit du challengeur contre la résistance à la collision de la fonction de hachage caméléon $\mathcal{C}^{\mathcal{HC}}_{\mathbf{collRes}}$ la clé publique dpk qui sera utilisée pour les collisions de l'oracle $\mathcal{HC}$.Forge. Le simulateur fournit à l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp}}$ cette clé publique dpk comme clé publique de délégation des futures signatures caméléons. Le simulateur simule l'oracle nécessaire à $\mathcal{A}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp}}$ de la manière suivante :

– $\mathcal{O}$.Modifie$(\mathsf{m}, \Sigma, \mathsf{MOD})$. Supposons que $\mathcal{A}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp}}$ demande les modifications MOD sur la paire message-signature (m, Σ) avec pour clé publique de signataire spk. Le simulateur commence par construire le message m' en fonction des modifications demandées MOD et par mettre à jour les tailles des parties dans ADM, il choisit ensuite un Nonce' aléatoirement dans $\{0,1\}^\lambda$ puis fait un appel à l'oracle de challenge $\mathcal{PRG}$.Challenge pour obtenir TAG'. Il utilise ensuite la procédure publique de reconstruction pour obtenir le message intermédiaire $\tilde{\mathsf{m}}^* = \tilde{\mathsf{m}}^*_1||\cdots||\tilde{\mathsf{m}}^*_t||{\mathsf{h}^*}_c||\mathsf{dpk}^*$. Pour tout $j \in \mathsf{MOD}$, le simulateur utilise l'oracle $\mathcal{HC}$.Forge du challengeur $\mathcal{C}^{\mathcal{HC}}_{\mathbf{collRes}}$ pour calculer les r' : $\mathsf{r}'_j = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, \mathsf{m}_j \| j, \mathsf{m}'_j \| j, \mathsf{r}_j, \tilde{\mathsf{m}}_j)$. Enfin, il fait un dernier appel à l'oracle pour obtenir r'_c : $\mathsf{r}'_c = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, \mathsf{TAG}' \| \mathsf{m}' \| \mathsf{spk}, \mathsf{r}_c, \mathsf{h}_c)$. Il retourne à l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{del}-\mathbf{Resp}}$ la paire message-signature modifiée correspondant à sa requête : (m', Σ') avec $\Sigma' = \{\sigma, \mathsf{TAG}', \mathsf{Nonce}', \mathsf{R}', \mathsf{ADM}'\}$), R' l'ensemble des aléas utilisés et ADM' la valeur mise à jour. Enfin le simulateur conserve l'ensemble des requêtes faites aux oracles $\mathcal{PRG}$.Challenge et $\mathcal{HC}$.Forge.

A la fin de l'expérience, $\mathcal{A}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp}}$ retourne un quadruplet $(\mathsf{spk}^*, \pi^*_{\mathsf{or/mod}}, \mathsf{m}^*\Sigma^*)$. L'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp}}$ gagne l'expérience si :

– Σ^* est une signature caméléon valide sur m^* avec la clé publique de délégation dpk et la clé de signature spk^* ;
– $\mathcal{SC}.\textsc{Juge}(\mathcal{SC}.\mathsf{param}, \mathsf{m}^*, \Sigma^*, \mathsf{spk}^*, \mathsf{dpk}, \pi_{\mathsf{or/mod}}) = \mathsf{Délégué}$;
– Et si pour toutes les requêtes $k \in [1, \mathsf{n}]$ faites à l'oracle $(\mathsf{spk}^*, \mathsf{m}^*) \neq (\mathsf{spk}_k, \mathsf{m}_k)$.

Supposons que l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathsf{sig}-\mathbf{Resp}}$ ait gagné son expérience, cela sous-entend que la preuve $\pi_{\mathsf{or/mod}}$ a accusé le $\mathsf{Délégué}$ et donc que $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ est telle que :

– $\mathsf{vrai} = \mathcal{SC}.\textsc{Vérifie}(\mathcal{SC}.\mathsf{param}, \mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
  On peut donc poser $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j)$ avec $\mathsf{R}_j = \{\mathsf{r}_{1_j}, \cdots, \mathsf{r}_{u_j}, \mathsf{r}_{c_j}\}$.
– $\mathsf{m}_j \neq \mathsf{m}$;
– $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}_j \| \mathsf{m}_j \| \mathsf{spk}, \mathsf{r}_{c_j})$
  avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.

Le simulateur regarde dans sa base de données si le message $\mathsf{TAG}_j \| \mathsf{m}_j \| \mathsf{spk}^*$ appartient aux requêtes à l'oracle $\mathcal{HC}$.Forge. S'il ne le trouve pas dans sa base de données, le

simulateur gagne l'expérience contre le challengeur $\mathcal{C}^{\mathcal{HC}}_{\mathbf{collRes}}$ en retournant le quintuplet $(\mathsf{dpk}, \mathsf{TAG}_j \| \mathsf{m}_j \| \mathsf{spk}^*, \mathsf{r}_{cj}, \mathsf{TAG}^* \| \mathsf{m}^* \| \mathsf{spk}^*, \mathsf{r}_c{}^*)$.
Le simulateur regarde ensuite dans sa base de données si le TAG_j appartient aux requêtes à l'oracle $\mathcal{PRG}$.Challenge. Notons que si m_j ne fait pas partie des requêtes à l'oracle $\mathcal{HC}$.Forge alors le TAG_j fait partie des requêtes à l'oracle $\mathcal{PRG}$.Challenge car nous supposons que l'adversaire $\mathcal{A}^{\mathcal{SC}}_{\mathbf{sig-Resp}}$ a gagné son expérience. Si TAG_j est un challenge alors le simulateur gagne l'expérience contre le challengeur $\mathcal{C}^{\mathcal{PRG}}_{\mathbf{OW}}$ en retournant la paire $(\mathsf{TAG}_j, \mathsf{x}_j)$.
En conclusion, la probabilité de succès d'un adversaire contre la Signataire-**Responsabilité** du schéma est :

$$\mathsf{Succ}^{\mathcal{SC}}_{\mathbf{sig-Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{PRG}}_{\mathbf{OW},\mathcal{A}}(\lambda) + \mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda).$$

□

Conclusion

Dans ce chapitre, j'ai tout d'abord exposé en détail le modèle classique de Brzuska *et al.* pour les signatures caméléons que nous avons pris comme référence pour nos travaux. J'ai ensuite exposé mes travaux, effectués avec S.Canard, sur ce type de signatures. J'ai ainsi proposé une analyse des différentes constructions de l'état de l'art, puis j'ai introduit notre nouvelle solution, sûre et la plus efficace dans ce modèle. Ces travaux ont été publiés, avec S. Canard, à la conférence CT-RSA 2010 [CJ10a].

Nous obtenons ainsi une construction qui permet à un signataire de générer une signature de telle sorte qu'un délégué puisse modifier, comme il le souhaite, les parties que le signataire lui a désignées, et cela autant de fois qu'il le souhaite. Il est cependant intéressant, dans certains cas, de pouvoir donner au signataire plus de contrôle sur les modifications du délégué. C'est ce que nous appelons les signatures caméléons étendues.

Chapitre 7

Signatures caméléons étendues

Sommaire

Nous avons décrit les signatures caméléons au chapitre précédent. Ces signatures permettent à un signataire d'indiquer certaines parties du message qu'il signe comme modifiable par un délégué de son choix. Le signataire peut librement modifier les parties modifiables. Certaines applications nécessitent que le signataire puisse encadrer les modifications du délégué.

C'est dans cette optique qu'en 2006 M.Klonowski et A.Lauks [KL06] proposent le concept d'extension. L'idée est d'étendre le principe des signatures caméléons en permettant au signataire d'encadrer les modifications que pourra faire le délégué. Dans l'article original, trois extensions sont proposées.

ValFixées Le principe de cette extension est de permettre au signataire de fixer les modifications de certaines parties du message dans un ensemble de valeurs autorisées. Le délégué

souhaitant modifier une telle partie devra choisir sa modification dans les valeurs fixées par le signataire.

ModifEgales Avec cette extension, le signataire peut imposer qu'un ensemble de parties modifiables restent identiques entre elles. Si le délégué souhaite modifier une des parties de l'ensemble, il doit toutes les modifier de la même manière.

LimNbModif Enfin, cette extension permet au signataire de limiter le nombre de parties que le délégué pourra modifier parmi les parties modifiables. Si le délégué dépasse cette limite, alors la fraude sera détectée.

Cependant aucun modèle n'est proposé dans [KL06]. La sécurité de ces propositions n'a donc pas été étudiée. Nous verrons dans la suite que cela implique de petites erreurs de conception dans *ValFixées* et de plus gros problèmes de sécurité pour les autres.

J'ai donc proposé, avec S.Canard, dans [CJ10a], un modèle de sécurité pour ces nouvelles extensions compatible avec le modèle de Brzuska *et al.* Nous avons ensuite proposé une réparation pour *ValFixées* ainsi qu'une nouvelle solution, basée sur l'idée de [KL06] pour *LimNbModif*. Enfin, nous proposons une nouvelle extension que nous appelons *LimNbVersion*. Celle-ci permet au signataire d'imposer une limite sur le nombre de version que le délégué pourra faire. Si le délégué dépasse cette limite, alors la fraude est détectée.

7.1 Notre extension du modèle Brzuska *et al.*

7.1.1 Quelques définitions de base

Nous utilisons ici les variables ADM et MOD comme définies en section 6.2. Pour rappel, la variable ADM définit les modifications autorisées par le signataire sur une instance et MOD définit les modifications que le délégué souhaite effectuer sur un message signé.

Nous introduisons maintenant les définitions formelles des quatre extensions que nous allons étudier.

Définition 50 (*ValFixées*).
On appelle *ValFixées* l'extension permettant au signataire d'un message m de définir, s'il le souhaite, un ensemble $\mathcal{V}_i$ de valeurs autorisées pour chaque partie modifiable m_i. Lorsque les valeurs ne sont pas fixées dans un ensemble, on note $\mathcal{V}_i = \{0,1\}^{\ell_i}$ pour toute partie modifiable, $i \in \mathsf{ADM}$. Enfin, on définit V comme l'ensemble de toutes les descriptions $\mathcal{V}_i$ pour tout $i \in \mathsf{ADM}$.

$$\mathsf{V} = \{\mathcal{V}_i \subset \{0,1\}^{\ell_i}, i \in \mathsf{ADM}\}.$$

*

Exemple.
Le signataire souhaite signer le message m suivant.

$$\underbrace{\text{“Ceci}}_{1}\ \underbrace{\text{n'est pas}}_{2}\ \underbrace{\text{un chat.”}}_{3}$$

Il permet au délégué de modifier la partie m_1 comme il le souhaite, la partie m_2 uniquement dans l'ensemble $\{\text{est}, \text{n'est pas}\}$ et fixe la partie m_3 comme non modifiable.

Les ensembles de l'extension sont définis comme suit.

$$\mathcal{V}_1 = \{0,1\}^{\ell_1} \quad \mathcal{V}_2 = \{\text{est}, \text{n'est pas}\}$$

$$\mathsf{V} = \{\mathcal{V}_1, \mathcal{V}_2\}$$

•

Définition 51 (*ModifEgales*).
On appelle *ModifEgales* l'extension qui permet au signataire d'un message m de fixer un ensemble de parties modifiables comme devant rester égales entre elles. On note $\mathcal{E}_j$ le $j$ème ensemble d'indices de parties devant rester égales entre elles. On définit E comme l'ensemble des ensembles $\mathcal{E}_j$ définis pour une instance.

$$\mathsf{E} = \{\mathcal{E}_j \subset [1, \mathsf{t}] | \forall i \in \mathcal{E}_j, i \in \mathsf{ADM} \text{ et } \forall k, \mathcal{E}_k \cap \mathcal{E}_j\}.$$

∗

Exemple.
Par exemple, le signataire signe le message suivant.

$$\underbrace{\text{“Tous les hommes sont}}_{1}\ \underbrace{\text{mortel}}_{2}\ \underbrace{\text{s.}}_{3}$$

$$\underbrace{\text{Or}}_{4}\ \underbrace{\text{Socrate}}_{5}\ \underbrace{\text{est un homme.}}_{6}$$

$$\underbrace{\text{Donc}}_{7}\ \underbrace{\text{Socrate}}_{8}\ \underbrace{\text{est}}_{9}\ \underbrace{\text{mortel}}_{10}\ \underbrace{\text{.”}}_{11}$$

Il permet au délégué de modifier les parties $2, 5, 8$ et 10 mais en exigeant que les parties 2 et 10 restent égales entre elles et d'autre part que les parties 5 et 8 soient identiques.

Les ensembles de l'extension sont définis comme suit.

$$\mathcal{E}_1 = \{2, 10\} \quad \mathcal{E}_2 = \{5, 8\}$$

$$\mathsf{E} = \{\mathcal{E}_1, \mathcal{E}_2\}.$$

•

Pour les deux extensions suivantes nous introduisons la notion de clé secrète d'utilisateur pour le délégué. Celui-ci aura, en plus d'une paire de clés de délégués, une paire de clés $(\mathsf{usk}, \mathsf{upk})$. Nous considérons par la suite que upk est inclue dans la clé publique de délégué dpk. Contrairement aux premières extensions, les deux extensions que nous allons voir maintenant n'empêchent pas le délégué de faire une action mais si celui-ci agit inconsidérément une fraude sera détectée et la clé secrète d'utilisateur usk pourra être publiquement retrouvée.

Définition 52 (*LimNbModif*).
On appelle *LimNbModif* l'extension qui permet au signataire d'un message m de fixer un maximum maxMod au nombre de parties modifiables par le délégué parmi les $|\mathsf{ADM}|$ parties modifiables. Si le délégué modifie plus de maxMod parties, la clé secrète usk devient publiquement accessible. Si le signataire choisit de ne pas utiliser cette extension, alors $\mathsf{maxMod} = |\mathsf{ADM}|$. ∗

Définition 53 (*LimNbVersion*).
On appelle *LimNbVersion* l'extension qui permet au signataire d'un message m de fixer un maximum maxV au nombre de versions autorisées par le signataire. Si le délégué génère plus de maxV versions, la clé secrète usk devient publiquement accessible. Si le signataire choisit de ne pas utiliser cette extension, alors $\mathsf{maxV} = \infty$. ∗

Remarque (ADM). Il serait tentant d'inclure les variables d'extensions V, E, maxMod et maxV dans la variable ADM. Cependant ADM est publique. Cette solution n'est donc possible que dans le cas où l'on accepte que toutes ces variables soient publiques elles aussi. Nous considérerons dans notre modèle que ces variables ne font pas parties d'ADM afin de pouvoir conserver la même modélisation que les variables soient publiques ou privées.

Définition 54 (MOD *correspond aux limitations*).
MOD *correspond aux limitations* si, en considérant que MOD décrit les modifications pour la $j$ème version modifiée, on a :
- $\forall i \in \mathsf{MOD}, \mathsf{m}'_i \in \mathcal{V}_i$,
- $\forall i \in \mathsf{MOD}$, si $\exists k$ tel que $i \in \mathcal{E}_k$ alors $\forall l \in \mathcal{E}_k$, $l \in \mathsf{MOD}$ et $\mathsf{m}'_l = \mathsf{m}'_i$,
- $\|\{i | \exists k \in [1, j], i \in \mathsf{MOD}_k\}\| \leq \mathsf{maxMod}$ avec MOD_k la variable MOD de la $k$ème version,
- et $j \leq \mathsf{maxV}$. *

7.1.2 Définition d'une signature caméléon étendue

Un schéma de signature caméléon étendu est un schéma de signature caméléon permettant au signataire d'encadrer précisément les modifications que le délégué peut effectuer sur ces signatures. Plus formellement, nous définissons un tel schéma comme suit.

Définition 55 (Signature Caméléon étendue).
Un schéma de signature caméléon étendu $\mathcal{SCE}$ est un schéma de signature défini par les neuf algorithmes suivants. Les variables $\mathsf{V}, \mathsf{E}, \mathsf{maxMod}$ et maxV pouvant être publiques ou secrètes, nous décrivons dans cette définition le cas le moins favorable où elles sont secrètes (c'est-à-dire connues uniquement du signataire et du délégué).

- $\mathcal{SCE}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système comprenant la description du groupe et les paramètres d'initialisation des schémas utilisés. On notera ces paramètres $\mathcal{SCE}.\mathsf{param}$. Nous considérerons dans la suite que λ est inclus dans les paramètres.

$$\mathcal{SCE}.\mathsf{param} \leftarrow \mathcal{SCE}.\textsc{Init}(1^\lambda)$$

- $\mathcal{SCE}$.SigGénClé génère la paire de clés du signataire $(\mathsf{spk}, \mathsf{ssk})$ en fonction des paramètres du système $\mathcal{SCE}.\mathsf{param}$.

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SCE}.\textsc{SigGénClé}(\mathcal{SCE}.\mathsf{param})$$

- $\mathcal{SCE}$.SanGénClé génère la paire de clés du délégué $(\mathsf{dpk}, \mathsf{dsk})$ et la clé secrète d'utilisateur usk en fonction des paramètres du système $\mathcal{SCE}.\mathsf{param}$

$$(\mathsf{dpk}, \mathsf{dsk}, \mathsf{usk}) \leftarrow \mathcal{SCE}.\textsc{SanGénClé}(\mathcal{SCE}.\mathsf{param})$$

- $\mathcal{SCE}$.Signe est l'algorithme de signature. Il prend en entrées un message m de taille ℓ qui est composé de t parties, la clé secrète du signataire ssk, la clé publique du délégué dpk et une variable ADM indiquant les parties qui pourront être modifiées par le délégué ainsi que la taille des différentes parties du message (cf. Définition 43). Enfin, il prend en entrées les variables d'extension V, E, maxMod et maxV. L'algorithme retourne une signature Σ sur le message m (ou $\perp$ en cas d'erreur). De plus le schéma peut retourner un élément secret s pour le délégué. Si cet élément n'est pas nécessaire dans le schéma, nous noterons $\mathsf{s} = \perp$. Nous considérons dans la suite que la variable ADM est incluse dans la signature Σ.

$$(\Sigma, \mathsf{s}) \leftarrow \mathcal{SCE}.\textsc{Signe}(\mathcal{SCE}.\mathsf{param}, \mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$$

– $\mathcal{SCE}$.Modifie utilise un message m et sa signature caméléon Σ, la clé publique du signataire spk, la clé secrète du délégué dsk, une variable MOD décrivant les modifications voulues sur le message m (cf. Définition 44), l'élément secret s ainsi que les variables V, E, maxMod et maxV fournies par le signataire. L'algorithme retourne une nouvelle signature Σ' sur le message modifié m' en fonction des variables V, E, maxMod et maxV définies par le signataire (ou $\perp$ en cas d'erreur).

$$(\Sigma', \mathsf{m}') \leftarrow \mathcal{SCE}.\textsc{Modifie}(\mathcal{SCE}.\mathsf{param}, \mathsf{m}, \mathsf{s}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV}, \mathsf{s})$$

– $\mathcal{SCE}$.Vérifie permet de vérifier une signature Σ sur un message m en utilisant les clés publiques du signataire spk et du délégué dpk. Il retourne vrai si la signature est valide et faux si elle ne l'est pas.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SCE}.\textsc{Vérifie}(\mathcal{SCE}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk})$$

– $\mathcal{SCE}$.Prouve prend en entrées une signature Σ sur un message m, la clé secrète du signataire ssk, la clé publique du délégué dpk ainsi qu'un ensemble de paires message-signature originales ayant été produit avec ces clés $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k=[1,\mathsf{q}]}$. Il retourne une preuve $\pi_{\mathsf{or/mod}}$ sur la signature Σ du message m.

$$\pi_{\mathsf{or/mod}} \leftarrow \mathcal{SCE}.\textsc{Prouve}(\mathcal{SCE}.\mathsf{param}, \mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, \mathsf{BD})$$

– $\mathcal{SCE}$.Juge est un algorithme public permettant de vérifier, à l'aide d'une preuve générée par le signataire, si une paire message-signature est originale ou non. Pour cela, il prend en entrées la paire concernée (m, Σ), une preuve $\pi_{\mathsf{or/mod}}$ provenant de l'algorithme $\mathcal{SCE}$.Prouve ainsi que les clés publiques du signataire spk et du délégué dpk. L'algorithme retourne l'origine du couple (m, Σ), c'est à dire Signataire si le couple est un original ou Délégué s'il s'agit d'une paire modifiée.

$$\{\mathsf{Signataire}, \mathsf{Délégué}\} \leftarrow \mathcal{SCE}.\textsc{Juge}(\mathcal{SCE}.\mathsf{param}, \mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$$

– $\mathcal{SCE}$.TestFraude est un algorithme public qui prend en entrées la clé publique du signataire spk, la clé publique du délégué dpk ainsi qu'un ensemble de paires message-signature d'une même instance que nous noterons pubDB. Il vérifie si le délégué a fraudé sur le nombre de versions ou sur le nombre de parties modifiées. En cas de fraude, il retourne la clé secrète d'utilisateur du délégué usk ainsi qu'une preuve de sa culpabilité π_{fraude} ou $\perp$ s'il n'y a pas de fraude.

$$(\mathsf{usk}, \pi_{\mathsf{fraude}}) \leftarrow \mathcal{SCE}.\textsc{TestFraude}(\mathcal{SCE}.\mathsf{param}, \mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB})$$

– $\mathcal{SCE}$.VérifFraude est un algorithme public qui vérifie la validité d'une preuve de culpabilité en cas de fraude. Il prend en entrée une preuve π_{fraude} et une clé secrète d'utilisateur usk. Il retourne vrai si la preuve de la culpabilité de l'utilisateur liée à la clé usk est valide et faux sinon.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SCE}.\textsc{VérifFraude}(\mathcal{SCE}.\mathsf{param}, \pi_{\mathsf{fraude}}, \mathsf{usk})$$

*

7.1.3 Propriétés de sécurité

Un schéma de signature caméléon étendu doit être, avant tout, un schéma de signature caméléon sûr et donc respecter les trois propriétés de base que nous rappelons brièvement.

Immuabilité. Il doit être impossible, même pour le délégué, de modifier une partie du message désignée comme non-modifiable par le signataire tout en gardant une signature caméléon valide de celui-ci.

Transparence. Seuls le signataire et le délégué doivent être capables de distinguer une paire message-signature originale d'une paire modifiée.

Responsabilité. En cas de problème sur l'origine d'une signature, le Juge doit être capable d'arbitrer correctement qui est à l'origine de la signature. Cette propriété se décompose en deux sous-propriétés.

- Délégué-**Responsabilité**. Si un message n'a pas été signé par le signataire, alors même un délégué corrompu doit être incapable de faire accuser le signataire par le Juge.
- Signataire-**Responsabilité**. Si un message et sa signature n'ont pas été modifiés par le délégué, alors même un signataire corrompu ne doit pas pouvoir faire accuser le délégué par le Juge.

Les spécificités qu'apportent nos extensions vont nous amener à adapter les expériences correspondant à certaines de ces propriétés et nous montrerons que ces changements n'impliquent pas de biais dans le modèle. D'autre part, nous devrons introduire deux nouvelles propriétés afin de nous assurer que les schémas étendues atteignent bien leurs objectifs.

Immuabilité étendue. Il doit être impossible, notamment pour le délégué, de modifier une partie du message désignée comme non-modifiable, d'utiliser une valeur non autorisée pour une partie modifiable ou de ne pas modifier de la même manière deux parties fixées aux même valeurs tout en gardant une signature caméléon étendue valide du signataire sur le message obtenu.

Traçabilité. Le délégué doit être incapable de construire plus de versions d'un message ou de modifier plus de parties d'un message original qu'autorisé par le signataire sans être accusé de fraude.

Nous allons d'abord voir les adaptations nécessaires sur les propriétés existantes avant de donner les définitions formelles de ces deux nouvelles propriétés.

Adaptation des propriétés existantes

La première modification, et la seule en ce qui concerne les deux expériences de **Responsabilité**, consiste en l'adaptation des oracles au modèle. Plus précisément, nous avons les changements suivants :

- L'oracle de signature $\mathcal{O}.\textsc{Signe}$ doit aussi prendre en entrée les conditions de limitations et retourner, en plus de la signature, la variable s.

$$(\Sigma, \mathsf{s}) \leftarrow \mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$$

- L'oracle de modification $\mathcal{O}.\textsc{Modifie}$ a comme entrées supplémentaires les définitions des limitations et la variable s.

$$(\Sigma', \mathsf{m}') \leftarrow \mathcal{O}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV}, \mathsf{s})$$

- L'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b$ prend en entrées un message m, ADM et MOD comme dans le cas classique, plus les définitions des extensions $\mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV}$. Dans notre modèle,

l'oracle fonctionne de manière légèrement différente, dans le sens où l'oracle doit faire en sorte que quelque soit la valeur du bit b autant de versions de message-signature ont été générées par l'oracle.
On note m' le message correspondant au message m modifié selon MOD.
Si $b = 0$, l'oracle génère une signature originale du signataire sur m' puis modifie celle-ci en fonction d'un MOD' aléatoire comme le ferait le délégué. Enfin il retourne la signature originale.
Si $b = 1$, il effectue une signature originale du signataire sur le message m selon ADM puis modifie celle-ci en fonction de MOD comme le ferait le délégué. Enfin, il retourne cette signature modifiée sur le message m'.

$$\Sigma \leftarrow \mathcal{O}.\textsc{Modif/Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$$

Remarque. Nous exigeons que les oracles $\mathcal{O}.\textsc{Signe}$ et $\mathcal{O}.\textsc{Modif/Signe}_b$ collaborent afin qu'ils ne dépassent pas, à eux deux, le nombre de versions autorisées. Nous supposons qu'ils partagent un compteur commun du nombre de versions délivrées.

– L'oracle $\mathcal{O}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{BD})$ est identique à l'oracle du modèle général. Il simule l'algorithme de preuve d'origine.

Modifications pour garder les implications de propriétés du modèle [BFF+09]. Afin de conserver les implications prouvées par Brzuska *et al.* dans leur modèle, les modifications d'oracle doivent être appliquées à toutes les propriétés.

L'**Intimité**, c'est-à-dire l'impossibilité pour un adversaire de retrouver à partir d'un message modifié le message original, est incluse dans la **Transparence** dans le modèle initial. Cette propriété doit subir les modifications suivantes afin d'être inclue dans la **Transparence** de notre modèle étendu.

Dans [BFF+09], l'adversaire a accès à quatre oracles durant l'expérience d'**Intimité**.

– Le premier est un oracle de signature qui correspond, dans le modèle étendu, à l'oracle

$$\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV}).$$

– Le second simule l'algorithme de modification, dans le modèle étendu on utilise

$$\mathcal{O}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV}, \mathsf{s}).$$

– Le troisième oracle simule l'algorithme de preuve et correspond à

$$\mathcal{O}.\textsc{Modif/Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV}).$$

– Enfin, le dernier oracle est spécifique à l'expérience d'**Intimité**.
Cet oracle, noté $\mathcal{O}.\textsc{LoRSanit}_b$, prend en entrée une variable ADM, deux message m_0, m_1 et deux variables de modification MOD_0, MOD_1 tel que le message m_0 modifié selon MOD_0 soit identique au message m_1 modifié selon MOD_1. Dans le modèle étendu, on doit ajouter aux entrées les définitions des extensions $\mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV}$ et exiger que les deux variables MOD_0 et MOD_1 correspondent aux limitations.
Enfin, comme dans le modèle initial, l'oracle retournera soit une signature du message m_0 modifiée selon MOD_0 si $b = 0$ soit une signature du message m_1 modifié selon MOD_1 si $b = 1$. L'adversaire gagne l'expérience s'il retourne un bit b' égal à b.

$$\mathcal{O}.\textsc{LoRSanit}_b(\mathsf{ADM}, \mathsf{m}_0, \mathsf{m}_1, \mathsf{MOD}_0, \mathsf{MOD}_1)$$

Les nouvelles propriétés.

Nous allons à présent décrire les deux nouvelles propriétés l'**Immuabilité étendue** et la **Traçabilité**. Pour cela nous utilisons les même oracles que ceux que nous venons de décrire pour les propriétés existantes. Pour des raisons de lisibilité des expériences, les entrées choisies par l'adversaire seront omises dans les oracles, de même le $\mathcal{O}.$ sera sous-entendu lorsqu'il est évident. Par exemple, l'oracle $\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$ pourra abusivement être noté Signe dans les expériences.

Définition 56 (Immuabilité étendue).
Soient λ un paramètre de sécurité et $\mathcal{SCE}$ un schéma de signature caméléon étendu comprenant $\mathcal{SCE}.\textsc{Init}$ l'algorithme d'initialisation et $\mathcal{SCE}.\textsc{SigGénClé}$ l'algorithme de génération de clés de signataire $(\mathsf{spk}, \mathsf{ssk})$. Soit $\mathcal{A}$ un attaquant ayant accès à accès à deux oracles : l'oracle de signature $\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$ et l'oracle de preuve $\mathcal{O}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{BD})$ et renvoyant un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$. Nous définissons l'expérience contre l'**Immuabilité étendue** comme suit :

$\textsc{Exp}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda)$:

$\mathcal{SCE}.\mathsf{param} \leftarrow \mathcal{SCE}.\textsc{Init}(1^\lambda)$

$(\mathsf{spk}, \mathsf{ssk}) \longleftarrow \mathcal{SCE}.\textsc{SigGénClé}(\mathcal{SCE}.\mathsf{param})$

$(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*) \longleftarrow \mathcal{A}^{\mathcal{O}.\textsc{Signe},\mathcal{O}.\textsc{Prouve}}(\mathsf{spk}, \mathcal{SCE}.\mathsf{param})$

Soit $\{(\mathsf{m}_k, \mathsf{ADM}_k, \mathsf{dpk}_k, \mathsf{V}_k, \mathsf{E}_k, \mathsf{maxMod}_k, \mathsf{maxV}_k)\}_{k\in[1,\mathsf{n}]}$ les requêtes à l'oracle $\mathcal{O}.\textsc{Signe}$ et $\{(\Sigma_k, \mathsf{s}_k,)\}_{k\in[1,\mathsf{n}]}$ les réponses correspondantes.

Retourne 1 si $\mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}^*, \Sigma^*, \mathsf{spk}, \mathsf{dpk}^*) = \mathsf{vrai}$ et :
– pour tout $k \in [1, \mathsf{n}]$,
  $\mathsf{dpk}^* \neq \mathsf{dpk}_k$ ou
  $\exists j \notin \mathsf{ADM}_k$ tel que $\mathsf{m}^*[j] \neq \mathsf{m}_k[j]$.
– ou $\exists j$ tel que $\mathsf{m}^*_j \notin \mathcal{V}^*_j$
– ou $\exists l$ tel que $\exists j, j' \in \mathcal{E}_l$ tels que $\mathsf{m}^*[j] \neq \mathsf{m}^*_[j']$
Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\textsc{Exp}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \textsc{Exp}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda)].$$

Un schéma de signature caméléon $\mathcal{SCE}$ est dit Et-**Immuable** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .}$$

*

Lemme 12. *Un schéma de signature caméléon Et-**Immuable** est **Immuable**.* ♢

Démonstration. Pour montrer le lemme 12, nous nous référerons à l'expérience d'**Immuabilité** d'un schéma de signature caméléon décrite en Définition 46. Il suffit alors de remarquer que

les critères pour gagner l'expérience d'**Immuabilité** correspondent à un sous-ensemble des critères pour gagner l'expérience d'**Immuabilité étendue**. Donc s'il existe un adversaire $\mathcal{A}^{\mathcal{SCE}}_{\mathbf{Imm}}$ contre l'expérience d'**Immuabilité** d'un schéma alors nous pouvons l'utiliser pour construire un adversaire $\mathcal{A}^{\mathcal{SCE}}_{Et-\mathbf{Imm}}$ contre l'**Immuabilité étendue** de ce même schéma. □

Définition 57 (Traçabilité).
Soient λ un paramètre de sécurité et $\mathcal{SCE}$ un schéma de signature caméléon étendu comprenant $\mathcal{SCE}$.Init l'algorithme d'initialisation et $\mathcal{SCE}$.SigGénClé l'algorithme de génération de clés de signataire $(\mathsf{spk}, \mathsf{ssk})$. Soit $\mathcal{A}$ un attaquant ayant accès à deux oracles : l'oracle de signature $\mathcal{O}$.Signe$(\mathsf{m}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$ et l'oracle de preuve $\mathcal{O}$.Prouve$(\mathsf{m}, \Sigma, \mathsf{BD})$, il renvoie une paire $(\mathsf{dpk}^*, \mathsf{pubDB}^* = \{(\mathsf{m}^*_k, \Sigma^*_k)\}_{k\in[1,n]})$. Nous définissons l'expérience contre la **Traçabilité** comme suit.

$\text{Exp}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda)$:
$\mathcal{SCE}.\mathsf{param} \leftarrow \mathcal{SCE}.\text{Init}(1^\lambda)$
$(\mathsf{spk}, \mathsf{ssk}) \longleftarrow \mathcal{SCE}.\text{SigGénClé}(\mathcal{SCE}.\mathsf{param})$
$(\mathsf{dpk}^*, \mathsf{pubDB}^*) \longleftarrow \mathcal{A}^{\mathcal{O}.\text{Signe},\mathcal{O}.\text{Prouve}}(\mathsf{spk})$
avec $\mathsf{pubDB}^* = \{(\mathsf{m}_k{}^*, \Sigma_k{}^*)\}_{k\in[1,\mathsf{n}]}$
Retourne 0 si $\exists j$ tel que $\mathcal{SCE}.\text{Vérifie}(\mathsf{m}_j{}^*, \Sigma_j{}^*, \mathsf{spk}, \mathsf{dpk}^*) = \mathsf{faux}$.
$(\mathsf{usk}, \pi_{\mathsf{fraude}}) \longleftarrow \mathcal{SCE}.\text{TestFraude}(\mathsf{spk}, \mathsf{dpk}^*, \mathsf{pubDB}^*)$
Retourne 1 si :
– usk ne correspond pas a dpk
– ou $\mathcal{SCE}.\text{VérifFraude}(\pi_{\mathsf{fraude}}, \mathsf{usk}) = \mathsf{faux}$
Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda)].$$

Un schéma de signature caméléon $\mathcal{SCE}$ est dit **Traçable** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .}$$

*

Le modèle étant posé, nous allons à présent nous intéresser aux constructions pour ces extensions. Pour chaque extension, nous étudierons d'abord la proposition de M.Klonowski et A.Lauks [KL06] afin d'en voir les limites. Ensuite, nous décrirons, pour trois d'entre elles, les solutions que j'ai proposées avec S.Canard dans [CJ10a]. Pour plus de lisibilité, nous omettrons pour chaque extension les variables et algorithmes non pertinents.

Notamment, les extensions *ValFixées* et *ModifEgales* n'utilisant pas de mécanisme de détection de fraude, nous ne décrirons pas de construction pour les algorithmes $\mathcal{SCE}$.TestFraude et $\mathcal{SCE}$.VérifFraude dans ces deux cas. De plus, nous omettrons la clé d'utilisateur du délégué usk qui correspond, lorsqu'un mécanisme de fraude est utilisé, au secret dévoilé publiquement en cas de fraude. Pour ces extensions la propriété de **Traçabilité** n'a pas de sens. Nous ne

prouverons donc pas cette propriété pour ces deux extensions car elle est triviale : il n'y a pas de fraude possible, vu que le principe de fraude n'est pas utilisé.

D'autre part, nous nous concentrerons sur chaque extension, une par une, et ne citerons à chaque fois que les variables d'extensions concernant la propriété étudiée, c'est-à-dire V pour *ValFixées*, E pour *ModifEgales*, maxMod pour *LimNbModif* et maxV pour *LimNbVersion*. Nous verrons en conclusion que certaines constructions sont compatibles entre-elles mais que cela n'est pas systématique.

7.2 Constructions pour l'extension *ValFixées*

Nous allons d'abord décrire la construction proposée par M.Klonowski et A.Lauks pour cette extension, avant de proposer une amélioration de celle-ci permettant d'obtenir toutes les propriétés de notre modèle.

Pour plus de clarté, nous notons ADM^+ le sous-ensemble des parties désignées comme modifiables qui sont limitées dans un ensemble de valeurs autorisées et ADM^- le sous-ensemble des parties modifiables qui n'ont pas cette restriction. On a $\mathsf{ADM}^+ \cap \mathsf{ADM}^- = \emptyset$ et $\mathsf{ADM}^+ \cup \mathsf{ADM}^- = \mathsf{ADM}$.

De même, on note MOD^+ le sous-ensemble des parties que le délégué souhaite modifier et qui sont limitées dans un ensemble de valeurs autorisées et MOD^- le sous ensemble des parties qu'il souhaite modifier mais qui n'ont pas cette restriction. On a $\mathsf{MOD}^+ \cap \mathsf{MOD}^- = \emptyset$ et $\mathsf{MOD}^+ \cup \mathsf{MOD}^- = \mathsf{MOD}$.

On rappelle que le message m est découpé de la manière suivante : $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_\mathsf{t}$.

7.2.1 La solution de M.Klonowski et A.Lauks

L'idée de ce schéma est de remplacer dans le message intermédiaire le condensé caméléon sur les parties modifiables dans un ensemble par des accumulateurs sur les valeurs autorisées. Les parties du message intermédiaire correspondent alors soit à la partie elle-même si elle n'est pas modifiable, soit au condensé caméléon de la partie si elle est modifiable librement soit à un accumulateur sur les valeurs autorisées pour la partie. Enfin, le signataire signe le message intermédiaire et ajoute à sa signature les aléas qui correspondent aux condensés des parties, ainsi que les témoins des valeurs utilisées pour les parties du message fixées dans un ensemble. Si le délégué souhaite modifier la valeur d'une partie fixée, il lui suffit de remplacer le témoin existant dans la signature qu'il modifie par celui correspondant à la valeur qu'il souhaite pour la partie.

De manière plus formelle, la construction est la suivante.

Construction 20.

Ce schéma utilise un schéma d'accumulateur $\mathcal{ACC}$ **résistant aux collisions** (voir Définition 27), un schéma de fonction de hachage caméléon $\mathcal{HC}$ **uniforme** et **résistant aux collisions** (voir Définition 22) ainsi qu'un schéma de signature **EU-CMA** sûr, noté $\mathcal{S}$ (voir Définition 29).

$\mathcal{SCE}.\textsc{Init}(1^\lambda)$. Cette étape initialise le schéma de signature $\mathcal{S}.\mathsf{param} = \mathcal{S}.\textsc{Init}(\lambda)$ et le schéma de fonction de hachage caméléon $\mathcal{HC}.\mathsf{param} = \mathcal{HC}.\textsc{Init}(\lambda)$ en fonction du paramètre de sécurité λ.

$$\mathcal{SCE}.\mathsf{param} = \{\mathcal{S}.\mathsf{param}, \mathcal{HC}.\mathsf{param}\} \leftarrow \mathcal{SCE}.\textsc{Init}(1^\lambda).$$

$\mathcal{SCE}.\textsc{SigGénClé}(\mathcal{SCE}.\mathsf{param})$. La paire de clés du signataire correspond à une paire de clés de signataire du schéma de signature choisi $(\mathsf{spk}, \mathsf{ssk}) = \mathcal{S}.\textsc{GénClé}(\mathcal{S}.\mathsf{param})$.

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SCE}.\textsc{SigGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.SanGénClé($\mathcal{SCE}$.param). La paire de clés du délégué correspond à une paire de clés d'une fonction caméléon sûre $(\mathsf{dsk}, \mathsf{dpk}) = CH.\text{KeyGen}(\mathcal{HC}.\mathsf{param})$.

$$(\mathsf{dpk}, \mathsf{dsk}) \leftarrow \mathcal{SCE}.\text{SanGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.Signe($\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}$). Le signataire choisit aléatoirement dans $\{0,1\}^\lambda$ un identifiant unique Id_m ainsi qu'un r_i pour chaque $i \in ADM^-$, on note $\mathsf{R} = \{\mathsf{r}_i, \forall i \in \mathsf{ADM}^-\}$. Il construit ensuite un accumulateur contenant les valeurs autorisées pour chaque partie limitée dans un ensemble. Pour chaque partie m_i de ADM^+, le signataire génère un accumulateur vide Acc_{0i} pouvant accueillir toutes les valeurs autorisées, soit $\|\mathsf{V}_i\|$ valeurs. Il ajoute ensuite les valeurs de V_i et obtient l'accumulateur Acc_i et l'ensemble des témoins $\mathcal{T}_i$ correspondant aux valeurs accumulées. De manière plus formelle, le signataire construit les accumulateurs de la manière suivante.
$\forall i \in \mathsf{ADM}^+$:

- $((\mathsf{ask}_i, \mathsf{apk}_i), \mathsf{Acc}_{0i}, \mathsf{etat}_{0i}, \mathsf{aux}_i) \leftarrow \mathcal{ACC}.\text{Gén}(\lambda, \|\mathsf{V}_i\|)$
- $(\mathsf{Acc}_i, \mathcal{T}_i, \mathsf{etat}_i) \leftarrow \mathcal{ACC}.\text{AjouteEns}(\mathsf{ask}_i, \mathsf{Acc}_0, \mathsf{V}_i, \mathsf{etat}_0, \mathsf{aux}_i)$

Le signataire calcule ensuite le message reconstitué de la façon suivante.

Procédure de reconstruction :
Cette procédure publique utilise les aléas R, les accumulateurs $\{\mathsf{Acc}_i\}_{i\in\mathsf{ADM}^+}$, la clé publique du délégué dpk et celle du signataire spk :

Pour chaque partie, on calcule un message reconstitué $\tilde{\mathsf{m}}_i$.

$$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{Acc}_i & \text{si } \mathsf{m}_i \in \mathsf{ADM}^+ \\ \mathsf{h}_i = \mathcal{HC}.\text{Hache}(\mathsf{dpk}, Id_\mathsf{m} \| i \| \mathsf{m}_i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM}^- \\ \mathsf{m}_i \| i & \text{sinon} \end{cases} .$$

Le message reconstitué est $\tilde{\mathsf{m}} = Id_\mathsf{m} \| \mathsf{t} \| \mathsf{dpk} \| \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_t$.

Le signataire signe le message reconstitué $\sigma = \mathcal{S}.\text{Signe}(\mathsf{spk}, \mathsf{ssk}, \tilde{\mathsf{m}})$. La signature caméléon sur le message m est $\Sigma = (\sigma, \mathsf{R}, \mathsf{T}, \mathsf{ADM}, Id_\mathsf{m}, \mathsf{dpk}, \{\mathsf{apk}_i\}_{i\in\mathsf{ADM}^+})$ avec $\mathsf{R} = \{\mathsf{r}_i | i \in \mathsf{ADM}^-\}$ et $\mathsf{T} = \{\mathsf{t}_i \text{ tel que } \mathsf{v}_i = \mathsf{m}_i\}_{i\in\mathsf{ADM}^+}$. Le signataire ajoute la signature et le message dans sa base de données BD. Enfin il envoie $\mathsf{s} = \{(\mathcal{T}_i, \mathsf{V}_i)\}_{i\in\mathsf{ADM}^+}$ au signataire.

$$(\Sigma = (\sigma, \mathsf{R}, \mathsf{T}, \mathsf{ADM}, Id_\mathsf{m}, \mathsf{dpk}, \{\mathsf{apk}_i\}_{i\in\mathsf{ADM}^+}), \mathsf{s} \leftarrow \mathcal{SCE}.\text{Signe}(\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V})$$

$\mathcal{SCE}$.Modifie($\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \mathsf{V}, \mathsf{s}$). Le délégué souhaite à présent transformer le message m en le message m'. Pour cela il décompose $\mathsf{s} = \{(\mathcal{T}_i, \mathsf{V}_i)\}_{i\in\mathsf{ADM}^+}$ puis utilise la *Procédure de modification* suivante.

Procédure de modification :
Cette procédure publique utilise les aléas R, les valeurs des parties reconstitués $\tilde{\mathsf{m}}_i$ obtenues par la *Procédure de reconstruction*, l'ensemble des témoins $\mathcal{T}_i$, la clé secrète du délégué dsk :

Pour tout $i \in \mathsf{MOD}^+$, le délégué remplace le témoin t_i original du message original m_i par le témoin, noté t'_i, correspondant à la valeur m'_i qu'il souhaite.

Pour tout $i \in \mathsf{MOD}^-$, il calcule

$$\mathsf{r}'_i = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, Id_{\mathsf{m}} \| i \| \mathsf{m}_i, Id_{\mathsf{m}} \| i \| \mathsf{m}'_i, \mathsf{r}_i, \tilde{\mathsf{m}}_i).$$

La délégué réactualise les aléas et les témoins : $\mathsf{R}' = \{\mathsf{r}'_i, i \in \mathsf{ADM}\}$ avec $\mathsf{r}'_i = \mathsf{r}_i$ si m_i n'a pas été modifié, et $\mathsf{T}' = \{\mathsf{t}'_i, i \in \mathsf{ADM}\}$ en considérant que $\mathsf{t}'_i = \mathsf{t}_i$ si m_i n'a pas été modifié.

La nouvelle signature est $\Sigma' = (\sigma, \mathsf{R}', \mathsf{T}', \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk}, \{\mathsf{apk}_i\}_{i \in \mathsf{ADM}^+})$.

$$(\Sigma', \mathsf{m}') \leftarrow \mathcal{SCE}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \mathsf{V}, \mathsf{s})$$

$\mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk})$. Le vérifieur reconstruit le message reconstitué en utilisant la *Procédure de reconstruction* et obtient $\tilde{\mathsf{m}}$. Il vérifie ensuite la signature à l'aide de la procédure $\mathcal{S}.\textsc{Vérifie}(\mathsf{spk}, \tilde{\mathsf{m}}, \sigma)$. Il retourne vrai si la signature est valide et faux si elle ne l'est pas.

$\mathcal{SCE}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, \mathsf{BD})$. En cas de problème, le signataire prouve qu'il n'a pas généré une paire valide (m, Σ) où $\Sigma = (\sigma, \mathsf{R}, \mathsf{T}, \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk}, \{\mathsf{apk}_i\}_{i \in \mathsf{ADM}^+}$ en divulguant au Juge le message original et sa signature sur celui-ci. Pour cela il cherche dans sa base de données $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k \in [1,\mathsf{q}]}$ une paire (m_x, Σ_x) telle que les *équations de vérification* soient vérifiées.

Équations de vérifications :

1. $Id_{\mathsf{m}_x} = Id_{\mathsf{m}}$;
2. $\forall j \in [1, \mathsf{t}]$,
$$\begin{cases} \mathcal{ACC}.\textsc{Teste}(\mathsf{v}_{xj}, \mathsf{t}_{xj}) = \mathcal{ACC}.\textsc{Teste}(\mathsf{v}_j, \mathsf{t}_j) & \text{si } \mathsf{m}_j \in \mathsf{ADM}^+ \\ \mathsf{h}_{x_j} = \mathsf{h}_j & \text{si } \mathsf{m}_j \in \mathsf{ADM}^- \\ \mathsf{m}_{xj} = \mathsf{m}_j & \text{sinon} \end{cases} ;$$
3. et $\sigma_x = \sigma$.

Si une telle paire existe alors $\pi_{\mathsf{or/mod}} = (\mathsf{m}_x, \Sigma_x)$ sinon $\pi_{\mathsf{or/mod}} = \perp$.

$\mathcal{SCE}.\textsc{Juge}(\mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$. Si la preuve contient $\perp$ alors le juge retourne $\mathsf{Signataire}$, sinon il vérifie les *équations de vérification* entre la paire m, Σ et la paire contenue dans la preuve $\pi_{\mathsf{or/mod}}$. Si les équations sont toutes vraies, il retourne $\mathsf{Délégué}$ sinon il retourne $\mathsf{Signataire}$.

Analyse de la solution [KL06].

Il existe une attaque contre ce schéma qui permet, à partir d'une combinaison de messages d'une même instance, de générer une signature modifiée valide sur un nouveau message sans

connaître la clé du signataire. Cette attaque est similaire à l'attaque contre [ACdT05] que nous avons vu au chapitre 6 section 6.3.3. La différence ici est qu'elle peut être appliquée non seulement pour les parties modifiables librement mais aussi pour les parties limitées dans un ensemble. L'attaque fonctionne de la manière suivante.

> Supposons qu'un attaquant connaisse deux messages signés $\mathsf{m}_a = \mathsf{m}_1 \| \mathsf{m}_{a2} \| \mathsf{m}_3 \| \mathsf{m}_{a4}$ et $\mathsf{m}_b = \mathsf{m}_1 \| \mathsf{m}_{b2} \| \mathsf{m}_3 \| \mathsf{m}_{b4}$ d'une même instance tels que les parties 2 et 4 soient modifiables et différentes ($\mathsf{m}_{a2} \neq \mathsf{m}_{b2}$ et $\mathsf{m}_{a4} \neq \mathsf{m}_{b4}$). Pour l'exemple, on suppose que la partie m_2 et modifiable dans un ensemble. L'attaquant connait donc les signatures correspondantes : $\Sigma_a = (\sigma, \{\mathsf{r}_{a_4}\}, \{\mathsf{t}a_2\}, \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk})$ et $\Sigma_b = (\sigma, \{\mathsf{r}_{b4}\}, \{\mathsf{t}_{b2}\}, \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk})$. Il peut alors construire une signature sur le message $\mathsf{m}^* = \mathsf{m}_1 \| \mathsf{m}_{a2} \| \mathsf{m}_3 \| \mathsf{m}_{b4}$: $\Sigma^* = (\sigma, \{\mathsf{r}_{a4}\}, \{\mathsf{t}_{b2}\}, \mathsf{ADM}, Id_{\mathsf{m}}, \mathsf{dpk})$.

En conséquence ce schéma n'est donc pas Signataire-**Responsable**. Notons que cette attaque permet de générer un nouveau message signé : l'**Infalsifiabilité** du schéma (voir Définition 6.2) ne peut donc pas être atteinte.

D'autre part, un signataire corrompu peut faire accuser le délégué par le Juge à partir du moment où celui-ci a généré au moins une paire message-signature modifiée. Pour cela, il lui suffit de retourner comme preuve la paire modifiée au lieu de la paire originale. Notons, que le délégué n'est jamais capable de faire accuser le signataire. Ce schéma est donc Délégué-**Responsable**.

En conclusion, on résume les propriétés de sécurité atteintes ✓ ou non ✗ par cette solution dans la Table 7.1.

	Et-**Imm**.	**Transp**.	**Responsable**		**Infal**.
			sig-**Resp**	del-**Resp**	
ValFixées	✓	✓	✗	✓	✗

Table 7.1 – Sécurité de *ValFixées* de [KL06].

7.2.2 Notre proposition

Nous allons maintenant présenter la réparation de la solution de [KL06] que nous avons présentée dans [CJ10a].

La réparation

Le principe de notre réparation est d'utiliser comme base le schéma de signature caméléon sûr décrit au chapitre 6 section 6.4 plutôt que le schéma [ACdT05] utilisé par [KL06]. On garde ensuite l'idée de remplacer le condensé caméléon des parties modifiables dans un ensemble par un accumulateur sur les valeurs autorisées.

Les différences entre les deux schémas résident en l'apparition du TAG et de la partie additionnelle. Le schéma étant une simple réparation du schéma que nous venons de présenter en utilisant le schéma de signature caméléon présenté au chapitre 6, nous ne donnons donc que les points clés du schéma réparé.

Premièrement, nous mettons en place le système de TAG.

- Nous ajoutons aux schémas utilisés un schéma de fonction pseudo aléatoire $\mathcal{PRF}$ (cf. Définition 24) et un schéma de générateur pseudo-aléatoire $\mathcal{PRG}$ (cf. Définition 23).
- Nous modifions l'algorithme $\mathcal{SCE}$.SigGénClé($\mathcal{SCE}$.param) afin d'ajouter la génération d'une clé secrète κ aléatoirement dans $\{0,1\}^\lambda$ qui sera utilisée pour la $\mathcal{PRF}$.

– Nous pouvons maintenant ajouter la génération du TAG au début de l'algorithme de signature $\mathcal{SCE}$.Signe($\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{V}$). Le signataire choisit aléatoirement $\mathsf{Nonce} \in \{0,1\}^\lambda$ et calcule

$$\mathsf{TAG} = \mathcal{PRG}(\mathsf{x}) \text{ avec } \mathsf{x} = \mathcal{PRF}(\kappa, \mathsf{Nonce}).$$

– De même, au début de l'algorithme $\mathcal{SCE}$.Modifie($\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \mathsf{V}, \mathsf{s}$) la génération aléatoire de TAG' et Nonce' dans $\{0,1\}^\lambda$.

Deuxièmement nous intégrons le principe de partie reconstituée additionnelle. Pour cela nous remplaçons la *procédure de reconstruction* de [KL06] par la procédure suivante.

Procédure de reconstruction :
Cette procédure publique utilise TAG, les aléas R, la clé publique du délégué dpk et celle du signataire spk :

1. Pour chaque partie, on calcule un message reconstitué $\tilde{\mathsf{m}}_i$.
$$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{Acc}_i & \text{si } \mathsf{m}_i \in \mathsf{ADM}^+ \\ \mathsf{h}_i = \mathcal{HC}.\text{Hache}(\mathsf{dpk}, \mathsf{m}_i||i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM}^- \\ \mathsf{m}_i||i & \text{sinon} \end{cases}$$
2. Puis on calcule une partie finale supplémentaire.
$\mathsf{h}_c = \mathcal{HC}.\text{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c)$

Le message reconstitué est $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1\| \cdots \|\tilde{\mathsf{m}}_t\|\mathsf{h}_c\|\mathsf{dpk}$.

Ce changement implique l'ajout de r_c dans l'ensemble des aléas R ainsi que l'ajout à la signature caméléon des variables TAG et Nonce.

Afin de s'adapter à ces changements dans la signature, nous remplaçons la *Procédure de modification* utilisée lors d'une modification par la procédure suivante.

Procédure de modification :
Cette procédure publique utilise les aléas R, les valeurs des parties reconstitués $\tilde{\mathsf{m}}_i$ obtenues par la *Procédure de reconstruction*, l'ensemble des témoins $\mathcal{T}_i$, la clé secrète du délégué dsk et la clé publique du signataire spk :

1. Pour tout $i \in \mathsf{MOD}^+$, le délégué remplace le témoin t_i original du message original m_i par le témoin, noté t'_i, correspondant à la valeur m'_i qu'il souhaite.
2. Pour tout $i \in \mathsf{MOD}^-$ le délégué calcule
$$\mathsf{r}'_i = \mathcal{HC}.\text{Forge}(\mathsf{dsk}, \mathsf{m}_i\|i, \mathsf{m}'_i\|i, \mathsf{r}_i, \tilde{\mathsf{m}}_i).$$
3. Enfin, il calcule $\mathsf{r}'_c = \mathcal{HC}.\text{Forge}(\mathsf{dsk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{TAG}'\|\mathsf{m}'\|\mathsf{spk}, \mathsf{r}_c, \mathsf{h}_c)$.

La délégué réactualise les aléas et les témoins : $\mathsf{R}' = \{\{\mathsf{r}'_i\}_{i\in\mathsf{ADM}}, \mathsf{r}'_c\}$ avec $\mathsf{r}'_i = \mathsf{r}_i$ si m_i n'a pas été modifié, et $\mathsf{T}' = \{\mathsf{t}'_i, i \in \mathsf{ADM}\}$ en considérant que $\mathsf{t}'_i = \mathsf{t}_i$ si m_i n'a pas été modifié.

L'ajout de TAG et de la partie finale additionnelle implique de modifier la preuve qui devient $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ avec $\Sigma_j = (\sigma_j, \mathsf{R}_j, \mathsf{T}_j, \mathsf{ADM}_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{dpk}, \{\mathsf{apk}_{i_j}\}_{i\in\mathsf{ADM}_j^+})$ et $\mathsf{r}_{c_j} \in \mathsf{R}_j$ qui respecte les *Équations de vérifications* suivantes.

Équations de vérifications :

1. $\mathcal{SCE}.\text{VÉRIFIE}(\mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk}) = \mathsf{vrai}$;
2. $\mathsf{m}_j \neq \mathsf{m}$;
3. $\mathcal{HC}.\text{HACHE}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\text{HACHE}(\mathsf{dpk}, \mathsf{TAG}_j\|\mathsf{m}_j\|\mathsf{spk}, \mathsf{r}_{c_j})$ avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.

Preuve de sécurité de notre schéma

Théorème 8. *Sous l'hypothèse que le schéma de signature $\mathcal{S}$ soit **EU-CMA** sûr, que les fonctions $\mathcal{PRF}$ et PRG choisies soient pseudo-aléatoires, que la fonction caméléon $\mathcal{HC}$ soit **uniforme** et **résistante aux collisions** et que le schéma d'accumulateur $\mathcal{ACC}$ soit **résistant aux collisions**, notre schéma est un schéma de signature caméléon sûr dans notre modèle.* ♢

Idée des preuves. Nous allons à présent démontrer notre théorème de sécurité, pour cela nous devons montrer que notre schéma est Et-**Immuable**, **Transparent**, **Responsable** et **Traçable**.

Immuabilité étendue. Un adversaire $\mathcal{A}$ gagne l'expérience de l'**Immuabilité étendue** s'il exhibe un uplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*, \mathsf{maxV}^*, \mathsf{maxMod}^*, \mathsf{V}^*)$ tel que

1. $\mathsf{dpk}^* \neq \mathsf{dpk}_i$ ou $\exists j_i \notin \mathsf{ADM}_i$ avec $\mathsf{m}^*[j_i] \neq \mathsf{m}_i[j_i]$
2. ou $\exists j$ tel que $\mathsf{m}^*[j] \notin \mathsf{V}^*_j$.

Dans le cas 1, nous redirigeons le lecteur vers la preuve d'**Immuabilité** de notre schéma de signature caméléon non étendu, Chapitre 6 section 6.4. Si on note S_1 cet événement, nous avons :

$$\mathsf{Succ}^{\mathcal{SCE}}_{S_1,\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

Dans le cas 2, soit l'adversaire a réussi à ajouter un élément dans l'accumulateur sans en modifier la valeur et on peut alors utiliser un tel adversaire contre la **résistance aux collisions** du schéma $\mathcal{ACC}$ soit il a ajouté l'élément en modifiant l'accumulateur qui est signé par le schéma de signature $\mathcal{S}$ et l'adversaire peut être utilisé pour une attaque à message choisi visant la falsification existentielle du schéma de signature $\mathcal{S}$. Si on note S_2 cet événement, nous avons :

$$\mathsf{Succ}^{\mathcal{SCE}}_{S_2,\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{ACC}}_{\mathbf{collRes},\mathcal{A}}(\lambda) + \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

En conclusion, la probabilité de succès d'un adversaire contre l'**Immuabilité étendue** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{ACC}}_{\mathbf{collRes},\mathcal{A}}(\lambda) + \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

□

Transparence. Les seules différences entre une signature originale et une signature modifiée résident dans la construction du TAG et des aléas de R. En effet les témoins utilisés dans une signature originale sont identiques à ceux utilisés par le délégué.

La preuve que ce schéma est transparent est identique à la preuve de **Transparence** de notre schéma de signature caméléon classique présenté au chapitre 6. Nous réorientons le lecteur vers la preuve correspondante chapitre 6, section 7.

La probabilité de succès d'un adversaire contre la **Transparence** de notre schéma est :

$$Adv^{\mathcal{SCE}}_{\mathbf{Transp},\mathcal{A}}(\lambda) \leq 2\mathsf{Adv}^{\mathcal{PRF}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) + 2\mathsf{Adv}^{\mathcal{PRG}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) + 2\mathsf{Adv}^{\mathcal{HC}}_{\mathbf{uni},\mathcal{A}}(\lambda).$$

□

Responsabilité. La **Responsabilité** de ce schéma repose sur la partie supplémentaire h_c. Or cette partie n'a pas été modifiée entre la version classique et la version étendue du schéma. La preuve de la **Responsabilité** de ce schéma se déroule exactement de la même manière que la preuve correspondante pour le schéma initial présenté dans le chapitre 6, section 6.4. Nous ne répétons donc pas la preuve ici et proposons au lecteur de se référer à la preuve correspondante décrite en section 6.3. La probabilité de succès d'un adversaire contre la Délégué-**Responsabilité** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

Et la probabilité de succès d'un adversaire contre la Signataire-**Responsabilité** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathsf{sig}-\mathbf{Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{PRG}}_{\mathbf{OW},\mathcal{A}}(\lambda) + \mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda).$$

□

7.3 Constructions pour l'extension *ModifEgales*

Dans cette extension, le signataire peut imposer qu'un ensemble de parties modifiables restent identiques entre elles. Si le délégué souhaite modifier une des parties de l'ensemble, il doit toutes les modifier de la même manière.

7.3.1 La solution de M.Klonowski et A.Lauks

L'idée de ce schéma est d'utiliser une fonction caméléon spécifique permettant de hacher toutes les parties du message en un seul condensé. Cette fonction pouvant générer une sous clé pour un ensemble de parties identiques. Plus précisément, un ensemble de secrets $\{x_i\}_{i\in\mathsf{ADM}}$ est choisi aléatoirement dans $\{0,1\}^\lambda$. Et, en considérant un message m découpé en t parties la fonction caméléon est :

$$\mathcal{HC}.\textsc{Hache}(\{x_i\}_{i\in\mathsf{ADM}}, \mathsf{m}, \mathsf{r}) = g^{\mathsf{r}} \cdot \prod_{i=1}^{\mathsf{t}} g^{x_i \cdot \mathsf{m}_i}.$$

Le signataire obtient le message reconstitué $\tilde{\mathsf{m}} = \mathcal{HC}.\textsc{Hache}(\{x_i\}_{i\in\mathsf{ADM}}, \mathsf{m}, \mathsf{r})$ qui est ensuite signé.
Pour sa part, le délégué reçoit une clé secrète partielle $s_j = \sum_{i\in\mathcal{E}_j} x_i$ pour chaque ensemble de messages modifiables devant garder la même valeur. Le délégué peut ainsi adapter l'aléa r en fonction des modifications qu'il souhaite sur les m_i.

Cependant cette solution présente quelques problèmes que nous pourrons mieux étudier après une description plus formelle.

Construction 21.
Solution *ModifEgales* de [KL06] Ce schéma utilise un schéma de signature **EU-CMA** sûr, noté $\mathcal{S}$ (voir Définition 29).

$\mathcal{SCE}.\textsc{Init}(1^\lambda)$. Cette étape initialise le schéma de signature $\mathcal{S}.\mathsf{param} = \mathcal{S}.\textsc{Init}(\lambda)$ en fonction du paramètre de sécurité λ et choisit un générateur g de $\mathbb{Z}_p^*$ pour p premier.

$$\mathcal{SCE}.\mathsf{param} = \{\mathcal{S}.\mathsf{param}, g, \mathbb{Z}_p^*\} \leftarrow \mathcal{SCE}.\textsc{Init}(1^\lambda)$$

$\mathcal{SCE}$.SigGénClé($\mathcal{SCE}$.param). La paire de clés du signataire correspond à une paire de clés de signataire du schéma de signature choisi $(\mathsf{spk}, \mathsf{ssk}) = \mathcal{S}$.GénClé($\mathcal{S}$.param).

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SCE}.\textsc{SigGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.Signe($\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{E}$). Le message m est décomposé en fonction de ADM : $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_\mathsf{t}$. Le signataire choisit aléatoirement $\mathsf{t}+1$ valeurs $x_1, \cdots, x_\mathsf{t}, \mathsf{r}$ dans $\{0,1\}^\lambda$.
Il calcule ensuite $h_i = g^{x_i}$ pour tout $i \in [1, \mathsf{t}]$ et génère le message reconstitué grâce à la *Procédure de reconstruction* suivante.

Procédure de reconstruction :
Cette procédure publique utilise les générateurs $g, h_1, \cdots, h_\mathsf{t}$ et l'aléa r :

$$\tilde{\mathsf{m}} = g^\mathsf{r} \cdot \prod_{i=1}^{\mathsf{t}} h_i^{\mathsf{m}_i}. \tag{7.1}$$

Le message reconstitué est $\tilde{\mathsf{m}}$.

Le signataire signe le message reconstitué $\sigma = \mathcal{S}$.Signe($\mathsf{spk}, \mathsf{ssk}, \tilde{\mathsf{m}}$) et obtient la signature caméléon $\Sigma = (\sigma, \mathsf{r}, h_1, \cdots, h_\mathsf{t})$. Enfin, il calcule une valeur secrète $s_j = \sum_{i \in \mathcal{E}_j} x_i$ pour chaque ensemble de parties à valeurs égales $\mathcal{E}_j \in \mathsf{E}$. On note s l'ensemble de ces valeurs secrètes.

$$(\Sigma = (\sigma, \mathsf{r}, h_1, \cdots, h_\mathsf{t}), \mathsf{s}) \leftarrow \mathcal{SCE}.\textsc{Signe}(\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \cdot, \mathsf{E}, \cdot, \cdot)$$

$\mathcal{SCE}$.Modifie($\mathsf{m}, \Sigma, \mathsf{dsk}, \mathsf{MOD}, \mathsf{E}, \mathsf{s}$). Le délégué utilise les données secrètes s fournies par le signataire pour modifier le message m en le message m' en fonction de MOD. Pour tout $\mathcal{E}_j \in \mathsf{E}$ le délégué connait s_j. On note $\mathsf{m}[j]$ l'ancienne valeur de toutes les parties de $\mathcal{E}_j$ et $\mathsf{m}'[j]$ la nouvelle valeur voulue. Le délégué calcule le nouvel aléa r' de la manière suivante.

$$\mathsf{r}' = r - \sum_j s_j \cdot (\mathsf{m}'[j] - \mathsf{m}[j]).$$

Il obtient la signature caméléon étendue $\Sigma' = (\sigma, \mathsf{r}, h_1, \cdots, h_\mathsf{t})$ sur le message m'.

$$(\Sigma' = (\sigma, \mathsf{r}', h_1, \cdots, h_\mathsf{t}), \mathsf{m}') \leftarrow \mathcal{SCE}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \cdot, \mathsf{E}, \cdot, \cdot, \mathsf{s})$$

$\mathcal{SCE}$.Vérifie($\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk}$). Le vérifieur utilise la *Procédure de reconstruction* afin d'obtenir le message reconstitué $\tilde{\mathsf{m}}$ puis il vérifie si $\mathcal{S}$.Vérifie($\mathsf{spk}, \tilde{\mathsf{m}}$) = vrai. Si ce n'est pas le cas, il retourne faux.

Analyse de la solution [KL06].

Les algorithmes $\mathcal{SCE}$.Prouve et $\mathcal{SCE}$.Juge ne sont pas définis dans l'article [KL06], la **Responsabilité** n'y est pas prévue. De plus, le délégué n'ayant pas de secret qu'il ne partage pas avec le signataire la propriété de Délégué-**Responsabilité** n'est donc pas applicable sans modification du schéma.

Ce schéma est vulnérable à une généralisation de l'attaque d'exposition de clés de G.Ateniese et B.de Medeiros [Ad04] que nous avons présentée dans le contexte des signatures caméléons en section 6.3. Cette attaque nous permet de retrouver certains s_j à partir de paires message-signature d'une même instance. Le principe de l'attaque proposée est le suivant.

Supposons qu'un attaquant connaisse un ensemble de messages signés $\{\mathsf{m}^{(k)}\}_{k\in[1,\mathsf{q}]}$, tous différents entre eux mais tous issus d'une même instance.
Par construction, pour chaque message $\mathsf{m}^{(k)}$ signé nous avons l'équation suivante.

$$\tilde{\mathsf{m}}^{(k)} = g^{\mathsf{r}} \cdot \prod_{i=1}^{\mathsf{t}} g^{x_i \cdot \mathsf{m}_{ki}} \tag{7.2}$$

Or pour tout $i \in \mathcal{E}_j$, les parties m_i sont égales entre elles. On note v_j cette valeur commune dans un ensemble $\mathcal{E}_j$. De plus, pour chaque ensemble, le délégué utilise toujours la même valeur secrète s_j. Donc, en notant y le logarithme discret de $\tilde{\mathsf{m}}$ en base g, $\|\mathcal{E}_j\|$ le cardinal de l'ensemble $\mathcal{E}_j$ et $\|\mathsf{E}\|$ le nombre d'ensembles $\mathcal{E}_j$ dans E, nous avons l'équation 7.3 pour chaque message signé $\mathsf{m}^{(k)}$.

$$y = \mathsf{r}^{(k)} + s_1 \cdot \|\mathcal{E}_1\| \cdot v_1^{(k)} + \ldots + s_{\|\mathsf{E}\|} \cdot \|\mathcal{E}_{\|\mathsf{E}\|}\| \cdot v_{\|\mathsf{E}\|}^{(k)} \tag{7.3}$$

L'attaquant peut alors construire le système d'équations suivant.

$$\begin{cases} y = \mathsf{r}^{(1)} + s_1 \cdot \|\mathcal{E}_1\| \cdot v_1^{(1)} + \ldots + s_{\|\mathsf{E}\|} \cdot \|\mathcal{E}_{\|\mathsf{E}\|}\| \cdot v_{\|\mathsf{E}\|}^{(1)} \\ y = \mathsf{r}^{(2)} + s_1 \cdot \|\mathcal{E}_1\| \cdot v_1^{(2)} + \ldots + s_{\|\mathsf{E}\|} \cdot \|\mathcal{E}_{\|\mathsf{E}\|}\| \cdot v_{\|\mathsf{E}\|}^{(2)} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ y = \mathsf{r}^{(i)} + s_1 \cdot \|\mathcal{E}_1\| \cdot v_1^{(\mathsf{q})} + \ldots + s_{\|\mathsf{E}\|} \cdot \|\mathcal{E}_{\|\mathsf{E}\|}\| \cdot v_{\|\mathsf{E}\|}^{(\mathsf{q})} \end{cases} \tag{7.4}$$

L'attaquant obtient ainsi un système de q équations à $\|\mathsf{E}\| + 1$ inconnues : y et les s_j. Si l'attaquant peut partiellement résoudre le système d'équation 7.4, alors il obtient une partie des s_j. Et ainsi créer des collisions sur les parties correspondantes.

Avec cette attaque un adversaire peut obtenir une signature valide sur un nouveau message modifié. Ce schéma ne peut donc pas être Signataire-**Responsable**.

Remarque. Le fait de pouvoir générer un nouveau message signé implique que la propriété d'**Infalsifiabilité** du schéma de signature caméléon (voir Définition 6.2) ne peut être atteinte.

En conclusion, on résume les propriétés de sécurité vérifiées ✓ ou non ✗ par cette solution dans la Table 7.2.

	Et-**Imm.**	**Transp.**	**Responsable**		**Infal.**
			sig-**Resp**	del-**Resp**	
ModifEgales	✓	✓	✗	Non applicable	✗

Table 7.2 – Sécurité de *ModifEgales* de [KL06].

7.3.2 Solution naïve

Il n'existe pas, à ce jour, de solution élégante pour cette extension. Il existe cependant une solution triviale qui consiste à ajouter au message reconstitué la description des ensembles de parties devant rester égales entre elles. Une telle solution appliquée à notre schéma de signature caméléon décrit au chapitre 6 respecterait les propriétés de sécurité exigées.

La preuve d'un tel schéma est identique aux preuves effectuées sur le schéma caméléon standard (voir Chapitre 6 Section 6.4) à l'exception de l'immuabilité étendue qui repose alors sur le fait que la signature classique $\mathcal{S}$ utilisée dans le schéma est **EU-CMA** sûre. Les autres propriétés sont ensuite prouvées de la même façon que dans le cas simple.

Cette solution naïve est peu efficace en espace (les descriptions des parties devant rester identiques étant ajoutées à la signature) et la description de ces parties est alors publique.

7.4 Constructions pour l'extension *LimNbModif*

Cette extension permet au signataire de fixer une limite au nombre de parties que le délégué pourra modifier parmi les parties modifiables.

7.4.1 La solution de M.Klonowski et A.Lauks

La solution proposée par M.Klonowski et A.Lauks [KL06] utilise l'interpolation polynomiale de Lagrange que nous avons vu en Définition 8 et 9.

L'idée de M.Klonowski et A.Lauks est de définir un polynôme secret P_{maxMod} de degré maxMod tel que la clé secrète du délégué corresponde à $P_{\mathsf{maxMod}}(0) = \mathsf{usk}$. A chaque fois que le délégué modifie une partie, la construction permet à un point du polynôme d'être déduit. Une fois $\mathsf{maxMod}+1$ parties modifiées, $\mathsf{maxMod}+1$ points du polynôme sont déductibles. Le polynôme $P_{\mathsf{maxMod}}(y)$ peut alors être calculé par interpolation lagrangienne (voir Définition 9) et la clé du délégué peut donc être retrouvée.

Le schéma proposé utilise le schéma de fonction de hachage caméléon faible décrit dans la définition 1 que nous notons $f\mathcal{HC}$. Il s'agit d'un schéma de fonction de hachage caméléon **uniforme** mais sensible à l'attaque d'exposition de clé de G.Ateniese et B.de Medeiros [Ad04] présenté au chapitre 1, Définition 1.

Le schéma de signature caméléon étendu à *LimNbModif* de [KL06] fonctionne de la manière suivante.

Construction 22 (Solution LimNbModif de [KL06]).
Ce schéma utilise un schéma de signature $\mathbf{EU-CMA}$ sûr, noté $\mathcal{S}$ (voir Définition 29) ainsi que le schéma de fonction de hachage caméléon faible $f\mathcal{HC}$.

$\mathcal{SCE}$.Init(1^λ) Cet algorithme consiste en l'initialisation du schéma de signature $\mathcal{S}.\mathsf{param} = \mathcal{S}.\text{Init}(\lambda)$ et de la fonction de hachage caméléon $f\mathcal{HC}.\mathsf{param} = f\mathcal{HC}.\text{Init}(\lambda)$ en fonction du paramètre de sécurité λ. Nous notons g le générateur défini dans les paramètres du schéma $f\mathcal{HC}$.

$$\mathcal{SCE}.\mathsf{param} = \{\mathcal{S}.\mathsf{param}, f\mathcal{HC}.\mathsf{param}\} \leftarrow \mathcal{SCE}.\text{Init}(1^\lambda)$$

$\mathcal{SCE}$.SigGénClé($\mathcal{SCE}.\mathsf{param}$) La paire de clés du signataire correspond à une paire de clés de signataire du schéma de signature choisi $(\mathsf{spk}, \mathsf{ssk}) = \mathcal{S}.\text{GénClé}(\mathcal{S}.\mathsf{param})$.

$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathcal{SCE}.\text{SigGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.SanGénClé($\mathcal{SCE}.\mathsf{param}$) L'algorithme choisit aléatoirement dans $\{0,1\}^\lambda$ la clé secrète d'utilisateur usk du délégué et construit la clé publique correspondante en utilisant le générateur g de $f\mathcal{HC}.\mathsf{param}$: $\mathsf{upk} = g^{\mathsf{usk}}$. Il choisit ensuite aléatoirement $k \geq \mathsf{maxMod}$ valeurs $f_1, \cdots, f_k$ dans $\{0,1\}^\lambda$ et calcule $g_i = g^{f_i}$ pour tout $i \in [1,k]$. Il obtient ainsi la paire de clés de délégué $(\mathsf{dsk} = \{f_1, \cdots, f_k\}, \mathsf{dpk} = \{\mathsf{upk}, g_1, \cdots, g_k\})$.
À tout moment, le délégué peut ajouter à sa clé publique de nouveaux f_i, g_i.

$$(\mathsf{dsk}, \mathsf{dpk}, \mathsf{usk}) \leftarrow \mathcal{SCE}.\text{SanGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.SIGNE($\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{maxMod}$) Le signataire vérifie que la clé publique du délégué contient plus de maxMod valeurs g_i. Si ce n'est pas le cas, il demande au délégué d'ajuster l'ensemble. Il choisit ensuite aléatoirement dans $\{0,1\}^\lambda$ un identifiant d'instance Id_m ainsi qu'un r_i pour chaque $i \in \mathsf{ADM}$, on note $\mathsf{R} = \{\mathsf{r}_i, \forall i \in \mathsf{ADM}\}$. Il génère ensuite une clé publique de fonction de hachage caméléon pour chaque partie. Pour alléger la notation on pose, exceptionnellement, $\mu\mathsf{maxMod}$.

$$\forall i \in [1,\mu], \quad z_i = \mathsf{upk} \cdot g_1^i \cdot \ldots \cdot g_\mu^{i\mu}$$

On remarque que le délégué connaissant les f_i pourra calculer un polynôme $F(y) = \mathsf{usk} + f_1 \cdot y + \ldots + f_{\mathsf{maxMod}} \cdot y^{\mathsf{maxMod}}$ tel que $z_i = g^{F(i)}, \forall i \in \mathsf{ADM}$.
Le signataire calcule ensuite le message $\tilde{\mathsf{m}}_i$ reconstitué de la façon suivante.

> *Procédure de reconstruction* :
> Cette procédure publique utilise les aléas R, les clés publiques de parties $\{z_i\}_{i\in\mathsf{ADM}}$, l'identifiant d'instance Id_m et la clé publique de délégué $\mathsf{dpk} = \{\mathsf{upk}, g_1, \cdots, g_k\}$:
>
> Pour chaque partie, on calcule un message reconstitué $\tilde{\mathsf{m}}_i$.
>
> $$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = f\mathcal{HC}.\text{HACHE}(z_i, Id_\mathsf{m}\|i\|\mathsf{m}_i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i\|i & \text{sinon} \end{cases}$$
>
> Le message reconstitué est $\tilde{\mathsf{m}} = Id_\mathsf{m}\|\mathsf{t}\|\mathsf{upk}\|\tilde{\mathsf{m}}_1\|\cdots\|\tilde{\mathsf{m}}_t$.

Enfin, le signataire signe ce message grâce au schéma de signature $\sigma = \mathcal{S}.\text{SIGNE}(\mathsf{spk}, \mathsf{ssk}, \tilde{\mathsf{m}})$ et obtient la signature caméléon étendue $\Sigma = (\sigma, \mathsf{R}, \mathsf{ADM}, \{z_i\}_{i\in\mathsf{ADM}})$ avec $\mathsf{R} = \{\mathsf{r}_i\}_{i\in\mathsf{ADM}}$.

$$(\Sigma = (\sigma, \mathsf{R}, \mathsf{ADM}, \{z_i\}_{i\in\mathsf{ADM}}), \mathsf{s} = \perp) \leftarrow \mathcal{SCE}.\text{SIGNE}(\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \cdot, \cdot, \mathsf{maxMod}, \cdot)$$

$\mathcal{SCE}$.MODIFIE($\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \mathsf{maxMod}, \mathsf{s}$) Le délégué vérifie que MOD correspond à ADM et, si ce n'est pas le cas, il s'arrête en retournant $\perp$. Le délégué reconstruit le message reconstitué grâce à la *Procédure de reconstruction* . Il obtient $\tilde{\mathsf{m}} = Id_\mathsf{m}\|\mathsf{t}\|\mathsf{upk}\|\tilde{\mathsf{m}}_1\|\cdots\|\tilde{\mathsf{m}}_t$ et donc les valeurs $\tilde{\mathsf{m}}_i = \mathsf{h}_i$ pour tout $i \in \mathsf{MOD} \subset \mathsf{ADM}$.
Si le délégué ne l'a pas déjà calculé et conservé lors d'une modification précédente d'un message de cette instance, il retrouve le polynôme $F(y) = \mathsf{usk} + f_1 \cdot y + \ldots + f_{\mathsf{maxMod}} \cdot y^{\mathsf{maxMod}}$ qui est tel que $z_i = g^{F(i)}, \forall i \in \mathsf{ADM}$. Avec ce polynôme, il peut construire les collisions sur les m_i grâce aux $F(i)$ et à l'algorithme $f\mathcal{HC}$.FORGE de la manière suivante.

$$\forall i \in \mathsf{MOD}, \mathsf{r}'_i = f\mathcal{HC}.\text{FORGE}(F(i), \mathsf{m}_i\|i, \mathsf{m}'_i\|i, \mathsf{r}_i, \tilde{\mathsf{m}}_i)$$

Enfin, il met à jour l'ensemble R' : $\mathsf{r}'_i = \mathsf{r}_i$ si $i \notin \mathsf{MOD}$ sinon $\mathsf{r}'_i = \mathsf{r}'_i$. La signature modifiée est $\Sigma' = (\sigma, \mathsf{R}', \mathsf{ADM}, \{z_i\}_{i\in\mathsf{ADM}})$.

$$(\Sigma' = (\sigma, \mathsf{R}', \mathsf{ADM}, \{z_i\}_{i\in\mathsf{ADM}}), \mathsf{m}') \leftarrow \mathcal{SCE}.\text{MODIFIE}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \cdot, \cdot, \mathsf{maxMod}, \cdot, \mathsf{s})$$

$\mathcal{SCE}$.VÉRIFIE($\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk}$) Le vérifieur reconstruit le message reconstitué grâce à la *Procédure de reconstruction*. Il obtient ainsi $\tilde{\mathsf{m}} = Id_\mathsf{m}\|\mathsf{t}\|\mathsf{upk}\|\tilde{\mathsf{m}}_1\|\cdots\|\tilde{\mathsf{m}}_t$. Enfin il retourne la sortie de l'algorithme de vérification sur le message $\mathcal{S}.\text{VÉRIFIE}(\mathsf{spk}, \sigma, \tilde{\mathsf{m}})$.

$\mathcal{SCE}$.TESTFRAUDE($\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB}$) On rappelle que pubDB est une base de données de paires message-signature d'une même instance. Pour tout $i \in \mathsf{ADM}$, le testeur cherche

deux messages signés $\mathsf{m}_a, \mathsf{m}_b \in \mathsf{pubDB}$ tels que $\mathsf{m}_{a_i} \neq \mathsf{m}_{b_i}$. On note pubDB' l'ensemble des triplets $(\mathsf{m}_a, \mathsf{m}_b, i)$. Pour chaque triplet, le testeur retrouve la clé secrète $F(i)$ utilisée dans la fonction de hachage caméléon faible correspondante grâce à l'attaque de [Ad04].
Il construit ensuite le polynôme d'interpolation de Lagrange (voir Définition 9) correspondant aux points d'interpolation $\{(k, F(k))\}_{k \in \mathsf{pubDB}'}$ et obtient un polynôme $P_{|\mathsf{pubDB}'|}(y)$.
Si $g^{P_{|\mathsf{pubDB}'|}(0)} \neq \mathsf{upk}$, il n'y a pas de fraude dans pubDB et le testeur retourne $\perp$. Sinon le testeur obtient $\mathsf{usk} = P_{|\mathsf{pubDB}'|}(0)$ et la preuve $\pi_{\mathsf{fraude}} = (\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB}', P_{|\mathsf{pubDB}'|}(y))$.

$$(\pi_{\mathsf{fraude}} = (\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB}'), \mathsf{dsk}) \leftarrow \mathcal{SCE}.\textsc{TestFraude}(\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB})$$

$\mathcal{SCE}.\textsc{VérifFraude}(\pi_{\mathsf{fraude}}, \mathsf{dsk})$ Le vérifieur vérifie que la clé secrète dsk correspond à la clé publique du délégué $\mathsf{upk} = g^{\mathsf{usk}}$ et au polynôme exhibé $P(0) = \mathsf{usk}$, si ce n'est pas le cas il retourne faux. Il vérifie ensuite la base de données pubDB' en recalculant les $F(j)$ par l'attaque de [Ad04] et en vérifiant que pour chacun d'entre eux $F(j) = P(j)$ si c'est le cas il retourne vrai sinon il retourne faux.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SCE}.\textsc{VérifFraude}(\pi_{\mathsf{fraude}}, \mathsf{dsk})$$

Analyse de la solution [KL06].

Les algorithmes $\mathcal{SCE}.\textsc{Prouve}$ et $\mathcal{SCE}.\textsc{Juge}$ n'existent pas dans la description de cette extension par [KL06] en effet, l'article ne considérait pas la **Responsabilité**. De plus, par construction, la fonction caméléon utilisée est sensible à l'attaque d'exposition de clé de [Ad04]. L'objectif étant que pour chaque partie modifiée, la clé secrète $F(j)$ de la partie puisse être publiquement retrouvée. Malheureusement cela implique qu'un adversaire peut utiliser ces clés pour modifier comme il le souhaite les parties concernées et donc obtenir une signature valide sur un nouveau message modifié. Ce schéma ne peut donc pas être Signataire-**Responsable**. De plus, il n'existe pas de solution triviale pour que le signataire prouve qu'il a (ou n'a pas) généré une paire. Le schéma n'est donc pas Délégué-**Responsable**.

Remarque. Le fait de pouvoir générer une collision sur les parties déjà modifiées du schéma implique que la propriété d'**Infalsifiabilité** du schéma de signature caméléon ne peut être atteinte.

En conclusion, on résume les propriétés de sécurité atteintes ✓ou non ✗ par cette solution dans la Table 7.3.

	Et-**Imm.**	**Transp.**	**Responsable**		**Infal.**	**Traçable**
			sig-**Resp**	del-**Resp**		
LimNbModif	✓	✓	✗	✗	✗	✓

Table 7.3 – Sécurité de *LimNbModif* [KL06].

7.4.2 Notre solution

Je présente ici une solution simplifiée de la construction proposée avec S.Canard dans [CJ10a] qui atteint une meilleure complexité.

Le schéma

Cette solution pour cette extension utilise l'interpolation polynomiale de Lagrange comme définie en Définition 7.4. Ce schéma utilise la même idée que [KL06]. Nous utilisons un polynôme secret P_{maxMod} de degré maxMod tel que la clé secrète du délégué corresponde à $P_{\mathsf{maxMod}}(0) = \mathsf{usk}$. A chaque fois que le délégué modifie une partie, la construction permet à un point du polynôme d'être déduit. Une fois $\mathsf{maxMod} + 1$ parties modifiées, $\mathsf{maxMod} + 1$ points du polynôme sont déductibles. Le polynôme d'interpolation de Lagrange peut alors être calculé sur les points d'interpolations $(j, P_{\mathsf{maxMod}}(j))$ (voir Définition 9) et la clé secrète d'utilisateur du délégué peut donc être retrouvée.

Cette solution est plus simple que la solution proposée dans [CJ10a] et est plus proche du schéma original de [KL06]. Il y a principalement trois points clés qui différencient notre schéma de celui de [KL06]. Premièrement nous réutilisons le principe de TAG que nous avons utilisé pour notre schéma de signature caméléon classique présenté au chapitre 6. Deuxièmement, nous ajoutons un deuxième schéma de signature caméléon **uniforme** et **résistant aux collisions**. Et enfin nous utilisons ce schéma de fonction de hachage caméléon sur pour générer une partie intermédiaire additionnelle qui correspondra au condensé caméléon sur l'intégralité du message et le TAG.

Le schéma [CJ10a] propose d'utiliser le schéma de signature caméléon simple présenté au chapitre 6 en ajoutant un second condensé par partie. Celui-ci est effectué sur l'aléa obtenu avec le premier condensé et avec une fonction caméléon faible qui permettra d'obtenir les points du polynôme d'interpolation. Cette solution multiplie par deux le nombre d'appels à une fonction de hachage caméléon et implique un doublement de l'ensemble des aléas compris dans la signature caméléon.

Afin de pouvoir présenter des preuves claires, nous décrivons à présent de manière formelle l'intégralité de notre construction.

Construction 23.
Notre solution *ModifEgales* Ce schéma utilise deux schémas de fonction de hachage caméléon : le schéma faible décrit en section 7.4 noté CH et un schéma **uniforme** et **résistant aux collisions** noté $\mathcal{HC}$ (cf. Définition 22). Enfin, il utilise un schéma de signature $\mathbf{EU-CMA}$ sûr noté $\mathcal{S}$ (cf. Définition 29), un schéma de fonction pseudo aléatoire $\mathcal{PRF}$ (cf. Définition 24) et un schéma de générateur pseudo-aléatoire $\mathcal{PRG}$ (cf. Définition 23). Le schéma fonctionne de la manière suivante.

$\mathcal{SCE}$.Init(1^λ). Cet algorithme prend en entrée 1^λ avec λ un paramètre de sécurité et initialise le système. Ce qui correspond ici à l'exécution de $\mathcal{HC}$.Init(λ) pour obtenir les paramètres de la fonction caméléon sûre $\mathcal{HC}.\mathsf{param}$, de $f\mathcal{HC}$.Init(λ) pour les paramètres $f\mathcal{HC}.\mathsf{param}$ du schéma faible CH. Nous notons g le générateur de groupe définit dans les paramètres du schéma de fonction de hachage caméléon faible $f\mathcal{HC}$ (cf. Définition 1). Enfin, il exécute $\mathcal{S}$.Init(λ) pour obtenir les paramètres du schéma de signature $\mathcal{S}.\mathsf{param}$. On note $\mathcal{SCE}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, f\mathcal{HC}.\mathsf{param}, \mathcal{S}.\mathsf{param}\}$.

$$\mathcal{SCE}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, f\mathcal{HC}.\mathsf{param}, \mathcal{S}.\mathsf{param}\} \leftarrow \mathcal{SCE}.\text{Init}(1^\lambda).$$

$\mathcal{SCE}$.SigGénClé$(\mathcal{SCE}.\mathsf{param})$. Cet algorithme exécute l'algorithme de génération des clés du schéma de signature $\mathcal{S}$.GénClé(1^λ) et obtient $(\mathsf{pk}, \mathsf{sk})$. Il choisit ensuite κ aléatoirement dans $\{0,1\}^\lambda$.

$$(\mathsf{spk} = \mathsf{pk}, \mathsf{ssk} = \{\mathsf{sk}, \kappa\}) \leftarrow \mathcal{SCE}.\text{SigGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.SanGénClé($\mathcal{SCE}$.param). L'algorithme exécute $\mathcal{HC}$.GénClé(1^λ) afin d'obtenir une paire de clés (csk, cpk) pour $\mathcal{HC}$. L'algorithme choisit ensuite aléatoirement dans $\{0,1\}^\lambda$ la clé secrète d'utilisateur usk du délégué et construit la clé publique correspondante en utilisant le générateur g : $\mathsf{upk} = g^{\mathsf{usk}}$. Il choisit ensuite aléatoirement $k \geq \mathsf{maxMod}$ valeurs $f_1, \cdots, f_k$ dans $\{0,1\}^\lambda$ et calcule $g_i = g^{f_i}$ pour tout $i \in [1,k]$ avec g le générateur contenu dans $\mathcal{HC}$.param. Il obtient ainsi la paire de clés de délégué ($\mathsf{dsk} = \{\mathsf{csk}, f_1, \cdots, f_k\}, \mathsf{dpk} = \{\mathsf{cpk}, \mathsf{upk}, g_1, \cdots, g_k\}$).

$$(\mathsf{dsk}, \mathsf{dpk}, \mathsf{usk}) \leftarrow \mathcal{SCE}.\text{SanGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.Signe(m, ssk, dpk, ADM, $\cdot$, $\cdot$, maxMod, $\cdot$). Le signataire génère un TAG pour son message. Il choisit aléatoirement $\mathsf{Nonce} \in \{0,1\}^\lambda$, calcule $\mathsf{x} = \mathcal{PRF}(\kappa, \mathsf{Nonce})$ et obtient $\mathsf{TAG} = \mathcal{PRG}(\mathsf{x})$. Il génère ensuite pour tout $i \in [1, \mathsf{ADM}]$ une clé publique fonctionnant pour la fonction de hachage caméléon faible.

$$z_i = \mathsf{upk} \cdot g_1^i \cdot \ldots \cdot g_{\mathsf{maxMod}}^{i^{\mathsf{maxMod}}}$$

On remarque que le délégué connaissant les f_i pourra calculer un polynôme $F(y) = \mathsf{usk} + f_1 \cdot y + \ldots + f_{\mathsf{maxMod}} \cdot y^{\mathsf{maxMod}}$ tel que $z_i = g^{F(i)}, \forall i \in \mathsf{ADM}$.
Il choisit $|\mathsf{ADM}| + 1$ aléas dans $\{0,1\}^\lambda$ que nous notons : $\mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}, \mathsf{r}_c\}$. Le signataire exécute alors une *procédure de reconstruction* sur le message m.

Procédure de reconstruction :
Cette procédure publique utilise TAG, les aléas R, la clé publique du délégué dpk, les clés publiques de la fonction caméléon faible $\{z_i\}_{i \in \mathsf{ADM}}$ et celle du signataire spk :

1. Pour chaque partie, on calcule un message reconstitué $\tilde{\mathsf{m}}_i$.
$$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = f\mathcal{HC}.\text{Hache}(z_i, \mathsf{m}_i || i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i || i & \text{sinon} \end{cases}$$
2. Puis on calcule une partie finale supplémentaire.
$$\mathsf{h}_c = \mathcal{HC}.\text{Hache}(\mathsf{cpk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, \mathsf{r}_c)$$

Le message reconstitué est $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_t \| \mathsf{h}_c \| \mathsf{cpk} \| \mathsf{upk} \| z_1 \| \cdots \| z_{|\mathsf{ADM}|}$.

Le signataire signe le message reconstitué $\tilde{\mathsf{m}}$ avec le schéma de signature choisi $\sigma = \mathcal{S}.\text{Signe}(\mathsf{spk}, \mathsf{ssk}, \tilde{\mathsf{m}})$ et obtient la signature caméléon suivante.

$$\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}, \{z_i\}_{i \in \mathsf{ADM}}) \text{ et } \mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}, \mathsf{r}_c\}.$$

La signature et le message correspondant sont ajoutés à la base de données du signataire BD.

$$\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}, \{z_i\}_{i \in \mathsf{ADM}}) \leftarrow \mathcal{SCE}.\text{Signe}(\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \cdot, \cdot, \mathsf{maxMod}, \cdot)$$

$\mathcal{SCE}$.Modifie(m, Σ, spk, dsk, MOD, $\cdot$, $\cdot$, maxMod, $\cdot$, s). Le délégué vérifie que MOD correspond à ADM et aux limitations, si ce n'est pas le cas il s'arrête en retournant $\perp$. Il met à jour les tailles ℓ_i des parties qui vont être modifiées puis choisit aléatoirement Nonce' et TAG' dans $\{0,1\}^\lambda$. Le délégué effectue la *procédure de reconstruction* comme décrit précédemment et obtient $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_t \| \mathsf{h}_c \| \mathsf{cpk} \| \mathsf{upk} \| z_1 \| \cdots \| z_{|\mathsf{ADM}|}$ et donc les valeurs h_c et $\tilde{\mathsf{m}}_i = \mathsf{h}_i$ pour tout $i \in \mathsf{MOD} \subset \mathsf{ADM}$.

Si le délégué ne l'a pas déjà calculé et conservé lors d'une modification précédente d'un message de cette instance, il retrouve le polynôme $F(y) = \mathsf{usk} + f_1 \cdot y + \ldots + f_{\mathsf{maxMod}} \cdot y^{\mathsf{maxMod}}$ qui est tel que $z_i = g^{F(i)}, \forall i \in \mathsf{ADM}$.
Il peut alors construire les collisions sur les parties m_i grâce aux $F(i)$ et la collision sur la partie supplémentaire grâce à sa clé secrète de délégué de la manière suivante.
– Pour tout $i \in \mathsf{MOD}$ le délégué calcule $\mathsf{r}'_i = f\mathcal{HC}.\textsc{Forge}(z_i, \mathsf{m}_i \| i, \mathsf{m}'_i \| i, \mathsf{r}_i, \tilde{\mathsf{m}}_i)$,
– puis $\mathsf{r}'_c = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk.csk} \|, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, \mathsf{TAG}' \| \mathsf{m}' \| \mathsf{spk}, \mathsf{r}_c, \mathsf{h}_c)$.
Enfin, il met à jour l'ensemble $\mathsf{R}' = \{\mathsf{r}'_1, \cdots, \mathsf{r}'_u, \mathsf{r}'_c\}$ de la manière suivante $\mathsf{r}'_i = \mathsf{r}_i$ si $i \notin \mathsf{MOD}$, sinon $\mathsf{r}'_i = \mathsf{r}'_i$. La signature modifiée est $\Sigma' = (\sigma, \mathsf{TAG}', \mathsf{Nonce}', \mathsf{R}', \mathsf{ADM}', \{z_i\}_{i \in \mathsf{ADM}})$.

$$\Sigma' \leftarrow \textsc{Sanitize}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \cdot, \cdot, \mathsf{maxMod}, \cdot, \mathsf{s}).$$

$\mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk})$. Le vérifieur exécute la *procédure de reconstruction* décrite plus haut et obtient le message reconstitué $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_t \| \mathsf{h}_c \| \mathsf{cpk} \| \mathsf{upk} \| \| z_1 \| \cdots \| z_{|\mathsf{ADM}|}$. Enfin il retourne la sortie de l'algorithme de vérification sur le message $\mathcal{S}.\textsc{Vérifie}(\mathsf{spk}, \sigma, \tilde{\mathsf{m}})$.

$\mathcal{SCE}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, \mathsf{BD} = (\mathsf{m}_k, \Sigma_k)_{k \in [1,\mathsf{q}]})$. Le signataire cherche un indice $j \in [1, \mathsf{q}]$ dans la base de données $\mathsf{BD} = (\mathsf{m}_k, \Sigma_k)_{k \in [1,\mathsf{q}]}$ tel que :
– $\mathsf{vrai} = \mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
– $\mathsf{m}_j \neq \mathsf{m}$;
– $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk.cpk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk.cpk}, \mathsf{TAG}_j \| \mathsf{m}_j \| \mathsf{spk}, \mathsf{r}_{c_j})$
avec $\mathsf{TAG}_j = \mathcal{PRG}(\mathsf{x}_j)$.
S'il trouve un tel indice j alors il obtient la preuve suivante.

$$\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, \mathsf{x}_j, \Sigma_j) \text{ avec } \Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j, \{z_i\}_{i \in \mathsf{ADM}_j} \text{ et } \mathsf{r}_{c_j} \in \mathsf{R}_j)$$

Sinon il note $\pi_{\mathsf{or/mod}} = \perp$.

$\mathcal{SCE}.\textsc{Juge}(\mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$. Si $\pi_{\mathsf{or/mod}} = \perp$ alors le juge retourne Signataire. Sinon il vérifie que la preuve $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, \mathsf{x}_j, \Sigma_j)$ respecte les propriétés suivantes :
– $\mathsf{vrai} = \mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
On pose $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j, \{z_i\}_{i \in \mathsf{ADM}_j})$ avec $\mathsf{R}' = \{\mathsf{r}_{1_j}, \cdots, \mathsf{r}_{u_j}, \mathsf{r}_{c_j}\}$.
– $\mathsf{m}_j \neq \mathsf{m}$;
– $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk.cpk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, \mathsf{r}_c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk.cpk}, \mathsf{TAG}_j \| \mathsf{m}_j \| \mathsf{spk}, \mathsf{r}_{c_j})$
avec $\mathsf{TAG}_j = \mathcal{PRG}(\mathsf{x}_j)$.
Si la preuve $\pi_{\mathsf{or/mod}}$ respecte toutes les propriétés, il retourne Délégué sinon il retourne Signataire.

$\mathcal{SCE}.\textsc{TestFraude}(\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB})$ On rappelle que pubDB est une base de données de paires message-signature d'une même instance. Pour tout $i \in \mathsf{ADM}$, le testeur cherche deux messages signés $\mathsf{m}_a, \mathsf{m}_b \in \mathsf{pubDB}$ tels que $\mathsf{m}_{a_i} \neq \mathsf{m}_{b_i}$. On note pubDB' l'ensemble des triplets $(\mathsf{m}_a, \mathsf{m}_b, i)$. Pour chaque triplet, le testeur retrouve la clé secrète $F(i)$ utilisée dans la fonction de hachage caméléon correspondante grâce à l'attaque de [Ad04].
Il construit ensuite le polynôme d'interpolation de Lagrange (voir Définition 9) correspondant aux points d'interpolation $\{(i, F(i))\}$ et obtient un polynôme $P_{|\mathsf{pubDB}'|}(y)$.
Si $g^{P_{|\mathsf{pubDB}'|}(0)} \neq \mathsf{upk}$, il n'y a pas de fraude dans l'ensemble pubDB et le testeur retourne $\perp$. Sinon il obtient $\mathsf{usk} = P_{|\mathsf{pubDB}'|}(0)$ et la preuve $\pi_{\mathsf{fraude}} = (\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB}', P_{|\mathsf{pubDB}'|}(y))$.

$$(\pi_{\mathsf{fraude}} = (\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB}', P_{|\mathsf{pubDB}'|}(y)), \mathsf{dsk}) \leftarrow \mathcal{SCE}.\textsc{TestFraude}(\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB})$$

$\mathcal{SCE}.\textsc{VérifFraude}(\pi_{\mathsf{fraude}}, \mathsf{usk})$ Le vérifieur vérifie que la clé secrète usk correspond à la clé publique de l'utilisateur $\mathsf{upk} = g^{\mathsf{usk}}$ et au polynôme exhibé $P_{\mathsf{maxMod}}(0) = \mathsf{usk}$, si ce n'est

pas le cas il retourne faux. Il vérifie ensuite la base de données pubDB'. Il vérifie d'abord que toute les paires de la base sont valides et appartiennent à la même instance (si elles ont toutes en commun le même σ). Puis il recalcule les $F(j)$ par l'attaque de [Ad04] et vérifie que pour chacun d'entre eux $F(j) = P_{\mathsf{maxMod}}(j)$ si c'est le cas il retourne vrai sinon il retourne faux.

↺

Preuve de sécurité de notre solution

Théorème 9. *Sous l'hypothèse que le schéma de signature $\mathcal{S}$ soit **EU-CMA** sûr, que les fonctions $\mathcal{PRF}$ et PRG choisies soient pseudo-aléatoires, que la fonction de hachage caméléon CH soit **uniforme** et que la fonction caméléon $\mathcal{HC}$ soit **uniforme** et **résistante aux collisions**, notre schéma est un schéma de signature caméléon sûr dans le modèle.* ♢

Démonstration. Nous allons à présent démontrer notre théorème de sécurité, pour cela nous devons montrer que notre schéma est Et-**Immuable**, **Transparent**, **Responsable** et **Traçable**.

Immuabilité étendue. Le schéma ne propose pas de limiter les messages dans un ensemble ou d'obliger certaines parties à rester égales entre elles. En conséquences V et E prennent leurs valeurs par défaut. Il n'existe donc pas d'adversaire capable d'exhiber un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$ tel que $\exists j$ tel que $\mathsf{m}_j^* \notin \mathcal{V}_j^*$ ou tel que $\exists k$ tel que $\exists j, j' \in \mathcal{E}_k$ tels que $\mathsf{m}_j^* \neq \mathsf{m}_{j'}^*$. Dans ce schéma prouver l'**Immuabilité étendue** revient à prouver l'**Immuabilité**. La preuve en elle-même est identique à la preuve d'**Immuabilité** de notre schéma de signature caméléon présenté au chapitre 6, section 7.
En conclusion, la probabilité de succès d'un adversaire contre l'**Immuabilité étendue** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda) = \mathsf{Succ}^{\mathcal{SCE}}_{\mathbf{Imm},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

□

Transparence. Nous utilisons le fait que les seules différences entre une signature originale et une signature modifiée résident dans la construction du TAG et des aléas de R. Plus formellement, nous utilisons une preuve par jeu. Les échanges entre l'adversaire $\mathcal{A}$ et les oracles sont simulés par un distingueur $\mathcal{D}$ qui cherche à casser l'**uniformité** des fonctions de hachage caméléon $\mathcal{HC}$ et CH ou le fait que $\mathcal{PRG}$ soit un générateur **pseudo-aléatoire**.

Jeu 0 : Le Jeu 0 correspond au jeu joué par l'adversaire dans l'expérience originale. Dans cette attaque nous nous intéressons à la forme du message obtenu grâce à la requête de challenge à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$ qui dépend du bit b : si $b = 0$, $\mathcal{D}$ retourne une paire signature originale Σ_0 avec $\mathsf{TAG}_0 = \mathcal{PRG}(x_0)$ et R_0 un ensemble d'aléas, si $b = 1$ il retourne une signature modifiée Σ_1 avec TAG_1 un aléa et R_1 les sorties des algorithmes $\mathcal{HC}.\textsc{Forge}$ et $f\mathcal{HC}.\textsc{Forge}$. À la fin de son attaque, l'adversaire retourne un bit b'. On note S_0 l'événement $b = b'$ dans le Jeu 0. On obtient :

$$\mathsf{Adv}^{\mathcal{SCE}}_{\mathbf{Transp},\mathcal{A}}(\lambda) = 2|Pr[S_0] - 1/2|.$$

Jeu 1 : Dans ce premier jeu, le distingueur modifie sa réponse aux requêtes à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$.

Dans le cas $b = 0$, le distingueur effectue une signature caméléon originale sur le message challenge mais en utilisant un x_0 choisit aléatoirement. Il retourne donc une signature Σ_0 avec TAG_0 construit à partir d'un aléa et R_0 un ensemble d'aléas. Le cas $b = 1$ reste inchangé.

La modification effectuée est indistinguable pour l'adversaire s'il n'est pas capable de distinguer un aléa d'une sortie de $\mathcal{PRF}$ tout en connaissant la graine Nonce_0. Si la modification est distinguable par l'adversaire, alors $\mathcal{D}$ peut utiliser cet adversaire contre la propriété de fonction **pseudo-aléatoire** de la $\mathcal{PRF}$. La différence entre les deux jeux est donc bornée de la façon suivante :

$$|Pr[S_1] - Pr[S_0]| \leq \mathsf{Adv}^{\mathcal{PRF}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda).$$

Jeu 2 : Dans le Jeu 2, le distingueur modifie, par rapport au Jeu 1, sa réponse à la requête à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$ par rapport au Jeu 1. Dans le cas $b = 0$, $\mathcal{D}$ effectue une signature caméléon originale sur le message challenge mais en utilisant cette fois un TAG_0 choisit aléatoirement. Il retourne donc une signature Σ_0 avec TAG_0 un aléa et R_0 un ensemble d'aléas. Le cas $b = 1$ reste inchangé.

La modification effectuée est indistinguable pour l'adversaire s'il n'est pas capable de distinguer un aléa d'une sortie de $\mathcal{PRG}$. Si la modification est distinguable par l'adversaire, alors $\mathcal{D}$ peut utiliser cet adversaire contre la propriété de générateur **pseudo-aléatoire** de $\mathcal{PRG}$. La différence entre les deux jeux est donc bornée de la façon suivante :

$$|Pr[S_2] - Pr[S_1]| \leq \mathsf{Adv}^{\mathcal{PRG}}_{\textbf{ps-aléa},\mathcal{A}}(\lambda).$$

Jeu 3 : Dans ce jeu, le distingueur modifie à nouveau sa réponse à la requête à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$: Dans le cas $b = 1$, le distingueur calcule une signature caméléon dans laquelle les aléas r_i pour tout i in ADM sont choisis aléatoirement. Les parties reconstituées correspondantes sont construites grâce à l'algorithme $f\mathcal{HC}.\textsc{Hache}$. Le message reconstitué additionnel est construit par collision comme dans une signature modifiée classique et TAG_1 est choisit aléatoirement. Il retourne donc une signature Σ_1 avec TAG_1 un aléa et R_1 composé d'un ensemble d'aléas et de r_c obtenu pour la collision. Le cas $b = 0$ reste inchangé.

La modification effectuée est indistinguable par l'adversaire s'il n'est pas capable de distinguer un aléa d'une sortie de $f\mathcal{HC}.\textsc{Forge}$. Si la modification est distinguable par l'adversaire, alors $\mathcal{D}$ peut utiliser cet adversaire contre l'**uniformité** du schéma de hachage caméléon CH. La différence entre les deux jeux est donc bornée de la façon suivante :

$$|Pr[S_3] - Pr[S_2]| \leq \mathsf{Adv}^{CH}_{\textbf{uni},\mathcal{A}}(\lambda).$$

Jeu 4 : Dans ce jeu, le distingueur modifie à nouveau sa réponse à la requête à l'oracle $\mathcal{O}.\textsc{Modif}/\textsc{Signe}_b(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \mathsf{V}, \mathsf{E}, \mathsf{maxMod}, \mathsf{maxV})$: Dans le cas $b = 1$, le distingueur calcule une signature caméléon originale sur le message challenge en utilisant un TAG_1 choisit aléatoirement. Il retourne donc une signature Σ_1 avec TAG_1 un aléa et R_1 un ensemble d'aléas. Le cas $b = 0$ reste inchangé.

La modification effectuée est indistinguable par l'adversaire s'il n'est pas capable de distinguer un aléa d'une sortie de $\mathcal{HC}.\textsc{Forge}$. Si la modification est distinguable par l'adversaire, alors $\mathcal{D}$ peut utiliser cet adversaire contre l'**uniformité** du schéma de

hachage caméléon $\mathcal{HC}$. La différence entre les deux jeux est donc bornée de la façon suivante :

$$|Pr[S_4] - Pr[S_3]| \leq \mathsf{Adv}^{\mathcal{HC}}_{\mathbf{uni},\mathcal{A}}(\lambda).$$

Or dans ce dernier jeu les réponses sont identiques quelle que soit la valeur de b. Donc, nous avons :

$$Pr[S_4] = 1/2.$$

Il est à présent possible d'évaluer l'avantage de l'adversaire contre la **Transparence** de notre schéma. Cet avantage est donné par :

$$\begin{array}{rcl} Adv^{\mathcal{SCE}}_{\mathbf{Transp},\mathcal{A}}(\lambda) & = & 2|Pr[S_0] - 1/2| \\ & = & 2|Pr[S_0] - Pr[S_4]| \\ & \leq & 2(|Pr[S_1]-Pr[S_0]|+|Pr[S_2]-Pr[S_1]| +|Pr[S_3]- \\ & & Pr[S_2]| + |Pr[S_4] - Pr[S_3]|) \\ & \leq & 2\mathsf{Adv}^{\mathcal{PRF}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) \quad + \quad 2\mathsf{Adv}^{\mathcal{PRG}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) \quad + \\ & & 2\mathsf{Adv}^{CH}_{\mathbf{uni},\mathcal{A}}(\lambda) \; + 2\mathsf{Adv}^{\mathcal{HC}}_{\mathbf{uni},\mathcal{A}}(\lambda). \end{array}$$

□

Responsabilité. La **Responsabilité** de ce schéma repose sur la partie supplémentaire h_c qui est identique à la version classique. La preuve de la **Responsabilité** de ce schéma est la même que la preuve correspondante pour le schéma initial présenté dans le chapitre 6, section 6.4. Nous ne répétons donc pas la preuve ici et proposons au lecteur curieux de se rapporter à la preuve correspondante décrite en section 6.3.
Le succès d'un adversaire contre la Délégué-**Responsabilité** de notre schéma est alors :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

Le succès d'un adversaire contre la Signataire-**Responsabilité** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathsf{sig}-\mathbf{Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{PRG}}_{\mathbf{OW},\mathcal{A}}(\lambda) + \mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda).$$

□

Traçabilité. Un adversaire $\mathcal{A}$ gagne l'expérience de la **Traçabilité** s'il exhibe une paire $(\mathsf{dpk}^*, \mathsf{pubDB}^* = \{(\mathsf{m}_k{}^*, \Sigma_k{}^*)\}_{k\in[1,n]})$ telle que $\mathcal{SCE}$.TestFraude$(\mathsf{spk}, \mathsf{dpk}^*, \mathsf{pubDB}^*)$ retourne $(\mathsf{usk}, \pi_{\mathsf{fraude}})$ et

1. soit usk ne correspond pas a dpk,
2. soit $\mathcal{SCE}$.VérifFraude$(\pi_{\mathsf{fraude}}, \mathsf{usk}) = \mathsf{faux}$

Pour le cas 1, $\mathcal{SCE}$.TestFraude teste si $g^{P(0)} = \mathsf{upk}$. Si c'est le cas, il pose $\mathsf{usk} = P(0)$. Or le Théorème 2 nous dit qu'il n'existe qu'un seul polynôme de degré maxMod passant par $\mathsf{maxMod} + 1$ points. Donc ce cas arrive avec une probabilité nulle.
Pour le cas 2, $\mathcal{SCE}$.VérifFraude retourne faux dans trois cas.

1. $\mathsf{upk} \neq g^{\mathsf{usk}}$ ou $P(0) \neq \mathsf{usk}$.
   Ce qui nous ramène au cas 1.
2. Les paires de la base de données contenues dans π_{fraude} sont mal formées.
   Or π_{fraude} est une sortie de l'algorithme $\mathcal{SCE}$.TestFraude. La **Consistance** implique que la probabilité de cet événement est nulle.
3. $\exists j$ tel que le $F(j)$ obtenue par l'attaque de [Ad04] et différent de $P(j)$. Or $P(y)$ est le polynôme d'interpolation passant par les points d'interpolation $(j, F(j))$. Par construction, cette événement à une probabilité nulle.

En conclusion, la probabilité de succès d'un adversaire contre la **Traçabilité** de ce schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda) = 0.$$

□

7.5 Constructions pour l'extension *LimNbVersion*

L'idée de la solution que j'ai proposé avec S.Canard dans [CJ10a] est d'utiliser une preuve de connaissance sans divulgation atteignant la **super-validité** (voir Définition 10).

Pour rappel, un tel protocole en trois passes fonctionne de la manière suivante. Le prouveur génère un engagement $b = g^a$ avec a un aléa et une valeur publique $y = g^x$ reliée au secret x. Le vérifieur pose une question c et le prouveur donne une réponse $d = a - c \cdot x$ reliée à l'aléa a, à la question c et au secret x. Le vérifieur teste si $b = g^d \cdot y^c$.

De plus, la **super-validité** de ces constructions assure qu'à partir d'un engagement b, si on a deux questions c et c', $c \neq c'$, et les réponses correspondantes d et d', alors on est capable de retrouver le secret x de la façon suivante :

$$x = \frac{d - d'}{c' - c}.$$

Enfin, dans un tel schéma n'importe qui peut générer un triplet (b, c, d) valide sans connaître x. Pour cela il suffit de choisir aléatoirement c et d puis calculer $b = g^d \cdot y^c$.

Notre schéma est basé sur l'idée qu'à chaque version d'une signature on associe un triplet (b, c, d) sur la clé secrète d'utilisateur $x = \mathsf{usk}$. Lors de son initialisation, le délégué génère un ensemble d'engagements $\{b_j = g^{a_j}\}$. Lorsque le signataire effectue une signature, il utilise la remarque que nous avons fait plus haut pour générer un triplet (b_0, c_0, d_0) valide sans connaître le secret x.

La valeur c est ensuite utilisée comme aléa pour la génération du condensé caméléon d'une partie additionnelle correspondant au message entier. Et la réponse sera ajoutée à la signature caméléon originale.

Le signataire choisit maxV engagements $\{b_j\}_{j\in[1,\mathsf{maxV}]}$ parmi les engagements publiés par le délégué. Il accumule les $\mathsf{maxV} + 1$ engagements dans un accumulateur Acc qui sera ajouté au message reconstitué qui est signé par le signataire. Ainsi on ne peut pas ajouter de nouveaux engagements.

Lors d'une modification par le délégué, celui-ci doit répondre à la question c qui correspond à la valeur retourné lors de la génération d'une collision sur le condensé caméléon du message modifié. Afin qu'on ne puisse pas retrouver sa clé secrète d'utilisateur, il utilise un engagement différent à chaque nouvelle version.

Lorsque le délégué veut dépasser le nombre de versions autorisées il est obligé d'utiliser un engagement qu'il a déjà utilisé. Comme par définition il souhaite modifier le message en une nouvelle version, la question est différente. Et il est publiquement possible de retrouver la clé secrète de l'utilisateur en utilisant la **super-validité** du protocole en trois passes (voir Définition 10).

7.5.1 Notre construction

Plus formellement, notre schéma fonctionne de la manière suivante.

Construction 24.
Solution *LimNbVersion* Ce schéma utilise un schéma de signature **EU-CMA** que nous notons $\mathcal{S}$ (cf. Définition 29), un schéma de fonction de hachage caméléon $\mathcal{HC}$ **uniforme** et **résistant aux collisions** (cf. Définition 22), un schéma de fonction pseudo aléatoire $\mathcal{PRF}$ (cf. Définition 24) et un schéma de générateur pseudo-aléatoire $\mathcal{PRG}$ (cf. Définition 23).

$\mathcal{SCE}$.Init(1^λ). Cet algorithme prend en entrée 1^λ avec λ un paramètre de sécurité et initialise le système. Ce qui correspond à l'exécution de $\mathcal{HC}$.Init(λ) pour obtenir les paramètres de la fonction caméléon $\mathcal{HC}$.param et de $\mathcal{S}$.Init(λ) pour les paramètres du schéma de signature $\mathcal{S}$.param. On choisit ensuite p et q deux nombres premiers tels que $p-1$ soit divisible par q. Enfin on choisit g un générateur de $\mathbb{Z}_p$. On note $\mathcal{SCE}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, \mathcal{S}.\mathsf{param}, p, q, g\}$.

$$\mathcal{SCE}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, \mathcal{S}.\mathsf{param}, p, q, g\} \leftarrow \mathcal{SCE}.\textsc{Init}(1^\lambda).$$

$\mathcal{SCE}$.SigGénClé($\mathcal{SCE}$.param). Cet algorithme exécute l'algorithme de génération des clés du schéma de signature $\mathcal{S}$.GénClé(1^λ) et obtient $(\mathsf{pk}, \mathsf{sk})$. Il choisit ensuite κ aléatoirement dans $\{0,1\}^\lambda$.

$$(\mathsf{spk} = \mathsf{pk}, \mathsf{ssk} = \{\mathsf{sk}, \kappa\}) \leftarrow \mathcal{SCE}.\textsc{SigGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.SanGénClé($\mathcal{SCE}$.param). L'algorithme exécute l'algorithme de génération des clés du schéma de fonction de hachage caméléon $\mathcal{HC}$.GénClé(1^λ) et obtient la paire de clés $(\mathsf{csk}, \mathsf{cpk})$ pour $\mathcal{HC}$. L'algorithme choisit ensuite aléatoirement dans $\mathbb{Z}_q$ la clé secrète d'utilisateur usk du délégué et construit la valeur publique correspondante en utilisant le générateur g : $y = g^{\mathsf{usk}}$. Il choisit ensuite $z \geq \mathsf{maxV}$ aléas dans $\mathbb{Z}_p$ $a_1, \cdots, a_z$ et calcule, pour tout $j \in [1, x]$ la valeur $b_j = g^{a_j} \bmod p$. Il obtient ainsi la paire de clés de délégué $(\mathsf{dsk} = \{\mathsf{csk}, a_1, \cdots, a_z\}, \mathsf{dpk} = \{\mathsf{cpk}, y, b_1, \cdots, b_x\})$.

$$(\mathsf{dsk}, \mathsf{dpk}, \mathsf{usk}) \leftarrow \mathcal{SCE}.\textsc{SanGénClé}(\mathcal{SCE}.\mathsf{param})$$

$\mathcal{SCE}$.Signe(m, ssk, dpk, ADM, maxV). Dans un premier temps, le signataire génère un TAG pour son message. Il choisit aléatoirement $\mathsf{Nonce} \in \{0,1\}^\lambda$, calcule $\mathsf{x} = \mathcal{PRF}(\kappa, \mathsf{Nonce})$ et obtient $\mathsf{TAG} = \mathcal{PRG}(\mathsf{x})$.
Le signataire choisit aléatoirement c_0 et d_0 et construit son numéro de version.

$$b_0 = y^{c_0} \cdot g^{d_0} \bmod p.$$

Il accumule ensuite les numéros de versions $b_0, \cdots, b_{\mathsf{maxV}}$ dans un accumulateur Acc et obtient les témoins correspondants $\mathcal{T} = \mathsf{t}_0, \cdots, \mathsf{t}_{\mathsf{maxV}}$:
– $((\mathsf{ask}, \mathsf{apk}), \mathsf{Acc}_0, \mathsf{etat}_0, \mathsf{aux}) \leftarrow \mathcal{ACC}.\textsc{Gén}(\lambda, \mathsf{maxV}+1)$
– $(\mathsf{Acc}, \mathcal{T}, \mathsf{etat}) \leftarrow \mathcal{ACC}.\textsc{AjouteEns}(\mathsf{ask}, \mathsf{Acc}_0, \{b_i\}_{i\in[0,\mathsf{maxV}]}, \mathsf{etat}_0, \mathsf{aux})$
Le signataire note $\mathsf{s} = \{(\mathsf{t}_i)\}_{i\in[1,\mathsf{maxV}]}$ avec t_i le témoin correspondant à la valeur b_i. Il choisit $|\mathsf{ADM}|$ aléas $\{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}\}$ dans $\{0,1\}^\lambda$. Nous notons : $\mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}, c_0\}$.
Le signataire exécute alors une *procédure de reconstruction* sur le message m.

Procédure de reconstruction :
Cette procédure publique utilise TAG, les aléas R, la clé publique du délégué dpk et celle du signataire spk :

1. Pour chaque partie, on calcule un message reconstitué $\tilde{\mathsf{m}}_i$.
$$\forall i \in [1, \mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}_i||i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i||i & \text{sinon} \end{cases}$$
2. Puis on calcule une partie finale supplémentaire.
$\mathsf{h}_c = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}\|\mathsf{m}\|\mathsf{spk}, c_0)$

Le message reconstitué est $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1\| \cdots \|\tilde{\mathsf{m}}_t\|\mathsf{h}_c\|\mathsf{dpk}\|\mathsf{Acc}$.

Le signataire signe le message reconstitué $\sigma = \mathcal{S}.\textsc{Signe}(\mathsf{spk}, \mathsf{ssk}, \tilde{\mathsf{m}})$. Et il obtient la signature caméléon $\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}, b_0, c_0, d_0, \mathsf{Acc}, \mathsf{t}_0)$ avec $\mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}\}$ et t_0 le témoin correspondant à b_0. La signature et le message correspondant sont ajoutés à la base de données du signataire BD.

$$(\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}, b_0, c_0, d_0, \mathsf{Acc}, \mathsf{t}_0), \mathsf{s}) \leftarrow (\mathsf{m}, \mathsf{ssk}, \mathsf{dpk}, \mathsf{ADM}, \mathsf{maxV}).$$

$\textsc{Sanitize}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \mathsf{maxV}, \mathsf{s})$. Le délégué souhaite à présent transformer le message m en fonction de MOD, c'est-à-dire en le message m'. Il vérifie d'abord que MOD correspond à ADM et aux limitations, si ce n'est pas le cas il s'arrête en retournant $\perp$. Il met à jour les tailles ℓ_i des parties qui vont être modifiées puis choisit aléatoirement Nonce' et TAG' dans $\{0,1\}^\lambda$. Le délégué effectue la *procédure de reconstruction* comme décrit précédemment et obtient $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_t \| \mathsf{h}_c \| \mathsf{dpk} \| \mathsf{Acc}$ et donc les valeurs h_c et $\tilde{\mathsf{m}}_i = \mathsf{h}_i$ pour tout $i \in \mathsf{MOD} \subset \mathsf{ADM}$. Il construit les collisions sur celles-ci grâce à sa clé secrète de délégué et à l'algorithme $\mathcal{HC}.\textsc{Forge}$ de la manière suivante.

- Pour tout $i \in \mathsf{MOD}$ le délégué calcule $\mathsf{r}'_i = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, \mathsf{m}_i \| i, \mathsf{m}'_i \| i, \mathsf{r}_i, \tilde{\mathsf{m}}_i)$,
- puis $c_j = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, \mathsf{TAG}' \| \mathsf{m}' \| \mathsf{spk}, c, \mathsf{h}_c)$.

Le délégué choisit un nouveau $b_j = g^{a_j}$ et calcule :

$$d_j = a_j - c_j \mathsf{usk}$$

Enfin, il construit l'ensemble $\mathsf{R}' = \{\mathsf{r}'_1, \cdots, \mathsf{r}'_u, c_j\}$ en prenant $\mathsf{r}'_i = \mathsf{r}_i$ si $i \notin \mathsf{MOD}$ et $\mathsf{r}'_i = \mathsf{r}'_i$ sinon. En notant t_j le témoin correspondant à b_j, la signature modifiée est $\Sigma' = (\sigma, \mathsf{TAG}', \mathsf{Nonce}', \mathsf{R}', \mathsf{ADM}', b_j, c_j, d_j, \mathsf{Acc}, \mathsf{t}_j)$.

$$(\Sigma', \mathsf{m}') \leftarrow \textsc{Sanitize}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dsk}, \mathsf{MOD}, \mathsf{maxV}, \mathsf{s}).$$

$\mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{spk}, \mathsf{dpk})$. Le vérifieur s'assure que $b = \mathsf{upk}^c \cdot g^d$. Si ce n'est pas le cas, il retourne faux. Il exécute la *procédure de reconstruction* comme décrit plus haut et obtient le message reconstitué $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_t \| \mathsf{h}_c \| \mathsf{dpk} \| \mathsf{Acc}$. Enfin il retourne la sortie de l'algorithme de vérification sur le message $\mathcal{S}.\textsc{Vérifie}(\mathsf{spk}, \sigma, \tilde{\mathsf{m}})$.

$\mathcal{SCE}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{ssk}, \mathsf{dpk}, \mathsf{BD} = (\mathsf{m}_k, \Sigma_k)_{k \in [1,\mathsf{q}]})$. Le signataire cherche un indice $j \in [1, \mathsf{q}]$ dans la base de données $\mathsf{BD} = (\mathsf{m}_k, \Sigma_k)_{k \in [1,\mathsf{q}]}$ tel que :

- $\mathsf{vrai} = \mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
- $\mathsf{m}_j \neq \mathsf{m}$;
- $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}_j \| \mathsf{m}_j \| \mathsf{spk}, c_j)$ avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.

Si il existe un tel indice j alors il obtient la preuve $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ avec $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j, b_j, c_j, d_j, \mathsf{Acc}_j, \mathsf{t}_j)$ sinon $\pi_{\mathsf{or/mod}} = \perp$.

$\mathcal{SCE}.\textsc{Juge}(\mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}})$. Si $\pi_{\mathsf{or/mod}} = \perp$ alors le juge retourne Signataire sinon il vérifie que la preuve $\pi_{\mathsf{or/mod}} = (\mathsf{spk}, \mathsf{m}_j, x_j, \Sigma_j)$ respecte les propriétés suivantes :

- $\mathsf{vrai} = \mathcal{SCE}.\textsc{Vérifie}(\mathsf{m}_j, \Sigma_j, \mathsf{spk}, \mathsf{dpk})$;
  On peut donc poser $\Sigma_j = (\sigma_j, \mathsf{TAG}_j, \mathsf{Nonce}_j, \mathsf{R}_j, \mathsf{ADM}_j, b_j, c_j, d_j, \mathsf{Acc}_j, \mathsf{t}_j)$.
- $\mathsf{m}_j \neq \mathsf{m}$;
- $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG} \| \mathsf{m} \| \mathsf{spk}, c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}_j \| \mathsf{m}_j \| \mathsf{spk}, c_j)$ avec $\mathsf{TAG}_j = \mathcal{PRG}(x_j)$.

Si la preuve $\pi_{\mathsf{or/mod}}$ respecte toutes les propriétés, il retourne Délégué sinon il retourne Signataire.

$\mathcal{SCE}.\textsc{TestFraude}(\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB})$ S'il le délégué a fraudé, alors il a été obligé d'utiliser deux fois le même b_i. Le testeur recherche donc dans la base de données des messages

signés d'une même instance pubDB deux messages signés valides m_j, m_l tels que $\mathsf{m}_j \neq \mathsf{m}_l$ et $b_j = b_l = g^{a_i}$. S'il ne trouve aucune paire de la sorte il retourne $\perp$. Sinon il retrouve la clé d'utilisateur du délégué de la façon suivante.

$$\mathsf{usk} = \frac{d_j - d_l}{c_l - c_j}$$

La preuve est $\pi_{\mathsf{fraude}} = (\mathsf{spk}, \mathsf{dpk}, (\mathsf{m}_j, \Sigma_j), (\mathsf{m}_l, \Sigma_l))$.

$$(\pi_{\mathsf{fraude}} = (\mathsf{spk}, \mathsf{dpk}, (\mathsf{m}_j, \Sigma_j), (\mathsf{m}_l, \Sigma_l)), \mathsf{usk}) \leftarrow \mathcal{SCE}.\textsc{TestFraude}(\mathsf{spk}, \mathsf{dpk}, \mathsf{pubDB})$$

$\mathcal{SCE}.\textsc{VérifFraude}(\pi_{\mathsf{fraude}}, \mathsf{usk})$ Le vérifieur s'assure que la clé secrète usk correspond à la clé publique de l'utilisateur $y = g^{\mathsf{usk}}$ et si $b_j = b_l = g^{a_i}$, si ce n'est pas le cas il retourne faux. Il vérifie ensuite la preuve. Il vérifie que (m_j, Σ_j) et (m_l, Σ_l) sont deux paires valides et appartiennent à la même instance (si elles ont toutes les deux le même σ) et retourne faux si ce n'est pas le cas. Enfin il vérifie si $\mathsf{usk} = \frac{d_j - d_l}{c_l - c_j}$ et retourne faux si ce n'est pas le cas et vrai sinon.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SCE}.\textsc{VérifFraude}(\pi_{\mathsf{fraude}}, \mathsf{dsk})$$

$\circlearrowleft$

7.5.2 Preuve de sécurité de notre schéma

Théorème 10. *Sous l'hypothèse que le schéma de signature $\mathcal{S}$ soit **EU-CMA** sûr, que les fonctions $\mathcal{PRF}$ et PRG choisies soient pseudo-aléatoires, que la fonction caméléon $\mathcal{HC}$ soit **uniforme** et **résistant aux collisions** et que le schéma d'accumulateur $\mathcal{ACC}$ soit **résistant aux collisions**, notre schéma est un schéma de signature caméléon sûr dans le modèle.* $\diamondsuit$

Démonstration. Nous allons à présent démontrer que notre théorème de sécurité, pour cela nous devons montrer que notre schéma est Et-**Immuable**, **Transparent**, **Responsable** et **Traçable**.

Immuabilité étendue. Le schéma ne propose pas de solution pour limiter les messages dans un ensemble ou pour obliger certaines parties à rester égales entre elles. En conséquences V et E prennent leurs valeurs par défaut. Il n'existe donc pas d'adversaire capable d'exhiber un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$ tel que $\exists j$ tel que $\mathsf{m}_j^* \notin \mathcal{V}_j^*$ ou tel que $\exists l$ tel que $\exists j, j' \in \mathcal{E}_l$ tels que $\mathsf{m}_j^* \neq \mathsf{m}_{j'}^*$. Dans ce cas prouver l'**Immuabilité étendue** revient à prouver l'**Immuabilité**. La preuve en elle-même est identique à la preuve d'**Immuabilité** de notre schéma de signature caméléon présenté au chapitre 6, section 7. Nous invitons le lecteur à s'y rapporter.

La probabilité de succès d'un adversaire contre l'**Immuabilité étendue** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{Et-\mathbf{Imm},\mathcal{A}}(\lambda) = \mathsf{Succ}^{\mathcal{SCE}}_{\mathbf{Imm},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

$\square$

Transparence. Les seules différences entre une signature originale et une signature modifiée résident dans la construction du TAG et des aléas de R. En effet les témoins utilisés dans une signature originale sont identiques à ceux utilisés par le délégué et la relation entre un b_j, un d_j et un c_j est la même dans les deux cas.

La preuve que ce schéma est transparent est alors identique à la preuve de **Transparence** de notre schéma de signature caméléon présenté au chapitre 6, section 7. Le lecteur pourra

s'y reporter.
La probabilité de succès d'un adversaire contre la **Transparence** de notre schéma est :

$$Adv^{\mathcal{SCE}}_{\mathbf{Transp},\mathcal{A}}(\lambda) \leq 2\mathsf{Adv}^{\mathcal{PRF}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) + 2\mathsf{Adv}^{\mathcal{PRG}}_{\mathbf{ps\text{-}aléa},\mathcal{A}}(\lambda) + 2\mathsf{Adv}^{\mathcal{HC}}_{\mathbf{uni},\mathcal{A}}(\lambda)$$

□

Responsabilité. La **Responsabilité** de ce schéma repose sur la partie supplémentaire h_c. Or cette partie est identique entre la version classique et la version étendue du schéma. La preuve de la **Responsabilité** de ce schéma se déroule exactement de la même manière que la preuve correspondante pour le schéma initial présenté dans le chapitre 6, section 6.4. Nous ne répétons donc pas la preuve ici et proposons au lecteur de se rapporter à la preuve correspondante décrite en section 6.3. La probabilité de succès d'un adversaire contre la Délégué-**Responsabilité** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathsf{del}-\mathbf{Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{S}}_{\mathbf{EU\text{-}CMA},\mathcal{A}}(\lambda).$$

Et la probabilité de succès d'un adversaire contre la Signataire-**Responsabilité** de notre schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathsf{sig}-\mathbf{Resp},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{PRG}}_{\mathbf{OW},\mathcal{A}}(\lambda) + \mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda).$$

□

Traçabilité. Un adversaire $\mathcal{A}$ gagne l'expérience de la **Traçabilité** s'il exhibe une paire $(\mathsf{dpk}^*, \mathsf{pubDB}^*)$ avec $\mathsf{pubDB}^* = \{(\mathsf{m}_k{}^*, \Sigma_k{}^*)\}_{k\in[1,n]}$ telle que
$(\mathsf{usk}, \pi_{\mathsf{fraude}}) \longleftarrow \mathcal{SCE}.\textsc{TestFraude}(\mathsf{spk}, \mathsf{dpk}^*, \mathsf{pubDB}^*)$ et

1. soit usk ne correspond pas a dpk,
2. soit $\mathsf{faux} \leftarrow \mathcal{SCE}.\textsc{VérifFraude}(\pi_{\mathsf{fraude}}, \mathsf{usk})$.

Dans le cas 1, nous pouvons utiliser un adversaire gagnant ce jeu contre le protocole d'identification de Schnorr à vérifieur honnête (voir Définition 10). En effet, les b_j correspondent à un engagement sur le secret usk, les c_j aux challenges et les d_j aux réponses. Le fait que $y \neq g^{\mathsf{usk}}$ implique que le prouveur du protocole de Schnorr peut changer de secret. Nous avons $\mathsf{Succ}^{Schnorr}_{eng,\mathcal{A}}$ le succès d'un tel adversaire contre le protocole d'identification de Schnorr. Dans le cas 2, $\mathcal{SCE}.\textsc{VérifFraude}$ retourne faux dans les 3 cas suivants.

1. $y \neq g^{\mathsf{usk}}$. Ce qui nous ramène au cas 1.
2. si les paires de la base de données contenues dans π_{fraude} sont mal formées.
   Or π_{fraude} est une sortie de l'algorithme $\mathcal{SCE}.\textsc{TestFraude}$. Donc par la **Consistance** de l'algorithme nous obtenons que la probabilité de cet événement est nulle.
3. ou si $\mathsf{usk} \neq \frac{d_j - d_l}{c_l - c_j}$. Or $\mathcal{SCE}.\textsc{TestFraude}$ étant correct, cela nous ramène au cas 1.

En conclusion, la probabilité de succès d'un adversaire contre la **Traçabilité** de ce schéma est :

$$\mathsf{Succ}^{\mathcal{SCE}}_{\mathbf{Tra},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{Schnorr}_{eng,\mathcal{A}}.$$

□

Conclusion

J'ai décrit dans ce chapitre les travaux, effectués avec S.Canard, sur les signatures caméléons étendues. Ceux-ci comporte tout d'abord une extension du modèle de Brzuska *et al.* permettant,

pour la première fois, d'évaluer la sécurité de signatures caméléons étendues pour quatre extensions. Les trois premières sont issues de propositions de [KL06]. Elles permettent d'encadrer les modifications dans un ensemble de valeurs autorisées, obliger une même modification sur plusieurs blocs de messages et, enfin, limiter le nombre de blocs modifiables. La dernière extension considérée dans notre modèle considère la limitation du nombre de versions de signature modifiée.

Ce nouveau modèle, incluant le modèle de Bruzka *et al.* nous a permis d'analyser les propositions d'extensions de la littérature et de relever leurs faiblesses. Nous avons ensuite proposé une réparation ou une nouvelle solution pour chacune des quatre extensions considérées.

Ces travaux ont été publiées avec S. Canard à la conférence CT-RSA 2010 [CJ10a].

La solution que nous proposons dans le cas où nous souhaitons conserver la même valeur sur plusieurs blocs de message est efficace, mais peu esthétique. De plus, celle-ci considère que le vérifieur peut apprendre quels blocs doivent être identiques entre eux. Il serait intéressant de travailler à une nouvelle solution permettant de garder le vérifieur dans l'ignorance.

A présent, nous allons nous intéresser aux situations où le signataire souhaite l'anonymat d'un groupe : les signatures de groupe caméléons.

Chapitre 8

Signatures de groupe caméléons

Sommaire

Nous allons nous intéresser dans ce chapitre aux signatures de groupe caméléons. L'idée est d'étendre notre travail sur les signatures caméléons au cas des signatures de groupe, c'est-à-dire d'obtenir un schéma de signature de groupe (voir Chapitre 2 Section 2.3.2) permettant au signataire d'un message de désigner un délégué qui pourra par la suite modifier certaines parties du message signé tout en gardant une signature valide du groupe sur le nouveau message.

Les travaux que je présente dans ce chapitre sont issus d'une collaboration avec S.Canard ayant donné lieu à un article en cours de soumission [CJ], voir Annexe B.

8.1 Concepts des signatures de groupe caméléons

Un schéma de signature de groupe caméléon (noté $\mathcal{SGC}$) est composé d'un ensemble U d'utilisateurs u_i, d'un gestionnaire de groupe GG, d'une autorité d'ouverture AO, d'un délégué et d'un vérifieur. Il permet aux utilisateurs de signer des messages en tant que membre du groupe de telle sorte que :

- les signatures soient **anonymes** (excepté pour l'autorité d'ouverture) ;
- l'autorité d'ouverture puisse révoquer l'anonymat d'une signature à tout moment ;

– le délégué (qui n'est pas nécessairement un membre du groupe) puisse modifier les parties indiquées comme modifiable par le signataire tout en gardant une signature de groupe s'ouvrant sur le signataire initial.

Pour qu'un schéma de signature soit considéré comme un schéma de signature de groupe, il doit être **consistant** et respecter les propriétés d'**Anonymat**, de **Traçabilité** et de **Non-diffamation** comme définies au Chapitre 2. Sachant que nous souhaitons obtenir une signature **anonyme** telle que l'autorité d'ouverture puisse correctement révoquer l'anonymat, nous devons conserver l'essence de toutes ces propriétés.

Nous avons défini au Chapitre 6 les signatures caméléons comme des signatures respectant les propriétés d'**Infalsifiabilité**, d'**Immuabilité**, d'**Intimité**, de **Transparence** et de **Responsabilité**. Les deux premières propriétés, **Infalsifiabilité** et **Immuabilité** assurent l'impossibilité de falsifier une signature et garantissent que les paires modifiées émises par le délégué respectent les limites fixées par le signataire. Le principe de ces deux propriétés n'est pas affecté par le fait que les signataires sont considérés comme des membres anonymes d'un groupe.

Par contre, les trois dernières notions, **Transparence**, **Intimité** et **Responsabilité** posent la question de l'importance que l'on souhaite mettre sur le concept de groupe dans nos signatures et donc sur l'objectif recherché par sa mise en place.

La notion représentée par la **Responsabilité** des signatures caméléon suppose qu'une partie ne peut faussement accuser l'autre partie d'être l'auteur d'une paire message-signature. Cependant cette propriété n'assure pas qu'une partie soit incapable d'endosser la responsabilité d'une paire qu'elle n'a pas générée. Dans le contexte des signatures de groupe, cela implique qu'un signataire pourrait faire endosser au groupe la responsabilité de paires modifiées. L'ouverture de ces signatures retournant l'identité du signataire, il endossera tout de même la responsabilité finale.

La notion d'**Intimité**, qui assure l'impossibilité, pour un tiers, de retrouver le message original à partir d'une paire message-signature modifiée, est inclue dans la notion plus forte de **Transparence** qui assure qu'un tiers est incapable de distinguer une paire originale d'une paire modifiée. Dans le cas des signatures de groupe caméléon ce tiers peut être défini soit comme une entité qui n'est ni le signataire ni le délégué impliqués dans une signature donnée soit comme une entité qui n'appartient ni au groupe des signataires ni au délégué. Enfin, si l'objectif est "simplement" d'obtenir des signatures de groupe modifiable par un tiers, il est possible d'ignorer la notion de **Transparence** et de ne considérer que celle d'**Intimité**.

Ces différents choix dépendent finalement de l'application et de la modélisation considérée.

S.Canard et moi avons travaillé sur les trois hypothèses suivantes.

1. Une signature de groupe caméléon pour laquelle il est publiquement possible de distinguer une signature originale d'une signature modifiée. Ce cas correspond aux applications pour lesquelles la **Transparence** n'est pas nécessaire et où l'important est uniquement d'obtenir une signature de groupe "modifiable".

   Deux sous-hypothèses sont à considérer en fonction de la présence ou de l'absence de la propriété d'**Intimité** définie, ici, comme l'impossibilité pour tout individu, autre que le signataire, de retrouver le message original à partir du message modifié.

   Nous appelons ce type de schéma, que nous notons $\mathcal{NT}$-$\mathcal{SGC}$, un schéma de Signature de Groupe Caméléon Non-Transparent avec/sans **Intimité**.

2. Une signature de groupe caméléon pour laquelle la **Transparence** repose sur une conception de groupe. Dans ce cas, le délégué ainsi que tout membre du groupe est capable de distinguer une signature originale d'une signature modifiée. Cependant, toute personne extérieure au système est incapable de faire cette distinction. L'**Intimité** inclue par cette

notion de **Transparence** considère que tout membre n'appartenant pas au groupe est dans l'incapacité de retrouver le message original à partir d'une paire modifiée.

Nous appelons ces schémas des Signatures de Groupe Caméléon à Transparence de Groupe que nous notons $\mathcal{GT}$-$\mathcal{SGC}$.

3. Enfin, une signature de groupe caméléon pour laquelle seuls le signataire et le délégué sont capables de distinguer une signature originale d'une signature modifiée. Ce cas est le plus restrictif en ce qui concerne la **Transparence** et l'**Intimité**. Dans ce cas, seul le signataire impliqué peut répondre correctement à une accusation. Ainsi, il nous faut nous assurer que le signataire contacté pour une paire donnée est bien le signataire original. Nous considérons donc que l'anonymat du signataire est levé avant de résoudre tout problème de responsabilité d'une signature.

   Ces schémas seront appelés Signature de Groupe Caméléon à Transparence Forte que nous notons $\mathcal{FT}$-$\mathcal{SGC}$.

Les six premières propriétés (**Anonymat**, **Traçabilité** et **Non-diffamation** pour les propriétés issues des signatures de groupe et **Infalsifiabilité**, **Immuabilité** et **Responsabilité** pour celles issues des signatures caméléons) sont rassemblées dans ce chapitre en quatre propriétés AO**Protection**, u**Protection**, del**Protection**et **Anonymat** qui assurent, respectivement, la protection de l'autorité d'ouverture, celle de l'utilisateur, celle du délégué et enfin l'anonymat des utilisateurs dans le groupe.

Enfin, chaque hypothèse implique une vision distincte des propriétés de transparence et d'intimité. Nous détaillerons dans ce chapitre chacune de ces possibilités.

8.1.1 Définitions

Nous allons à présent définir ce qu'est une signature de groupe caméléon. Comme nous étudions ici trois possibilités très différentes, et afin de ne pas se répéter, je commencerai par définir une base commune aux trois possibilités avant de définir les particularités de chacune. Nous utilisons les variables ADM et MOD comme définies au Chapitre 6 en Définitions 43 et 44. Pour rappel, la variable ADM définit les modifications autorisées par le signataire sur un message qu'il a signé et MOD définit les modifications que le délégué souhaite effectuer sur un message.

Définition 58 (*Base* d'une Signature de Groupe Caméléon).
La *base* d'un schéma de signature de groupe caméléon, noté $\mathcal{SGC}$, est définie par les sept algorithmes suivants.

- $\mathcal{SGC}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système comprenant la clé publique du groupe gpk, la clé secrète du gestionnaire de groupe ggsk, la clé secrète de l'autorité d'ouverture osk ainsi que les paramètres généraux du système $\mathcal{SGC}.\mathsf{param}$. Nous considérons par la suite que λ et les paramètres $\mathcal{SGC}.\mathsf{param}$ sont inclus dans la clé publique du groupe gpk.

$$(\mathsf{gpk}, \mathsf{ggsk}, \mathsf{osk}) \leftarrow \mathcal{SGC}.\textsc{Init}(1^\lambda)$$

- $\mathcal{SGC}$.SanGénClé génère la paire de clés du délégué $(\mathsf{dpk}, \mathsf{dsk})$ en fonction de la clé publique du système gpk.

$$(\mathsf{dpk}, \mathsf{dsk}) \leftarrow \mathcal{SGC}.\textsc{SanGénClé}(\mathsf{gpk})$$

- $\mathcal{SGC}$.Rejoins est un protocole interactif entre le gestionnaire de groupe GG et un utilisateur u_i qui souhaite rejoindre le groupe. L'utilisateur prend éventuellement en entrée une paire de clés personnelles $(\mathsf{psk}_i, \mathsf{ppk}_i)$ tandis que le gestionnaire de groupe prend en entrée sa clé

secrète ggsk. À la fin du protocole, l'utilisateur u_i obtient une clé secrète usk_i d'utilisateur du groupe et un certificat d'appartenance au groupe cer_i. Le gestionnaire de groupe ajoute une entrée $\mathsf{reg}[i]$ dans la table d'enregistrement reg. Nous considérons dans la suite que la table d'enregistrement reg fait partie de la clé publique du groupe gpk.

$$(\mathsf{u}_i : \mathsf{usk}_i, \mathsf{cer}_i; \mathsf{GG} : \mathsf{reg}[i]) \leftarrow \mathcal{SGC}.\textsc{Rejoins}(\mathsf{u}_i : (\mathsf{psk}_i, \mathsf{ppk}_i), \mathsf{gpk}, ; \mathsf{GG} : \mathsf{gpk}, \mathsf{ggsk})$$

– $\mathcal{SGC}.\textsc{Signe}$ est l'algorithme de signature. Il prend en entrées un message m de taille ℓ qui est composé de t parties, la clé secrète et le certificat de l'utilisateur $(\mathsf{usk}_i, \mathsf{cer}_i)$, la clé publique du groupe gpk et la clé publique du délégué dpk. Enfin, il prend en entrée la variable ADM qui permet au signataire d'indiquer les parties qui pourront être modifiées par le délégué ainsi que la taille des différentes parties du message (cf. Définition 43 page 83). L'algorithme retourne une signature Σ sur le message m (ou $\perp$ en cas d'erreur). Nous considérons dans la suite que la variable ADM est incluse dans la signature Σ.

$$\Sigma \leftarrow \mathcal{SGC}.\textsc{Signe}(\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{dpk}, \mathsf{gpk}, \mathsf{ADM})$$

– $\mathcal{SGC}.\textsc{Modifie}$ utilise un message m et sa signature caméléon Σ, la clé publique du groupe gpk, la clé secrète du délégué dsk ainsi qu'une variable MOD décrivant les modifications voulues sur le message m (cf. Définition 44 page 83). L'algorithme retourne une nouvelle signature Σ' sur le message m' modifiée en fonction de MOD (ou $\perp$ en cas d'erreur).

$$(\Sigma', \mathsf{m}') \leftarrow \mathcal{SGC}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dsk}, \mathsf{MOD})$$

– $\mathcal{SGC}.\textsc{Vérifie}$ permet de vérifier une signature Σ sur un message m en utilisant la clé publique du délégué dpk et celle du groupe gpk. Il retourne vrai si la signature est valide et faux si elle ne l'est pas.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SGC}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk})$$

– $\mathcal{SGC}.\textsc{Ouvre}$ permet à l'autorité d'ouverture de retrouver l'identité du signataire original d'une paire message-signature, c'est-à-dire l'identité du signataire ayant signé la paire originale correspondante. Pour cela, il prend la paire (m, Σ) dont il veut connaitre l'origine, la clé secrète de l'autorité d'ouverture osk et les clés publiques du groupe gpk et du délégué dpk. Il retourne l'identité du signataire u_i et une preuve π_u que c'est bien lui qui est le signataire de la paire originale. En cas d'erreur, l'algorithme retourne $\perp$.

$$(\mathsf{u}_i, \pi_\mathsf{u}) \leftarrow \mathcal{SGC}.\textsc{Ouvre}(\mathsf{m}, \Sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$$

Remarque. Le cas où $\mathcal{SGC}.\textsc{Ouvre}$ retourne $\perp$ comprend tout types d'erreurs notamment le fait de ne pas trouver de signataire (ou pas dans le groupe) pour un couple message-signature donné.

– $\mathcal{SGC}.\textsc{OuvreJuge}$ est un algorithme public permettant de vérifier, à l'aide d'une preuve générée par l'autorité d'ouverture, si l'identité du signataire de la paire originale qui correspond à la paire message-signature (m, Σ) est bien l'utilisateur suspecté. Pour cela, il prend en entrées la paire concernée (m, Σ), la preuve π_u provenant, a priori, de l'algorithme $\mathcal{SGC}.\textsc{Ouvre}$ ainsi que l'identité u_i de l'utilisateur suspecté. L'algorithme retourne vrai si u_i est bien l'auteur original du couple (m, Σ) et faux sinon.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SGC}.\textsc{OuvreJuge}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u})$$

*

Cette définition correspond à la base d'une signature de groupe caméléon et n'est donc pas suffisante pour définir une signature complète. Il manque, notamment, le(s) algorithme(s) permettant à un Juge de retrouver l'auteur d'une paire contestée.

Nous avons défini en introduction les trois hypothèses auxquelles nous nous intéressons ici. Chacune implique un ensemble d'algorithme(s) spécifique(s). Nous donnons donc à présent ces trois ensembles.

Définition 59 (Signature de Groupe Caméléon Non-Transparente).
Un schéma de signature de groupe caméléon non-transparent, noté $\mathcal{NT}$-$\mathcal{SGC}$, est composé d'une base, comme spécifiée en Définition 58, et de l'algorithme public suivant qui permet de distinguer une signature originale d'une signature modifiée.

– $\mathcal{SGC}$.Groupe-Juge est un algorithme public permettant de vérifier l'origine d'une paire message-signature. Pour cela, il prend en entrée la paire concernée (m, Σ) ainsi que les clés publiques du groupe gpk et du délégué dpk. L'algorithme retourne l'origine du couple (m, Σ), c'est à dire Groupe si un membre du groupe en est à l'origine et $\mathsf{Délégué}$ s'il s'agit d'un message modifié (ou $\perp$ en cas d'erreur).

$$\{\mathsf{Groupe}, \mathsf{Délégué}\} \leftarrow \mathcal{SGC}.\text{Groupe-Juge}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk})$$

*

Définition 60 (Signature de Groupe Caméléon à Transparence de Groupe).
Un schéma de signature de groupe caméléon à transparence de groupe, notée $\mathcal{FT}$-$\mathcal{SGC}$, est composé d'une base comme décrite en Définition 58 et des deux algorithmes suivants.

– $\mathcal{SGC}$.Groupe-Prouve prend en entrées une paire signature-message $(\Sigma, \mathsf{m}$, la clé secrète d'un membre du groupe usk_i, la clé publique du groupe gpk et celle du délégué dpk. Il retourne une preuve $\pi_{\mathsf{or/mod}}$ démontrant s'il s'agit d'une paire originale ou d'une paire modifiée (ou $\perp$ en cas d'erreur).

$$\pi_{\mathsf{or/mod}} \leftarrow \mathsf{u}\text{-}\mathcal{SGC}.\text{Groupe-Prouve}(\mathsf{m}, \Sigma, \mathsf{usk}_i, \mathsf{gpk}, \mathsf{dpk})$$

– $\mathcal{SGC}$.Groupe-Juge est un algorithme public permettant de vérifier, à l'aide d'une preuve $\pi_{\mathsf{or/mod}}$, si une paire message-signature est originale ou non. Pour cela, il prend en entrée la paire concernée (m, Σ), la preuve $\pi_{\mathsf{or/mod}}$ provenant de l'algorithme $\mathcal{SGC}$.Prouve ainsi que les clés publiques du groupe gpk et du délégué dpk. L'algorithme retourne l'origine du couple (m, Σ), c'est à dire Groupe si le couple provient directement d'un membre du groupe ou $\mathsf{Délégué}$ s'il s'agit d'une paire modifiée par le délégué.

$$\{\mathsf{Groupe}, \mathsf{Délégué}\} \leftarrow \mathcal{SGC}.\text{Groupe-Juge}(\mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}}, \mathsf{gpk}, \mathsf{dpk})$$

*

Définition 61 (Signature de Groupe Caméléon à Transparence Forte).
Un schéma de signature de groupe caméléon à transparence forte, noté $\mathcal{FT}$-$\mathcal{SGC}$, est formé à partir d'une base, comme décrite en définition 58 et des deux algorithmes suivants.

– $\mathcal{SGC}$.u-Prouve prend en entrées une paire message-signature (Σ, m) ainsi qu'une preuve d'ouverture $(\mathsf{u}_i, \pi_{\mathsf{u}})$, la clé secrète de l'utilisateur usk_i, la clé publique du groupe gpk et celle du délégué dpk ainsi qu'un ensemble de couples message-signature ayant été produits avec ces clés $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k=[1,\mathsf{q}]}$. Il retourne une preuve $\pi_{\mathsf{or/mod}}$ indiquant si la paire est originale ou modifiée (ou $\perp$ en cas d'erreur).

$$\pi_{\mathsf{or/mod}} \leftarrow \mathcal{SGC}.\mathsf{u}\text{-Prouve}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_{\mathsf{u}}, \mathsf{usk}_i, \mathsf{gpk}, \mathsf{dpk}, \mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$$

– $\mathcal{SGC}$.u-JUGE est un algorithme public permettant de vérifier, à l'aide d'une preuve d'ouverture $(\mathsf{u}_i, \pi_\mathsf{u})$ et d'une preuve $\pi_{\mathsf{or/mod}}$ générée par l'utilisateur u_i, l'origine d'une paire message-signature. Pour cela, il prend en entrées la paire concernée (m, Σ), les preuves $\pi_{\mathsf{or/mod}}$ et $(\mathsf{u}_i, \pi_\mathsf{u})$ ainsi que les clés publiques du groupe gpk et du délégué dpk. L'algorithme retourne l'origine du couple (m, Σ), c'est-à-dire u_i s'il en est à l'origine, $\mathsf{Délégué}$ s'il s'agit d'un message modifié d'une signature de u_i ou $\perp$ s'il y a une erreur (erreur sur l'utilisateur ou dans les preuves).

$$\{\mathsf{u}_i, \mathsf{Délégué}, \perp\} \leftarrow \mathcal{SGC}.\text{u-JUGE}(\mathsf{m}, \Sigma, \pi_{\mathsf{or/mod}}, \mathsf{u}_i, \pi_\mathsf{u}, \mathsf{gpk}, \mathsf{dpk})$$

Remarque. Nous posons, qu'en cas de problème, l'anonymat de l'utilisateur est levé avant de lui demander de prouver s'il est ou non à l'origine d'une paire. Ceci permet de s'assurer que l'algorithme $\mathcal{SGC}$.u-PROUVE ne puisse être indirectement utilisé par un délégué pour lever l'anonymat sur une signature provenant d'un signataire honnête. En effet, il lui suffirait de générer une signature modifiée sur la signature cible puis d'accuser chaque signataire un par un, le seul capable de se défendre étant le signataire original.

*

8.1.2 Propriétés de sécurité

Nous allons dans un premier temps décrire brièvement l'ensemble des propriétés nécessaires à un schéma de signature de groupe caméléon avant de décrire chacune d'entre elles dans le détail. Celles-ci sont basées d'une part sur les propriétés de sécurité des signatures de groupe (voir Chapitre 2 Section 2.3.2) et d'autre part sur les propriétés nécessaires aux signatures caméléons (voir Chapitre 6 Section 6.2).

Nous avons défini, avec S.Canard, cinq propriétés de sécurité pour un schéma de signature de groupe caméléon. Les quatre premières propriétés sont générales et donc à appliquer aux trois types de schémas tandis que la dernière se décline en fonction de l'hypothèse de transparence considérée. Informellement, ces propriétés sont les suivantes.

– AO**Protection**. Il doit être impossible de produire une signature valide pour laquelle l'autorité d'ouverture ne retrouve pas l'auteur ou retrouve l'auteur mais est incapable de faire la preuve correspondante.
  Cette propriété est reliée à la notion de traçabilité des signatures de groupe.
– u**Protection**. Il doit être impossible, même pour une coalition d'utilisateurs et du délégué, de produire une paire message-signature valide qui ait au moins une des conditions suivantes.
  1. La paire est une version modifiée et au moins une partie a été modifiée alors qu'elle est indiquée comme non modifiable.
  2. L'ouverture de la paire indique un utilisateur honnête u_i comme auteur original et
     – le délégué indiqué dans la paire n'est pas le délégué désigné par l'utilisateur u_i ;
     – ou une preuve d'origine valide sur la paire retourne que l'utilisateur u_i en est l'auteur alors qu'il n'a jamais généré cette signature.

  Cette propriété s'intéresse à la protection des utilisateurs, elle est reliée aux notions d'immuabilité et de responsabilité du délégué introduites pour les signatures caméléons.
– del**Protection**. Si un message et sa signature n'ont pas été modifiés par le délégué, alors même une coalition comprenant tous les utilisateurs, le gestionnaire de groupe et l'autorité d'ouverture ne doit pas pouvoir faire accuser le délégué par le Juge.

Cette propriété s'assure de la protection du délégué. Elle recouvre en particulier la notion de responsabilité des signataires que l'on trouve dans les signatures caméléon.

– **Anonymat**. Étant données des signatures produites par un utilisateur parmi deux utilisateurs honnêtes choisis par un adversaire, celui-ci doit être incapable de retrouver, avec un avantage significatif, lequel est à l'origine de ces signatures. Cette propriété considère que l'adversaire peut monter une coalition comprenant le gestionnaire de groupe, des utilisateurs et le délégué.
  Cette propriété s'assure que la propriété d'anonymat des utilisateurs est respectée même vis-à-vis du gestionnaire de groupe et du délégué. Cette notion correspond à l'anonymat des signatures de groupe.

À ces quatre propriétés de sécurité nous devons ajouter une des trois déclinaisons suivantes de la propriété de **Transparence**.

Non-**Transparence**. Une paire message-signature originale est publiquement distinguable d'une paire modifiée.
Un schéma respectant les quatre propriétés principales mais aucune notion de transparence est un schéma de signature caméléon non-**Transparent**, noté $\mathcal{NT}$-$\mathcal{SGC}$.
L'absence de transparence implique d'effectuer un choix pour l'**Intimité** du schéma. Nous intéressons dans ce chapitre à deux sous-hypothèses : l'absence d'**Intimité** et l'**Intimité** définie comme l'impossibilité pour tout adversaire de retrouver le message original à partir d'une paire message-signature modifiée.

Transparence de Groupe. Pour toute paire message-signature provenant d'une instance initiée par un membre d'un groupe, seuls le délégué et les membres de ce groupe sont capables de distinguer une paire message - signature originale d'une paire modifiée. L'**Intimité** impliquée par cette notion de **Transparence** assure que toute entité n'appartenant pas au groupe est incapable de retrouver le message original à partir d'un message modifié.
Un tel schéma est appelé un schéma de signature caméléon à **Transparence** de Groupe, noté $\mathcal{GT}$-$\mathcal{SGC}$.

Transparence forte. Pour toute paire message-signature provenant d'une instance initiée par un membre u_i d'un groupe donné, seuls le délégué et l'utilisateur u_i sont capables de distinguer une paire message - signature originale d'une paire modifiée. Dans ce cas, l'**Intimité** contenue dans cette notion de **Transparence** est définie comme l'impossibilité pour un adversaire, autre que le signataire original, de retrouver le message original à partir d'une paire modifiée.
Un tel schéma est un schéma de signature caméléon à **Transparence** forte, il est noté $\mathcal{FT}$-$\mathcal{SGC}$.

Afin de donner une définition plus formelle de ces propriétés, nous allons à présent définir l'ensemble des oracles qui seront utilisés par l'adversaire dans les expériences correspondantes.

Les Oracles

Pour rappel, les définitions de propriétés de sécurité définissent un adversaire $\mathcal{A}$ qui cherche à gagner une expérience contre un challengeur $\mathcal{C}$. Les expériences que nous allons définir utilisent chacune un sous-ensemble des oracles et algorithmes suivants. Dans nos diverses expériences, l'initialisation correspond à une unique procédure interactive entre le challengeur et l'adversaire qui correspond à l'exécution des algorithmes $\mathcal{SGC}$.Init et $\mathcal{SGC}$.SanGénClé. En fonction des rôles joués par l'adversaire, celui-ci peut obtenir auprès du challengeur la clé secrète du délégué dsk, les clés du gestionnaire de groupe ggsk et/ou celle de l'autorité d'ouverture osk. D'autre part, les valeurs publiques gpk, dpk, $\mathcal{SGC}$.param sont générées à cette étape.

– ParamGénClé(1^λ) retourne les paramètres $\mathcal{SGC}.\mathsf{param}$ ainsi que les valeurs publique pk et privée sk, générées par le challengeur, avec $\mathsf{pk} \subset \{\mathsf{gpk}, \mathsf{dpk}\}$ et $\mathsf{sk} \subset \{\mathsf{ggsk}, \mathsf{osk}, \mathsf{dsk}\}$

$$(\mathcal{SGC}.\mathsf{param}, \mathsf{pk}, \mathsf{sk}) \leftarrow \text{ParamGénClé}(1^\lambda)$$

Nous utilisons, d'autre part, les oracles suivants.

$\mathcal{O}$.Corromps permet à l'adversaire de corrompre des utilisateurs. Cet oracle prend en entrée l'utilisateur u_i que l'adversaire souhaite corrompre et retourne $(\mathsf{usk}_i, \mathsf{cert}_i)$.

$$(\mathsf{usk}_i, \mathsf{cert}_i) \leftarrow \mathcal{O}.\text{Corromps}(\mathsf{u}_i)$$

$\mathcal{O}$.Signe correspond à l'algorithme $\mathcal{SGC}$.Signe. Il prend en entrées un message m, la clé publique d'un délégué dpk, un utilisateur u_i et une variable ADM définissant les modifications qui seront autorisées au délégué, notons que l'adversaire peut laisser certains champs vides. L'oracle retourne la signature Σ correspondante.

$$\Sigma \leftarrow \mathcal{O}.\text{Signe}(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$$

$\mathcal{O}$.Modifie agit de manière analogue à l'algorithme de modification $\mathcal{SGC}$.Modifie. Il prend en entrées un couple message - signature (m, Σ) et une variable MOD indiquant les modifications à effectuer. L'oracle retourne la signature modifiée correspondante si les modifications sont admissibles et $\perp$ sinon.

$$\Sigma' \leftarrow \mathcal{O}.\text{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD})$$

$\mathcal{O}$.Ouvre correspond à l'algorithme d'ouverture de signature $\mathcal{SGC}$.Ouvre et permet ainsi à l'adversaire d'obtenir l'identité de l'utilisateur qui est à l'origine d'une paire message-signature (m, Σ).

$$(\mathsf{u}_i, \pi_\mathsf{u}) \leftarrow \mathcal{O}.\text{Ouvre}(\mathsf{m}, \Sigma)$$

$\mathcal{O}$.Prouve est l'oracle correspondant à l'algorithme de preuve d'origine. Cet oracle n'est donc utile que pour les constructions de type $\mathcal{FT}$-$\mathcal{SGC}$ ou $\mathcal{GT}$-$\mathcal{SGC}$. Dans le premier cas, il correspond à l'algorithme $\mathcal{SGC}.\mathsf{u}$-Prouve et dans le second à $\mathcal{SGC}.\mathsf{Groupe}$-Prouve. L'oracle prend en entrées le couple message-signature cible (m, Σ) et l'utilisateur u_i interrogé. Dans le cas des signatures $\mathcal{FT}$-$\mathcal{SGC}$, une preuve d'ouverture π_u ainsi que la base de données des signatures caméléons de groupe déjà publiées $\mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]}$ sont ajoutés aux entrées. L'oracle retourne la preuve $(\pi_{\mathsf{or/mod}}, \mathsf{u})$ que l'algorithme de preuve correspondant aurait retourné.

$$\pi_{\mathsf{or/mod}} \leftarrow \mathcal{O}.\text{Prouve}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$$

$\mathcal{O}$.FaisRejoindre est un oracle permettant à un adversaire, jouant le rôle du gestionnaire de groupe, de simuler le protocole interactif $\mathcal{SGC}$.Rejoins avec un utilisateur honnête souhaitant rejoindre le groupe des signataires.

$$\mathcal{O}.\text{FaisRejoindre}(\mathsf{ggsk})$$

$\mathcal{O}$.Rejoins permet à un adversaire, jouant le rôle d'un utilisateur u_j, de simuler le protocole interactif $\mathcal{SGC}$.Rejoins avec un gestionnaire de groupe honnête.

$$\mathcal{O}.\text{Rejoins}(\mathsf{u}_j, (\mathsf{psk}_j, \mathsf{ppk}_j)$$

$\mathcal{O}$.Challenge$_b$ prend en entrées deux utilisateurs honnêtes u_0 et u_1, un message m, une clé publique de délégué dpk et une description ADM. Il renvoie la signature de groupe caméléon qu'aurait produit l'utilisateur u_b sur le message m pour permettre au délégué correspondant à la clé publique dpk de la modifier selon la variable ADM.

$$\Sigma \leftarrow \mathcal{O}.\text{Challenge}_b(\mathsf{u}_0, \mathsf{u}_1, \mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$$

$\mathcal{O}$.Modif/Signe$_b$ prend en entrées un utilisateur honnête u_i, un message m, deux descriptions ADM et MOD. Si $b = 0$, l'oracle renvoie une signature originale de l'utilisateur u_i sur le message correspondant au message modifié selon MOD. Et si $b = 1$, l'oracle signe le message m selon ADM comme le ferait l'utilisateur u_i puis génère une signature modifiée selon MOD, enfin il retourne la signature modifiée.

$$\Sigma \leftarrow \mathcal{O}.\text{Modif/Signe}_b(\mathsf{u}_i, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$$

$\mathcal{O}$.GDModifie$_b(\mathsf{u}_i, \mathsf{m}_0, \mathsf{m}_1, \mathsf{m}', \mathsf{ADM})$ prend en entrées un utilisateur honnête u_i, deux messages une description ADM, un message modifié cible m' ainsi que deux messages m_0 et m_1 égaux sur les parties définies comme non modifiables par ADM et qui peuvent être transformés en le message modifié m'. L'oracle signe le message m_b selon ADM comme le ferait l'utilisateur u_i puis génère une signature modifiée pour obtenir m', enfin il retourne la signature modifiée.

$$\Sigma \leftarrow \mathcal{O}.\text{GDModifie}_b(\mathsf{u}_i, \mathsf{m}_0, \mathsf{m}_1, \mathsf{m}', \mathsf{ADM})$$

Les quatre propriétés de base

Pour des raisons de lisibilité des expériences, les entrées choisies par l'adversaire pour les oracles ainsi que le $\mathcal{O}$. seront omis lorsque cela sera évident. Par exemple, l'oracle de signature $\mathcal{O}$.Signe$(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$ sera abusivement noté Signe dans les expériences. D'autre part, on note $\mathcal{HU}$ l'ensemble des utilisateurs honnêtes et $\mathcal{CU}$ l'ensemble des utilisateurs corrompus. On a $\mathsf{U} = \mathcal{HU} \cup \mathcal{CU}$ et $\mathcal{HU} \cap \mathcal{CU} = \emptyset$.

AOProtection Cette propriété a pour objectif de s'assurer que l'autorité d'ouverture peut effectuer correctement son travail. Dans l'expérience correspondante, nous considérons que l'adversaire ne peut pas jouer le rôle du gestionnaire de groupe afin de nous assurer que celui-ci effectue correctement son travail et, en particulier, met à jour la table des registres. Cependant l'adversaire peut jouer le rôle du délégué et ajouter au groupe les utilisateurs honnêtes de son choix. De manière plus formelle, nous avons la définition suivante.

Définition 62 (AOProtection).
Soient λ un paramètre de sécurité et ParamGénClé(1^λ) l'algorithme défini précédemment qui permet au challengeur d'initialiser les paramètres du schéma $\mathcal{SGC}.\mathsf{param}$, la clé secrète du délégué dsk, celle de l'autorité d'ouverture osk et celle du gestionnaire de groupe ggsk. Il publie ensuite la clé publique du groupe gpk et celle du délégué dpk. Soit $\mathcal{A}$ un attaquant qui a accès aux clés secrètes du délégué dsk et de l'autorité d'ouverture osk ainsi qu'aux oracles suivants : $\mathcal{O}$.Rejoins$(\mathsf{u}_j, (\mathsf{psk}_j, \mathsf{ppk}_j))$, $\mathcal{O}$.Ajoute(u_j)) et $\mathcal{O}$.Corromps(u_i). À la fin de l'expérience, il retourne un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$ composé d'une signature Σ^* modifiable par le délégué dpk^* sur le message m^*. Nous définissons l'expérience contre l'AO**Protection** comme suit.

$\text{Exp}^{\mathcal{SGC}}_{\mathsf{AO}\mathcal{P},\mathcal{A}}(\lambda)$:
$(\mathsf{sk} = \{\mathsf{osk}, \mathsf{ggsk}, \mathsf{dsk}\}, \mathsf{pk} = \{\mathsf{gpk}, \mathsf{dpk}\}) \leftarrow \text{ParamGénClé}(1^\lambda)$,
$(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*) \longleftarrow \mathcal{A}^{\text{Rejoins,Ajoute,Corromps}}(\mathsf{dsk}, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$,
Retourne 1 si $\mathcal{SGC}.\text{Vérifie}(\mathsf{m}^*, \Sigma^*, \mathsf{gpk}, \mathsf{dpk}^*) = \mathsf{vrai}$,
$\mathcal{SGC}.\text{Ouvre}(\mathsf{m}^*, \Sigma^*, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk}^*) = (\mathsf{u}, \pi_\mathsf{u})$ et :
soit $\mathcal{SGC}.\text{OuvreJuge}(\mathsf{m}^*, \Sigma^*, \mathsf{u}, \pi_\mathsf{u}, \mathsf{gpk}, \mathsf{dpk}^*) = \mathsf{faux}$,
soit $(\mathsf{u}, \pi_\mathsf{u}) = \perp$.
Sinon il retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SGC}}_{\mathsf{AO}\mathcal{P},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{AO}\mathcal{P},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{SGC}}_{\mathsf{AO}\mathcal{P},\mathcal{A}}(\lambda)].$$

Un schéma de signature de groupe caméléon $\mathcal{SGC}$ est dit **AOprotégé** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{AO}\mathcal{P},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{AO}\mathcal{P},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \quad *$$

uProtection Cette propriété permet de garantir que les utilisateurs membres du groupe ne peuvent pas être faussement accusés et que les signatures qu'ils produisent ne peuvent pas être modifiées sans respecter la variable ADM. Dans l'expérience correspondante, l'adversaire peut jouer le rôle du gestionnaire de groupe, de l'autorité d'ouverture et du délégué, et peut aussi corrompre des utilisateurs.

Définition 63 (uProtection).
Soient λ un paramètre de sécurité et $\text{ParamGénClé}(1^\lambda)$ l'algorithme, défini précédemment. Cet algorithme permet au challengeur d'initialiser les paramètres du schéma $\mathcal{SGC}.\mathsf{param}$ ainsi que la clé secrète du délégué dsk, celle de l'autorité d'ouverture osk et celle du gestionnaire de groupe ggsk ainsi que la clé publique du délégué dpk et du groupe gpk. Soit $\mathcal{A}$ un adversaire a accès d'une part aux clés secrètes du délégué dsk, de l'autorité d'ouverture osk et du gestionnaire de groupe ggsk et d'autre part aux oracles $\mathcal{O}.\text{FaisRejoindre}(\mathsf{ggsk})$, $\mathcal{O}.\text{Signe}(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$, $\mathcal{O}.\text{Prouve}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$ et enfin $\mathcal{O}.\text{Corromps}(\mathsf{u}_i)$. À la fin de l'expérience, $\mathcal{A}$ retourne un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$ composé d'une signature Σ^* modifiable par le délégué dpk^* sur le message m^*. Nous définissons l'expérience contre l'u**Protection** comme suit.

$\text{Exp}^{\mathcal{SGC}}_{\mathsf{u}\mathcal{P},\mathcal{A}}(\lambda)$:

$(\mathsf{sk} = \{\mathsf{osk}, \mathsf{ggsk}, \mathsf{dsk}\}, \mathsf{pk} = \{\mathsf{gpk}, \mathsf{dpk}\}) \leftarrow \text{ParamGénClé}(1^\lambda)$,

$(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*) \longleftarrow \mathcal{A}^{\text{FaisRejoindre,Signe,Prouve,Corromps}}(\mathsf{dsk}, \mathsf{osk}, \mathsf{ggsk}, \mathsf{gpk}, \mathsf{dpk})$,

Soit $\{(\mathsf{m}_k, \mathsf{u}_k, \mathsf{dpk}_k, \mathsf{ADM}_k)\}_{k\in[1,\mathsf{n}]}$ les requêtes à l'oracle $\mathcal{O}$.Signe et $\{\Sigma_k\}_{k\in[1,\mathsf{n}]}$ les réponses correspondantes.

Retourne 1 si :
- $\mathcal{SGC}$.Vérifie$(\mathsf{m}^*, \Sigma^*, \mathsf{gpk}, \mathsf{dpk}^*) = \mathsf{vrai}$,
- $\mathcal{SGC}$.Ouvre$(\mathsf{m}^*, \Sigma^*, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk}^*) = (\mathsf{u}, \pi_\mathsf{u})$ avec $\mathcal{SGC}$.OuvreJuge$(\mathsf{m}^*, \Sigma^*, \mathsf{u}, \pi_\mathsf{u}, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{vrai}$ et $\mathsf{u} \in \mathcal{HU}$,
- $\forall j \in [1, \mathsf{n}], (\mathsf{m}^*, \mathsf{u}, \mathsf{dpk}^*) \neq (\mathsf{m}_j, \mathsf{u}_j, \mathsf{dpk}_j)$ et

  soit $\forall j \in [1, \mathsf{n}]$, si $\mathcal{SGC}$.Ouvre$(\mathsf{m}_j, \Sigma_j, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk}_j) = (\mathsf{u}, \pi_{\mathsf{u}j})$ alors $\mathsf{dpk}_j \neq \mathsf{dpk}^*$ ou $\exists i \in [1, t] : \mathsf{m}_{j_i} \neq \mathsf{m}^*_i$ et $i \notin \mathsf{ADM}_j$;

  soit $\mathcal{SGC}$.u-Juge$(\mathsf{m}^*, \Sigma^*, \pi_{\mathsf{or/mod}}, \mathsf{u}, \pi_\mathsf{u}, \mathsf{gpk}, \mathsf{dpk}^*) = \mathsf{u}$ avec $\pi_{\mathsf{or/mod}} \leftarrow \mathcal{SGC}$.u-Prouve$(\mathsf{m}^*, \Sigma^*, \mathsf{u}, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$ (dans le cas d'un schéma $\mathcal{FT}$-$\mathcal{SGC}$) ;

  soit $\mathcal{SGC}$.Groupe-Juge$(\mathsf{m}^*, \Sigma^*, \pi_{\mathsf{or/mod}}, \mathsf{gpk}, \mathsf{dpk}^*) = \mathsf{Groupe}$ avec $\pi_{\mathsf{or/mod}} \leftarrow \mathcal{SGC}$.Groupe-Prouve$(\mathsf{m}^*, \Sigma^*)$ ou $\pi_{\mathsf{or/mod}} = \emptyset$ (dans le cas d'un schéma $\mathcal{NT}$-$\mathcal{SGC}$ ou $\mathcal{GT}$-$\mathcal{SGC}$).

Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SGC}}_{\mathsf{u}\mathcal{P},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{u}\mathcal{P},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{SGC}}_{\mathsf{u}\mathcal{P},\mathcal{A}}(\lambda)].$$

Un schéma de signature de groupe caméléon $\mathcal{SGC}$ est dit **uprotégé** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{u}\mathcal{P},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{u}\mathcal{P},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \quad *$$

delProtection Cette propriété permet d'assurer que le délégué ne puisse pas être faussement accusé. Dans l'expérience correspondante, l'adversaire peut jouer le rôle du gestionnaire de groupe, de l'autorité d'ouverture et de tous les utilisateurs du groupe, mais pas celui du délégué.

Définition 64 (delProtection).
Soient λ un paramètre de sécurité et ParamGénClé(1^λ) l'algorithme définit précédemment. Cet algorithme permet au challengeur d'initialiser les paramètres du schéma $\mathcal{SGC}$.param ainsi que les clés secrètes de l'autorité d'ouverture osk, du gestionnaire de groupe ggsk et de générer la paire de clés du délégué (dsk, dpk). Les paramètres sont intégrés dans la clé publique du groupe gpk. Il publie ensuite les clés publiques du délégué dpk et du groupe gpk. Soit $\mathcal{A}$ un adversaire qui a accès d'une part aux clés secrètes de l'autorité d'ouverture osk et du gestionnaire de groupe ggsk, et d'autre part aux oracles $\mathcal{O}$.FaisRejoindre(ggsk) et $\mathcal{O}$.Modifie$(\mathsf{m}, \Sigma, \mathsf{MOD})$. L'adversaire retourne un triplet $(\pi^*_{\mathsf{or/mod}}, \mathsf{m}^*, \Sigma^*)$ composé d'une signature Σ^* sur le message m^* et d'une preuve d'origine $\pi^*_{\mathsf{or/mod}}$. Nous définissons l'expérience contre la del**Protection** comme suit.

$\text{Exp}^{\mathcal{SGC}}_{\mathsf{del}\mathcal{P},\mathcal{A}}(\lambda)$:

$(\mathsf{sk} = \{\mathsf{osk}, \mathsf{ggsk}, \mathsf{dsk}\}, \mathsf{pk} = \{\mathsf{gpk}, \mathsf{dpk}\}) \leftarrow$ ParamGénClé(1^λ),

$(\pi^*_{\mathsf{or/mod}}, \mathsf{m}^*, \Sigma^*) \longleftarrow \mathcal{A}^{\text{FaisRejoindre,Modifie}}(\mathsf{ggsk}, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$,

Soit $\{(\mathsf{m}'_k, \Sigma'_k)\}_{k\in[1,\mathsf{n}]}$ les réponses aux requêtes $\{(\mathsf{m}_k, \Sigma_k, \mathsf{MOD}_k)\}_{k\in[1,\mathsf{n}]}$ à l'oracle $\mathcal{O}$.Modifie, on note $\{(\mathsf{u}_k, \pi_{\mathsf{u}k})\}_{k\in[1,\mathsf{n}]}$ les sorties de $\mathcal{SGC}$.Ouvre$(\mathsf{m}_k, \Sigma_k, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$.

$(\mathsf{u}, \pi_\mathsf{u}) \leftarrow \mathcal{SGC}$.Ouvre$(\mathsf{m}^*, \Sigma^*, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$.

Retourne 1 si :
- $\mathcal{SGC}$.Vérifie$(\mathsf{m}^*, \Sigma^*, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{vrai}$,
- $\forall j \in [1, \mathsf{n}]$,
  - soit $(\mathsf{m}^*, \mathsf{u}) \neq (\mathsf{m}'_j, \mathsf{u}_j)$,
  - soit $(\mathsf{m}^*, \mathsf{u}) = (\mathsf{m}'_j, \mathsf{u}_j)$ avec $\mathsf{m}^*_0 \neq \mathsf{m}_{j_0}$ dans le cas $\mathcal{NT}$-$\mathcal{SGC}$.
- et enfin :
  - soit $\mathcal{SGC}$.Groupe-Juge$(\mathsf{m}^*, \Sigma^*, \pi^*_{\mathsf{or/mod}}, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{Délégué}$ (dans le cas d'un schéma $\mathcal{NT}$-$\mathcal{SGC}$ ou $\mathcal{GT}$-$\mathcal{SGC}$) ;
  - soit $\mathcal{SGC}$.u-Juge$(\mathsf{m}^*, \Sigma^*, \mathsf{u}, \pi_\mathsf{u}, \pi^*_{\mathsf{or/mod}}, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{Délégué}$ (dans le cas d'un schéma $\mathcal{FT}$-$\mathcal{SGC}$).

Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SGC}}_{\mathsf{del}\mathcal{P},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{del}\mathcal{P},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{SGC}}_{\mathsf{del}\mathcal{P},\mathcal{A}}(\lambda)].$$

Un schéma de signature de groupe caméléon $\mathcal{SGC}$ est dit **delprotégé** si quelque soit l'adversaire polynomial $\mathcal{A}$ son succès $\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{del}\mathcal{P},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Succ}^{\mathcal{SGC}}_{\mathsf{u}\mathcal{P},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

Anonymat Cette propriété permet de garantir l'anonymat des utilisateurs du groupe. Dans l'expérience correspondante, l'adversaire peut jouer le rôle du gestionnaire de groupe et du délégué mais pas de l'autorité d'ouverture. Il peut aussi corrompre des utilisateurs.

Définition 65 (Anonymat).
Soient λ un paramètre de sécurité, ParamGénClé(1^λ) l'algorithme, défini précédemment, qui permet au challengeur d'initialiser les paramètres du schéma $\mathcal{SGC}$.param, la clé secrète du délégué dsk, de l'autorité d'ouverture osk et celle du gestionnaire de groupe ggsk. Il publie ensuite la clé publique du délégué dpk et du groupe gpk. Soit $\mathcal{A}$ un adversaire a accès d'une part à la clé secrète du délégué dsk et à celle du gestionnaire de groupe ggsk et d'autre part aux oracles suivants : $\mathcal{O}$.FaisRejoindre(ggsk), $\mathcal{O}$.Corromps(u_i), $\mathcal{O}$.Signe$(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$, $\mathcal{O}$.Ouvre(m, Σ) ainsi que $\mathcal{O}$.Prouve$(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$ et $\mathcal{O}$.Challenge$_b(\mathsf{u}_0, \mathsf{u}_1, \mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$. À la fin de l'expérience, $\mathcal{A}$ retourne un bit b^*. Nous définissons l'expérience contre l'**Anonymat** comme suit.

$\text{Exp}^{\mathcal{SGC}}_{\mathbf{anon},\mathcal{A}}(\lambda)$:
$(\mathsf{sk} = \{\mathsf{osk}, \mathsf{ggsk}, \mathsf{dsk}\}, \mathsf{pk} = \{\mathsf{gpk}, \mathsf{dpk}\}) \leftarrow \text{ParamGénClé}(1^\lambda)$,
$b \longleftarrow \{0,1\}$,
$b^* \longleftarrow \mathcal{A}^{\text{FaisRejoindre},\text{Corromps},\text{Signe},\text{Ouvre},\text{Prouve},\text{Challenge}_b}(\mathsf{dsk}, \mathsf{ggsk}, \mathsf{gpk}, \mathsf{dpk})$,
Retourne 1 si $b^* = b$ sinon retourne 0.

L'avantage de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SGC}}_{\mathbf{anon},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{anon},\mathcal{A}}(\lambda) = |Pr[1 \longleftarrow \text{Exp}^{\mathcal{SGC}}_{\mathbf{anon},\mathcal{A}}(\lambda)] - 1/2|.$$

Un schéma de signature de groupe caméléon $\mathcal{SGC}$ est dit **anonyme** si quelque soit l'adversaire polynomial $\mathcal{A}$ son avantage $\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{anon},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{anon},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \quad *$$

Différentes notions de transparence

Nous allons maintenant définir plus formellement les différentes notions de transparence.

La première déclinaison correspond à l'absence de **Transparence**. Il y a donc deux possibilités. La première est de conserver la notion d'**Intimité** définie pour les signatures caméléons et nous nous assurons que le message original ne peut pas être retrouvé à partir d'une paire message-signature modifiée. Cette possibilité correspond à l'expérience que nous décrirons ici. La seconde possibilité implique un schéma sans **Intimité**. Dans ce cas, nulle expérience n'est nécessaire.

Intimité Cette propriété permet de s'assurer que l'on ne puisse retrouver le message original à partir d'un message modifié. Dans l'expérience, l'adversaire peut jouer le rôle de l'autorité d'ouverture et celui du gestionnaire de groupe.

Définition 66 (Intimité).
Soient λ un paramètre de sécurité et $\text{ParamGénClé}(1^\lambda)$ l'algorithme définit précédemment qui permet au challengeur d'initialiser les paramètres du schéma $\mathcal{SGC}.\mathsf{param}$ ainsi que les clés secrètes de l'autorité d'ouverture osk, du gestionnaire de groupe ggsk et de générer la paire de clés du délégué $(\mathsf{dsk}, \mathsf{dpk})$. Il publie ensuite les clés publiques du délégué dpk et du groupe gpk. Soit $\mathcal{A}$ un adversaire qui a accès d'une part aux clés secrètes de l'autorité d'ouverture osk et du gestionnaire de groupe ggsk et d'autre part aux oracles $\mathcal{O}.\text{FaisRejoindre}(\mathsf{ggsk})$, $\mathcal{O}.\text{Signe}(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$, $\mathcal{O}.\text{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD})$, $\mathcal{O}.\text{Prouve}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$ et $\mathcal{O}.\text{GDModifie}_b(\mathsf{u}_i, \mathsf{m}_0, \mathsf{m}_1, \mathsf{m}', \mathsf{ADM})$. En fin d'expérience, l'adversaire $\mathcal{A}$ retourne un bit b^*. Nous définissons l'expérience contre l'**Intimité** comme suit.

$\text{Exp}^{\mathcal{SGC}}_{\mathbf{Int},\mathcal{A}}(\lambda)$:
$(\mathsf{sk} = \{\mathsf{osk}, \mathsf{ggsk}, \mathsf{dsk}\}, \mathsf{pk} = \{\mathsf{gpk}, \mathsf{dpk}\}) \leftarrow \text{ParamGénClé}(1^\lambda)$,
$b \longleftarrow \{0,1\}$,
$b^* \longleftarrow \mathcal{A}^{\text{FaisRejoindre},\text{Signe},\text{Modifie},\text{Prouve},\text{GDModifie}_b}(\mathsf{osk}, \mathsf{ggsk}, \mathsf{gpk}, \mathsf{dpk})$,
Retourne 1 si $b^* = b$ sinon retourne 0.

L'avantage de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SGC}}_{\mathbf{Int},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{Int},\mathcal{A}}(\lambda) = |Pr[1 \longleftarrow \text{Exp}^{\mathcal{SGC}}_{\mathbf{Int},\mathcal{A}}(\lambda)] - 1/2|.$$

Un schéma de signature de groupe caméléon $\mathcal{SGC}$ est dit **Intime** si quelque soit l'adversaire polynomial $\mathcal{A}$ son avantage $\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{Int},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{Int},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

Transparence de Groupe Cette propriété permet de garantir que seul un membre du groupe ou le délégué sont capables de distinguer une paire originale d'une paire modifiée. En conséquence l'adversaire n'est pas autorisé à corrompre d'utilisateur, car tous les utilisateurs du groupe sont potentiellement capables de distinguer une signature originale d'une signature modifiée. Un tel adversaire gagnerait donc toujours.

Pour cette expérience, l'adversaire peut jouer le rôle de l'autorité d'ouverture et celui du gestionnaire de groupe.

Définition 67 (Transparence de Groupe).
Soient λ un paramètre de sécurité et $\text{ParamGénClé}(1^\lambda)$ l'algorithme définit précédemment qui permet au challengeur d'initialiser les paramètres du schéma $\mathcal{SGC}.\mathsf{param}$, les clés secrètes de l'autorité d'ouverture osk et du gestionnaire de groupe ggsk et de générer la paire de clés du délégué $(\mathsf{dsk}, \mathsf{dpk})$. Il publie ensuite les clés publiques du délégué dpk et du groupe gpk. Soit $\mathcal{A}$ un adversaire qui a accès aux clés secrètes de l'autorité d'ouverture osk et du gestionnaire de groupe ggsk ainsi qu'aux oracles $\mathcal{O}.\text{FaisRejoindre}(\mathsf{ggsk})$, $\mathcal{O}.\text{Signe}(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$, $\mathcal{O}.\text{Modifie}(\mathsf{m}, \Sigma, \mathsf{MOD})$, $\mathcal{O}.\text{Prouve}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$ et enfin à l'oracle de challenge $\mathcal{O}.\text{Modif/Signe}_b(\mathsf{u}_i, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$. Notons que l'oracle $\mathcal{O}.\text{Prouve}$ ne peut pas prendre en entrée une signature engendrée par $\mathcal{O}.\text{Modif/Signe}_b$ et que ce dernier prend un utilisateur honnête en entrée. En fin d'expérience, l'adversaire $\mathcal{A}$ retourne un bit b^* indiquant que la signature challenge est un original si $b^* = 0$ ou indiquant qu'il s'agit d'une signature modifiée si $b^* = 1$. Nous définissons l'expérience contre la **Transparence** de groupe comme suit.

$\text{Exp}^{\mathcal{SGC}}_{\mathbf{gTransp},\mathcal{A}}(\lambda)$:
$(\mathsf{sk} = \{\mathsf{osk}, \mathsf{ggsk}, \mathsf{dsk}\}, \mathsf{pk} = \{\mathsf{gpk}, \mathsf{dpk}\}) \leftarrow \text{ParamGénClé}(1^\lambda)$,
$b \longleftarrow \{0, 1\}$,
$b^* \longleftarrow \mathcal{A}^{\text{FaisRejoindre,Signe,Modifie,Prouve,Modif/Signe}_b}(\mathsf{osk}, \mathsf{ggsk}, \mathsf{gpk}, \mathsf{dpk})$,
Retourne 1 si $b^* = b$ sinon retourne 0.

L'avantage de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{SGC}}_{\mathbf{gTransp},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{gTransp},\mathcal{A}}(\lambda) = |Pr[1 \longleftarrow \text{Exp}^{\mathcal{SGC}}_{\mathbf{gTransp},\mathcal{A}}(\lambda)] - 1/2|.$$

Un schéma de signature de groupe caméléon $\mathcal{SGC}$ est dit à **Transparence** de Groupe si quelque soit l'adversaire polynomial $\mathcal{A}$ son avantage $\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{gTransp},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{gTransp},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

Transparence forte. Cette propriété a pour objectif d'assurer que seuls le signataire et le délégué sont capables de distinguer une paire originale d'une paire modifiée. Dans l'expérience correspondante, l'adversaire peut jouer le rôle de l'autorité d'ouverture et du gestionnaire de groupe. Il peut aussi corrompre des utilisateurs. Cependant il doit toujours rester au moins un utilisateur honnête dans le système.

Définition 68 (Transparence forte).
Soient λ un paramètre de sécurité et ParamGénClé(1^λ) l'algorithme définit précédemment. Celui-ci permet au challengeur d'initialiser les paramètres du schéma $\mathcal{SGC}.\mathsf{param}$, les clés secrètes de l'autorité d'ouverture osk et du gestionnaire de groupe ggsk ainsi que la paire de clés du délégué $(\mathsf{dsk}, \mathsf{dpk})$. Le challengeur publie ensuite les clés publiques du délégué dpk et du groupe gpk. Soit $\mathcal{A}$ un adversaire qui a accès à la clé secrète de l'autorité d'ouverture osk et à celle du gestionnaire de groupe ggsk ainsi qu'aux oracles $\mathcal{O}$.FaisRejoindre(ggsk), $\mathcal{O}$.Corromps(u_i), $\mathcal{O}$.Signe($\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM}$), $\mathcal{O}$.Modifie($\mathsf{m}, \Sigma, \mathsf{MOD}$), $\mathcal{O}$.Prouve($\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]}$), et enfin, $\mathcal{O}$.Modif/Signe$_b$($\mathsf{u}_i, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD}$). Notons que l'oracle $\mathcal{O}$.Prouve ne peut prendre en entrée une signature engendrée par $\mathcal{O}$.Modif/Signe$_b$ et que ce dernier prend un utilisateur honnête en entrée. En fin d'expérience, l'adversaire $\mathcal{A}$ retourne un bit b^* indiquant une paire originale si $b^* = 0$ ou une paire modifiée si $b^* = 1$. Nous définissons l'expérience contre la **Transparence** forte comme suit.

$\mathrm{Exp}^{\mathcal{SGC}}_{\mathbf{fTransp},\mathcal{A}}(\lambda)$:
$(\mathsf{sk} = \{\mathsf{osk}, \mathsf{ggsk}, \mathsf{dsk}\}, \mathsf{pk} = \{\mathsf{gpk}, \mathsf{dpk}\}) \leftarrow$ ParamGénClé(1^λ),
$b \longleftarrow \{0, 1\}$,
$b^* \longleftarrow \mathcal{A}^{\text{FaisRejoindre,Corromps,Signe,Modifie,Prouve,Modif/Signe}_b}(\mathsf{osk}, \mathsf{ggsk}, \mathsf{gpk}, \mathsf{dpk})$,
Retourne 1 si $b^* = b$ sinon retourne 0.

L'avantage de l'adversaire $\mathcal{A}$ dans l'expérience $\mathrm{Exp}^{\mathcal{SGC}}_{\mathbf{fTransp},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{fTransp},\mathcal{A}}(\lambda) = |Pr[1 \longleftarrow \mathrm{Exp}^{\mathcal{SGC}}_{\mathbf{fTransp},\mathcal{A}}(\lambda)] - 1/2|.$$

Un schéma de signature de groupe caméléon $\mathcal{SGC}$ est dit fortement **Transparent** si quelque soit l'adversaire polynomial $\mathcal{A}$ son avantage $\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{fTransp},\mathcal{A}}(\lambda)$ est négligeable.

$$\mathsf{Adv}^{\mathcal{SGC}}_{\mathbf{fTransp},\mathcal{A}}(\lambda) < \epsilon \text{ avec } \epsilon \text{ négligeable .} \qquad *$$

8.1.3 Introduction à nos constructions

Dans les différentes constructions que je présente ici, nous utilisons le fait que les schémas de signatures de groupe sûrs dans le modèle BSZ (voir Chapitre 2) les plus utilisés (notamment [NSN04, FI05, DP06] qui sont prouvés sûrs dans le modèle de l'oracle aléatoire et [Gro07] qui est prouvé dans le modèle standard) suivent une même architecture pour la signature.

1. Le certificat d'appartenance de l'utilisateur est chiffré pour l'autorité d'ouverture.
2. Le message est signé, en général avec d'autres éléments.
3. Une preuve de connaissance est construite afin de prouver que l'on connait un certificat d'appartenance et que le chiffré de l'étape 1 est bien un chiffré correctement formé de celui-ci.

Dans le cas de [Gro07], les valeurs du certificat sont mises en gage pour une preuve de connaissance à témoin indistinguable "Groth-Sahaï" sous l'hypothèse DLIN. Ces engagements correspondent alors aux chiffrés des valeurs du certificat par un chiffrement linéaire. L'étape 1 correspond alors à la génération de ces engagements.

Dans le cas de signatures de groupes prouvées sûre dans le modèle aléatoire, la signature utilisée est une signature de connaissance "Fiat-Shamir" qui permet ainsi de signer le message en tant que personne connaissant le certificat d'appartenance correspondant au chiffré donné. Les étapes 2 et 3 sont ainsi effectuées en même temps.

Dans tous les cas, l'étape 1 peut être effectué avant de connaître le message a signé et la connaissance de (ou des) chiffré(s) de cet étape avant de choisir le message à signer ne donne pas d'avantage significatif à un adversaire contre les propriétés d'**Anonymat**, de **Traçabilité** ou de **Non-diffamation** comme définies dans le modèle BSZ.

Dans ce chapitre, nous noterons $\{T_i\}_{i\in[1,\tau]}$ le(s) chiffré, pour l'autorité d'ouverture, du certificat d'appartenance obtenu à la première étape. Nous considérons dans les preuves que l'oracle $\mathcal{O}.\mathcal{SG}.\textsc{Signe}$ peut être décomposé en deux sous-oracles reliés entre eux $\mathcal{O}.\mathcal{SG}.\textsc{Signe}_1(\mathsf{u})$ et $\mathcal{O}.\mathcal{SG}.\textsc{Signe}_2(\mathsf{m})$ tels que le premier nous donne les valeurs $\{T_i\}_{i\in[1,\tau]}$ tandis que le second permet d'obtenir la signature complète du message m comprenant les valeurs $\{T_i\}_{i\in[1,\tau]}$.

De même l'oracle de challenge $\mathcal{O}.\mathcal{SG}.\textsc{Challenge}_b$ est décomposé de manière similaire.

8.2 Notre solution non-transparente

Nous nous intéressons à présent à un schéma pour lequel un couple message-signature original est publiquement distinguable d'un couple message-signature modifié. Je décrirai d'abord une solution sans intimité. Je présenterai ensuite les modifications nécessaires à ce premier schéma afin d'atteindre cette propriété.

8.2.1 Le schéma

L'idée de notre construction est de combiner une signature de groupe avec un schéma de signature caméléon simplifié. Une telle construction peut être prouvé sûre avec ou sans oracle aléatoire en fonction de la nécessité d'un tel oracle pour les preuves de sécurité des schémas choisis pour la signature de groupe (voir Chapitre 2) et la fonction de hachage caméléon (voir définition 22).

Le principe de cette solution est basé sur les deux observations suivantes.

1. Dans la version initiale des signatures caméléons, nous nous assurons qu'un autre signataire soit incapable de réutiliser une collision, en ajoutant la clé publique du signataire au message haché (voir Chapitre 6). Dans le contexte des signatures de groupe nous devons considérer l'anonymat du signataire dans le groupe. Une solution naïve serait de remplacer la clé publique du signataire par la clé publique du groupe. Cependant, cela implique que n'importe quel membre du groupe pourrait réutiliser ces collisions pour casser la del**Protection**. Nous proposons donc d'utiliser le chiffré du certificat d'appartenance au groupe qui est généré à la première étape de la signature de groupe (voir ci-dessus). Informellement, cette donnée permet de limiter la réutilisation des collisions. En effet, chaque signataire a un certificat d'appartenance différent et lors des phases 2 et 3 de la signature, le signataire doit prouver la connaissance de celui-ci et que le(s) chiffré(s) obtenue(s) en phase 1 sont bien un chiffré correct de celui-ci.

2. Dans une signature non transparente sans intimité, il est possible de conserver le message original dans les signatures modifiées. Ainsi pour vérifier qu'un message modifié est valide il suffit de le comparer au message original et de vérifier que les parties différentes sont compatibles avec la description ADM fixée par le signataire.

Plus formellement, nous obtenons la solution suivante.

Construction 25 (Schéma de signature de groupe caméléon $\mathcal{NT}$-$\mathcal{SGC}$).
Notre schéma utilise un schéma de signature de groupe, noté $\mathcal{SG}$, sûr dans le modèle BSZ (voir définition 32 page 35). Nous utilisons, d'autre part, un schéma de fonction de hachage caméléon $\mathcal{HC}$ **uniforme** et **résistant aux collisions** (cf. Définition 22 page 13). Le schéma fonctionne de la manière suivante.

$\mathcal{SGC}$.Init(1^λ) Cet algorithme exécute d'abord $\mathcal{HC}$.Init(λ) pour obtenir les paramètres de la fonction caméléon $\mathcal{HC}.\mathsf{param}$. Puis il utilise $\mathcal{SG}$.Init(1^λ) afin d'obtenir les paramètres du schéma de signature de groupe $\mathcal{SG}.\mathsf{param}$, la clé secrète du gestionnaire du groupe ggsk, celle de l'autorité d'ouverture osk et enfin la clé publique du groupe gpk. On note $\mathcal{SGC}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, \mathcal{SG}.\mathsf{param}\}$ les paramètres du groupe que l'on considère, par la suite, inclus dans la clé publique de groupe gpk.

$$(\mathsf{gpk}, \mathsf{ggsk}, \mathsf{osk}) \leftarrow \mathcal{SGC}.\textsc{Init}(1^\lambda)$$

$\mathcal{SGC}$.SanGénClé$(\mathcal{SGC}.\mathsf{param})$ L'algorithme exécute $\mathcal{HC}$.GénClé(1^λ) et utilise la paire de clés obtenue comme paire de clés de délégué $(\mathsf{dsk}, \mathsf{dpk})$.

$$(\mathsf{dpk}, \mathsf{dsk}) \leftarrow \mathcal{SGC}.\textsc{SanGénClé}(\mathsf{gpk})$$

$\mathcal{SGC}$.Rejoins$(\mathsf{u}_i : \mathsf{gpk}, (\mathsf{psk}_i, \mathsf{ppk}_i); \mathsf{GG} : \mathsf{gpk}, \mathsf{ggsk})$ Ce protocole interactif entre le gestionnaire de groupe GG et un utilisateur u_i suppose que l'utilisateur est muni d'une paire de clés personnelles $(\mathsf{psk}_i, \mathsf{ppk}_i)$ et que GG est muni de sa clé secrète ggsk.
Le gestionnaire de groupe et l'utilisateur jouent le protocole interactif $\mathcal{SG}$.Rejoins de la signature de groupe $\mathcal{SG}$. L'utilisateur obtient une clé secrète d'utilisateur usk_i ainsi qu'un certificat d'appartenance au groupe cer_i. Le gestionnaire de groupe obtient de cet échange une nouvelle entrée dans sa table de registre $\mathsf{reg}[i]$ qui correspond à l'utilisateur u_i. Pour rappel, cette table est accessible à l'autorité d'ouverture.

$$(\mathsf{u}_i : \mathsf{usk}_i, \mathsf{cer}_i; \mathsf{GG} : \mathsf{reg}[i]) \leftarrow \mathcal{SGC}.\textsc{Rejoins}(\mathsf{u}_i : \mathsf{gpk}, (\mathsf{psk}_i, \mathsf{ppk}_i); \mathsf{GG} : \mathsf{gpk}, \mathsf{ggsk})$$

$\mathcal{SGC}$.Signe$(\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{dpk}, \mathsf{gpk}, \mathsf{ADM})$ Le signataire exécute la première partie de la signature de groupe afin de générer les valeurs $\{T_i\}_{i\in[1,\tau]}$ de la future signature de groupe. Il pose $\bar{\mathsf{m}} = \mathsf{m}\|T_1\|\cdots\|T_\tau$. Il choisit ensuite un aléa r dans $\{0,1\}^\lambda$ et calcule.

$$\mathsf{h} = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \bar{\mathsf{m}}, \mathsf{r})$$

Il obtient ainsi un message intermédiaire $\tilde{\mathsf{m}} = \mathsf{h}\|\mathsf{ADM}\|\mathsf{m}\|\mathsf{r}\|\mathsf{dpk}$. Puis il exécute les phases 2 et 3 de la signature de groupe afin d'obtenir $\sigma = \mathcal{SG}.\textsc{Signe}(\tilde{\mathsf{m}}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{gpk})$. Il retourne enfin la signature caméléon suivante $\Sigma = (\sigma, \mathsf{r}, \mathsf{ADM})$.

$$\Sigma = (\sigma, \mathsf{r}, \mathsf{ADM}) \leftarrow \mathcal{SGC}.\textsc{Signe}(\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{dpk}, \mathsf{gpk}, \mathsf{ADM})$$

$\mathcal{SGC}$.Modifie$(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dsk}, \mathsf{MOD}')$ Le délégué vérifie la validité de la signature Σ sur le message m. Si $\mathcal{SGC}$.Vérifie$(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{faux}$, il retourne $\perp$. Sinon, il utilise la procédure de reconstruction suivante sur la paire (Σ, m).

Procédure de reconstruction :
Cette procédure publique dépend de la forme de la signature.
- Si Σ est une signature originale, alors $\Sigma = (\sigma, \mathsf{r_{or}}, \mathsf{ADM_{or}})$ et $\mathsf{m} = \mathsf{m_{or}}$. Elle pose $\bar{\mathsf{m}}_{\mathsf{or}} = \mathsf{m_{or}} \| T_1 \| \cdots \| T_\tau$ et calcule

$$\mathsf{h} = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \bar{\mathsf{m}}_{\mathsf{or}}, \mathsf{r}).$$

  Et elle retourne enfin le message intermédiaire $\tilde{\mathsf{m}} = \mathsf{h} \| \mathsf{ADM_{or}} \| \mathsf{m_{or}} \| \mathsf{r_{or}} \| \mathsf{dpk}$ ainsi que $\bar{\mathsf{m}}_{\mathsf{or}}$.
- Sinon Σ est une signature modifiée et $\Sigma = (\sigma, \mathsf{m_{or}}, \mathsf{r_{or}}, \mathsf{ADM_{or}}, \mathsf{r}, \mathsf{MOD})$. Elle pose $\bar{\mathsf{m}} = \mathsf{m} \| T_1 \| \cdots \| T_\tau \| \mathsf{m_{or}}$ et $\bar{\mathsf{m}}_{\mathsf{or}} = \mathsf{m_{or}} \| T_1 \| \cdots \| T_\tau$ et calcule

$$\mathsf{h} = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \bar{\mathsf{m}}, \mathsf{r}).$$

  Et elle retourne le message intermédiaire $\tilde{\mathsf{m}} = \mathsf{h} \| \mathsf{ADM_{or}} \| \mathsf{m_{or}} \| \mathsf{r_{or}} \| \mathsf{dpk}$ ainsi que $\bar{\mathsf{m}}_{\mathsf{or}}$ et $\bar{\mathsf{m}}$.

Le délégué pose m' le message correspondant à m modifié selon MOD'. Il pose ensuite $\bar{\mathsf{m}}' = \mathsf{m}' \| T_1 \| \cdots \| T_\tau \| \mathsf{m_{or}}$ et génère une collision sur h grâce à sa clé secrète de délégué et à l'algorithme $\mathcal{HC}.\textsc{Forge}$ de la manière suivante.

$$\mathsf{r}' = \mathcal{HC}.\textsc{Forge}(\mathsf{dsk}, \bar{\mathsf{m}}_{\mathsf{or}}, \bar{\mathsf{m}}', \mathsf{r}, \mathsf{h})$$

Le délégué obtient ainsi la signature de groupe caméléon $\Sigma' = (\sigma, \mathsf{m_{or}}, \mathsf{r_{or}}, \mathsf{ADM_{or}}, \mathsf{r}', \mathsf{MOD}')$ sur le message m'.

$$(\Sigma' = (\sigma, \mathsf{m_{or}}, \mathsf{r_{or}}, \mathsf{ADM_{or}}, \mathsf{r}', \mathsf{MOD}'), \mathsf{m}') \leftarrow \mathcal{SGC}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dsk}, \mathsf{MOD})$$

$\mathcal{SGC}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk})$ Le vérifieur exécute la *procédure de reconstruction* décrite ci-dessus sur la paire (Σ, m) et obtient ainsi $\tilde{\mathsf{m}}$, $\bar{\mathsf{m}}_{\mathsf{or}}$ et éventuellement $\bar{\mathsf{m}}$.
S'il a obtenu $\bar{\mathsf{m}}$ alors il s'agit d'une paire modifiée. Et le vérifieur doit s'assurer que MOD correspond à ADM, que m correspond à $\mathsf{m_{or}}$ modifié selon MOD et que l'égalité suivante est vraie. Il retourne faux si ce n'est pas le cas.

$$\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \bar{\mathsf{m}}_{\mathsf{or}}, \mathsf{r_{or}}) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \bar{\mathsf{m}}, \mathsf{r})$$

Le vérifieur retourne la sortie de l'algorithme de vérification $\mathcal{S}.\textsc{Vérifie}(\mathsf{spk}, \sigma, \tilde{\mathsf{m}})$.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SGC}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk})$$

$\mathcal{SGC}.\textsc{Ouvre}(\mathsf{m}, \Sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$ L'autorité d'ouverture vérifie d'abord la signature Σ sur le message m, si $\mathcal{SGC}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{faux}$, elle retourne $\perp$. Sinon, elle utilise la *procédure de reconstruction* pour reconstruire $\tilde{\mathsf{m}}$ et retourne la sortie $(\mathsf{u}_i, \pi_\mathsf{u})$ de l'algorithme d'ouverture de la signature de groupe $\mathcal{SG}.\textsc{Ouvre}(\tilde{\mathsf{m}}, \sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{reg})$.

$$(\mathsf{u}_i, \pi_\mathsf{u}) \leftarrow \mathcal{SGC}.\textsc{Ouvre}(\mathsf{m}, \Sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$$

$\mathcal{SGC}.\textsc{OuvreJuge}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u})$ Le juge vérifie d'abord la signature Σ sur le message m, si $\mathcal{SGC}.\textsc{Vérifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{faux}$, il retourne $\perp$. Il utilise ensuite la *procédure de reconstruction* décrite ci-dessus afin d'obtenir $\tilde{\mathsf{m}}$. Enfin, il retourne la sortie de l'algorithme de vérification de preuve d'ouverture de la signature de groupe $\mathcal{SG}.\textsc{Juge}(\mathsf{gpk}, \tilde{\mathsf{m}}, \mathsf{u}_i, \pi_\mathsf{u})$.

$$\{\mathsf{vrai}, \mathsf{faux}\} \leftarrow \mathcal{SGC}.\textsc{OuvreJuge}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u})$$

$\mathcal{SGC}$.Groupe-Juge($\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk}$) L'algorithme vérifie d'abord la validité de la signature Σ sur le message m. Si $\mathcal{SGC}$.Vérifie($\Sigma, \mathsf{m}, \mathsf{gpk}, \mathsf{dpk}$) = faux, il retourne $\perp$.
Il décompose ensuite la signature. Si $\Sigma = (\sigma, \mathsf{r_{or}}, \mathsf{ADM_{or}})$ il retourne Groupe. Sinon, c'est que $\Sigma = (\sigma, \mathsf{m_{or}}, \mathsf{r_{or}}, \mathsf{ADM_{or}}, \mathsf{r}, \mathsf{ADM})$ et il retourne $\mathsf{Délégué}$.

$$\{\mathsf{Groupe}, \mathsf{Délégué}\} \leftarrow \mathcal{SGC}.\text{Groupe-Juge}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dpk})$$

Cette solution peut être instanciée dans le modèle de l'oracle aléatoire ou dans le modèle standard en fonction du modèle dans lequel les schémas de signatures de groupe et de fonction caméléon ont été prouvés. Nous avons rappelé en introduction quelques solutions possibles pour les schémas de signature de groupe. En ce qui concerne les schémas de fonction de hachage caméléon, les solutions présentées dans [Ad04] sont prouvées sûres dans le modèle de l'oracle aléatoire tandis que le schéma proposé par D.Catalano, R.Gennaro, N.Howgrave-Graham, et P.Q.Nguyen dans [CGHGN01] sous le nom "d'engagement à trappe" est prouvé sûr dans le modèle standard.

8.2.2 Preuve de sécurité

Théorème 11. *Sous l'hypothèse que le schéma de signature de groupe $\mathcal{SG}$ est sûr dans le modèle BSZ et que la fonction de hachage caméléon $\mathcal{HC}$ est **uniforme** et **résistante aux collisions**, le schéma de signature groupe caméléon est un schéma $\mathcal{NT}$-$\mathcal{SGC}$ sûr, sans **Intimité**.* ♢

Démonstration. Nous devons montrer que le schéma respecte les propriétés d'AO**Protection**, d'u**Protection**, de del**Protection** et enfin d'**Anonymat**.

Dans ces preuves, nous considérons que l'oracle de signature du challengeur peut nous donner la signature en deux morceaux. Plus précisément, nous utilisons dans les preuves la décomposition de l'oracle $\mathcal{O}.\mathcal{SG}$.Signe en deux sous-oracles reliés entre eux $\mathcal{O}.\mathcal{SG}$.Signe$_1(\mathsf{u})$ et $\mathcal{O}.\mathcal{SG}$.Signe$_2(\mathsf{m})$ tels que le premier effectue la première étape de la signature de groupe et nous donne les valeurs $\{T_i\}_{i\in[1,\tau]}$ tandis que le second effectue les étapes 2 et 3, et retourne la signature complète du message m.

De manière analogue, nous considérons que l'oracle $\mathcal{O}.\mathcal{SG}$.Challenge$_b$ peut être décomposé en deux sous-oracles reliés entre eux $\mathcal{O}.\mathcal{SG}$.Challenge$_{b1}(\mathsf{u}_0, \mathsf{u}_1)$ et $\mathcal{O}.\mathcal{SG}$.Challenge$_{b2}(\mathsf{m})$ tels que le premier effectue la première étape de la signature de groupe et nous donne les valeurs $\{T_i\}_{i\in[1,\tau]}$tandis que le second effectue les étapes 2 et 3, et retourne la signature complète du message m.

AO**Protection**. Cette propriété repose sur la traçabilité de la signature de groupe $\mathcal{SG}$ comme définie dans le modèle BSZ.
Plus formellement, nous montrons que s'il existe un adversaire contre l'AO**Protection** du schéma, que nous notons $\mathcal{A}_{\mathsf{AO}\mathcal{P}}^{\mathcal{NT}\text{-}\mathcal{SGC}}$, alors nous pouvons l'utiliser pour construire un adversaire, noté $\mathcal{A}_{\mathbf{trace},\mathcal{A}}^{\mathcal{SG}}$, contre la **Traçabilité** du schéma de signature de groupe $\mathcal{SG}$.
Au début de l'expérience, le simulateur reçoit du challengeur la clé publique du groupe gpk ainsi que la clé secrète de l'autorité d'ouverture osk. Le simulateur génère une paire de clé de délégué $(\mathsf{dsk}, \mathsf{dpk})$. Il fournit ensuite à l'adversaire $\mathcal{A}_{\mathsf{AO}\mathcal{P}}^{\mathcal{NT}\text{-}\mathcal{SGC}}$ les clés $\mathsf{dsk}, \mathsf{osk}, \mathsf{dpk}, \mathsf{gpk}$. Il simule les oracles nécessaires à $\mathcal{A}_{\mathsf{AO}\mathcal{P}}^{\mathcal{NT}\text{-}\mathcal{SGC}}$ en utilisant les oracles correspondants du challengeur.
A la fin de l'expérience, $\mathcal{A}_{\mathsf{AO}\mathcal{P}}^{\mathcal{NT}\text{-}\mathcal{SGC}}$ retourne un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$. L'adversaire $\mathcal{A}_{\mathsf{AO}\mathcal{P}}^{\mathcal{NT}\text{-}\mathcal{SGC}}$ gagne l'expérience si Σ^* est une signature caméléon valide de m^* avec la clé publique de

délégation dpk^* et qu'en notant $(\mathsf{u}, \pi_\mathsf{u})$ la sortie de $\mathcal{SGC}.\text{OUVRE}(\mathsf{m}^*, \Sigma^*, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk}^*)$ calculable par le simulateur, nous avons :

1. soit $\mathcal{SGC}.\text{OUVREJUGE}(\mathsf{m}^*, \Sigma^*, \mathsf{u}, \pi_\mathsf{u}, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{faux}$,
2. soit $(\mathsf{u}, \pi_\mathsf{u}) = \perp$.

Supposons que l'adversaire $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{AO}\mathcal{P}}$ a gagné son expérience.
Le simulateur décompose la signature Σ^* et utilise ensuite la *procédure de reconstruction* afin d'obtenir le message intermédiaire $\tilde{\mathsf{m}}^*$. L'adversaire $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{AO}\mathcal{P}}$ a pu gagner de deux manière différente.
Dans le premier cas, l'algorithme $\mathcal{SGC}.\text{OUVREJUGE}$ retourne faux. Or le seul cas où cet algorithme retourne faux est lorsque l'algorithme $\mathcal{SG}.\text{JUGE}$ retourne faux. La preuve $(\mathsf{u}, \pi_\mathsf{u})$ ayant été obtenue honnêtement en utilisant l'algorithme $\mathcal{SGC}.\text{OUVRE}$ et donc, indirectement, par l'algorithme $\mathcal{SG}.\text{OUVRE}$, le simulateur gagne l'expérience contre la **Traçabilité** de la signature de groupe en retournant la paire $(\tilde{\mathsf{m}}^*, \sigma^*)$.
Dans le second cas, l'adversaire gagne en faisant en sorte que l'algorithme $\mathcal{SGC}.\text{OUVRE}$ retourne $\perp$. Or la signature Σ^* est une signature de groupe caméléon valide du message m^*. Donc la paire $(\tilde{\mathsf{m}}^*, \sigma^*)$ est telle que l'algorithme de la signature de groupe $\mathcal{SG}.\text{OUVRE}$ a retourné $\perp$ et le simulateur gagne l'expérience contre la **Traçabilité** de la signature de groupe en retournant la paire $(\tilde{\mathsf{m}}^*, \sigma^*)$.
En conclusion, le succès d'un adversaire contre la AO**Protection** du schéma est :

$$\mathsf{Succ}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{AO}\mathcal{P},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{SG}}_{\mathbf{trace},\mathcal{A}}(\lambda).$$

□

u**Protection**. Un adversaire gagne l'expérience s'il retourne un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$ avec Σ^* une signature valide de m^* et au moins une des propositions suivantes est vraies.

1. Une partie indiquée comme non modifiable du message a été modifiée.
2. Le signataire honnête sur lequel ouvre la signature n'a jamais donné l'autorisation à ce délégué de modifier cette signature.
3. Le signataire honnête sur lequel ouvre la signature n'a jamais signé ce message et pourtant l'algorithme public $\mathcal{SGC}.$Groupe-JUGE retourne Groupe et donc indique qu'il s'agit d'une signature originale.

Nous utilisons d'une part la résistance à la non-diffamation de la signature de groupe $\mathcal{SG}$ et d'autre part le fait que la description ADM, la clé publique du délégué autorisé ainsi que le message original sont signés. Ainsi, si le délégué indiqué par dpk^* n'a jamais été autorisé, que la preuve honnêtement construite retourne Groupe alors que u n'a pas généré cette signature, ou qu'une partie non modifiable du message a été altérée, c'est que l'attaquant a, au minimum, réussi à générer une nouvelle paire message-signature de groupe sur un message que l'utilisateur n'a jamais signé.
Plus formellement, nous montrons que s'il existe un adversaire contre la u**Protection** du schéma, que nous notons $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}$, alors nous pouvons l'utiliser pour construire un adversaire, noté $\mathcal{A}^{\mathcal{SG}}_{\mathbf{nd}}$, contre la **Non-diffamation** du schéma de signature de groupe.
Au début du jeu, le simulateur reçoit du challengeur la clé publique gpk qui sera utilisée pour les signatures de l'oracle de signature $\mathcal{SG}.\text{SIGNE}$ ainsi que les clés secrètes osk de l'autorité d'ouverture et ggsk du gestionnaire de groupe. Le simulateur génère une paire de clés de délégué $(\mathsf{dsk}, \mathsf{dpk})$ puis fournit à l'adversaire $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}$ l'ensemble de clés $\{\mathsf{dsk}, \mathsf{osk}, \mathsf{ggsk}, \mathsf{gpk}, \mathsf{dpk}\}$ comme clés du schéma de signature de groupe caméléon. Le simulateur simule les oracles nécessaires à $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}$ de la manière suivante :

- $\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$. Le simulateur exécute l'algorithme $\mathcal{NT}\text{-}\mathcal{SGC}.\textsc{Signe}$ en remplaçant les appels à la signature de groupe par les appels correspondants aux sous-oracles $\mathcal{O}.\mathcal{SG}.\textsc{Signe}_1$ et $\mathcal{O}.\mathcal{SG}.\textsc{Signe}_2$. Le simulateur retourne la signature de groupe caméléon $\Sigma = (\sigma, \mathsf{r}, \mathsf{ADM})$.
- $\mathcal{O}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \{(\mathsf{m}_k, \Sigma_k)\}_{k\in[1,\mathsf{q}]})$. Dans un schéma $\mathcal{NT}\text{-}\mathcal{SGC}$ il n'y a pas d'algorithme de génération de preuve car il est publiquement possible de distinguer une signature originale d'une signature modifiée. Ainsi cet oracle n'a pas de raison d'être.
- Les deux autres oracles $\mathcal{O}.\textsc{FaisRejoindre}(\mathsf{ggsk})$ et $\mathcal{O}.\textsc{Corromps}(\mathsf{u}_i)$ ne nécessitent aucune modification du simulateur, il effectue simplement un appel à l'oracle correspondant auprès du challengeur.

A la fin de l'expérience, $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}$ retourne un triplet $(\mathsf{dpk}^*, \mathsf{m}^*, \Sigma^*)$. Le simulateur calcule la paire $(\mathsf{u}, \pi_\mathsf{u})$ correspondant à la paire (m^*, Σ^*) grâce à l'algorithme $\mathcal{SGC}.\textsc{Ouvre}$ et à la clé d'ouverture osk. L'adversaire $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}$ gagne l'expérience si Σ^* est une signature de groupe caméléon valide de m^* avec la clé publique de délégation dpk^*, que la signature Σ^* a pour origine un utilisateur u honnête, que pour tout $j \in [1, \mathsf{n}]$, nous avons $(\mathsf{m}^*, \mathsf{u}, \mathsf{dpk}^*) \neq (\mathsf{m}_j, \mathsf{u}_j, \mathsf{dpk}_j)$ et qu'une de ces deux propositions est vraie.

1. Pour tout $j \in [1, \mathsf{n}]$ tel que $\mathcal{SGC}.\textsc{Ouvre}(\mathsf{m}_j, \Sigma_j, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk}_j) = (\mathsf{u}, \pi_\mathsf{u})$, nous avons soit $\mathsf{dpk}_j \neq \mathsf{dpk}^*$, soit $\exists i \in [1, t] : \mathsf{m}_j[i] \neq \mathsf{m}^*[i]$ et $i \notin \mathsf{ADM}_j$.
2. $\mathcal{SGC}.\mathsf{Groupe\text{-}}\textsc{Juge}(\mathsf{m}^*, \Sigma^*, \pi_{\mathsf{or/mod}}, \mathsf{gpk}, \mathsf{dpk}^*)$ retourne Groupe.

Supposons que l'adversaire $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}$ a gagné son expérience.
La signature Σ^* est donc une signature de groupe caméléon valide de m^* avec la clé publique de délégation dpk^*. Le simulateur exécute la *procédure de reconstruction* et obtient $\tilde{\mathsf{m}}^* = \mathsf{h}^* \| \mathsf{ADM}^* \| \mathsf{m}^*_{\mathsf{or}} \| \mathsf{r}^*_{\mathsf{or}} \| \mathsf{dpk}^*$, $\bar{\mathsf{m}}_{\mathsf{or}}$ et, éventuellement $\bar{\mathsf{m}}$.
L'adversaire peut gagner dans deux cas.

1. Supposons que l'adversaire gagne avec la première proposition. Nous notons $O \subset [1, \mathsf{n}]$ le sous-ensemble des indices pour lesquels nous avons

$$\forall j \in O, \mathcal{SGC}.\textsc{Ouvre}(\mathsf{m}_j, \Sigma_j, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk}_j) = (\mathsf{u}, \pi_\mathsf{u}).$$

   L'adversaire ayant gagné, nous avons que, pour tout $j \in O$, soit $\mathsf{dpk}_j \neq \mathsf{dpk}^*$ et il est alors immédiat que $\tilde{\mathsf{m}}^* \neq \tilde{\mathsf{m}}_j$, soit $\exists i \in [1, t] : \mathsf{m}_j[i] \neq \mathsf{m}^*[i]$ et $i \notin \mathsf{ADM}_j$. Comme, Σ^* est une signature valide du message m^*, nous avons $\mathsf{ADM}^* \neq \mathsf{ADM}_j$ ce qui implique que $\tilde{\mathsf{m}}^* \neq \tilde{\mathsf{m}}_j$.
   Donc, si l'adversaire gagne dans ce cas, c'est qu'il a généré une signature valide ayant pour message intermédiaire $\tilde{\mathsf{m}}^*$ alors que celui-ci n'a jamais été signé par l'oracle de signature de la signature de groupe. Le simulateur gagne donc l'expérience contre la **Non-diffamation** de la signature de groupe en retournant le quadruplet $(\tilde{\mathsf{m}}^*, \sigma^*, \mathsf{u}, \pi_\mathsf{u})$.
2. Nous supposons à présent que l'adversaire n'a pas gagné avec la première proposition, comme il a gagné l'expérience nous avons :

$$\mathcal{SGC}.\mathsf{Groupe\text{-}}\textsc{Juge}(\mathsf{m}^*, \Sigma^*, \pi_{\mathsf{or/mod}}, \mathsf{gpk}, \mathsf{dpk}^*) = \mathsf{Groupe}$$

   Pour tout $\forall j \in [1, \mathsf{n}]$ nous avons $(\mathsf{m}^*, \mathsf{u}, \mathsf{dpk}^*) \neq (\mathsf{m}_j, \mathsf{u}_j, \mathsf{dpk}_j)$ donc pour tout j tel que $\mathsf{u} = \mathsf{u}_j$ nous avons $(\mathsf{m}^*, \mathsf{dpk}^*) \neq (\mathsf{m}_j, \mathsf{dpk}_j)$. Nous avons vu plus haut que si $\mathsf{dpk}_j \neq \mathsf{dpk}^*$ alors $\tilde{\mathsf{m}}^* \neq \tilde{\mathsf{m}}_j$. Nous supposons donc que $\mathsf{dpk}_j = \mathsf{dpk}^*$ et donc que $\mathsf{m}_j \neq \mathsf{m}^*$. L'algorithme $\mathcal{SGC}.\mathsf{Groupe\text{-}}\textsc{Juge}$ retourne Groupe uniquement si la

signature Σ^* est de la forme $(\sigma^*, \mathsf{r}^*, \mathsf{ADM}^*)$ et donc que $\mathsf{m}^*_{\mathsf{or}} = \mathsf{m}^*$. En ce qui concerne les requêtes, il s'agissait uniquement de signatures originales, nous avons donc pour tout j tel que $\mathsf{u} = \mathsf{u}_j$, $\mathsf{m}_{j_{\mathsf{or}}} = \mathsf{m}_j$. Et nous avons $\tilde{\mathsf{m}}_j \neq \tilde{\mathsf{m}}^*$. Et, le simulateur gagne l'expérience contre la **Non-diffamation** de la signature de groupe en retournant le quadruplet $(\tilde{\mathsf{m}}^*, \sigma^*, \mathsf{u}, \pi_{\mathsf{u}})$.

En conclusion, le succès d'un adversaire contre la u**Protection** du schéma est :

$$\mathsf{Succ}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}(\lambda) \leq \mathsf{Succ}^{\mathcal{SG}}_{\mathbf{nd},\mathcal{A}}(\lambda).$$

□

del**Protection**. Un adversaire gagne l'expérience s'il retourne un triplet $(\pi^*_{\mathsf{or/mod}}, \mathsf{m}^*, \Sigma^*)$ avec Σ^* une signature valide de m^* ayant pour origine l'utilisateur u, d'après l'algorithme $\mathcal{SGC}$.Ouvre$(\mathsf{m}^*, \Sigma^*, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk})$, et tel que les propositions suivantes soient vraies.

1. $\forall j \in [1, \mathsf{n}]$,
   soit $(\mathsf{m}^*, \mathsf{u}) \neq (\mathsf{m}'_j, \mathsf{u}_j)$,
   soit $(\mathsf{m}^*, \mathsf{u}) = (\mathsf{m}'_j, \mathsf{u}_j)$ avec $\mathsf{m}^*_{\mathsf{or}} \neq \mathsf{m}_{j_{\mathsf{or}}}$.
2. $\mathcal{SGC}$.Groupe-Juge$(\mathsf{m}^*, \Sigma^*, \pi^*_{\mathsf{or/mod}}, \mathsf{gpk}, \mathsf{dpk}) = \mathsf{Délégué}$.

Nous nous appuyons sur la résistance aux collisions de la fonction de hachage caméléon $\mathcal{HC}$.

Plus formellement, nous montrons que s'il existe un adversaire contre la del**Protection** du schéma, que nous notons $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{del}\mathcal{P}}$, alors nous pouvons l'utiliser pour construire un adversaire, noté $\mathcal{A}^{\mathcal{HC}}_{\mathbf{collRes}}$, contre la **résistance aux collisions** du schéma de fonction de hachage caméléon $\mathcal{HC}$.

Au début du jeu, le simulateur reçoit du challengeur la clé publique dpk et les paramètres $\mathcal{HC}.\mathsf{param}$ de la fonction caméléon. Le simulateur utilise ensuite $\mathcal{SG}$.Init(1^λ) afin d'obtenir les paramètres du schéma de signature de groupe $\mathcal{SG}.\mathsf{param}$, la clé secrète du gestionnaire du groupe ggsk, celle de l'autorité d'ouverture osk et enfin la clé publique du groupe gpk. On note $\mathcal{SGC}.\mathsf{param} = \{\mathcal{HC}.\mathsf{param}, \mathcal{SG}.\mathsf{param}\}$ les paramètres que le simulateur inclus dans gpk. Le simulateur fournit à l'adversaire $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{del}\mathcal{P}}$ l'ensemble de clés $\{\mathsf{osk}, \mathsf{ggsk}, \mathsf{gpk}, \mathsf{dpk}\}$ comme clés du schéma de signature de groupe caméléon, puis simule les deux oracles de la manière suivante :

– $\mathcal{O}$.FaisRejoindre(ggsk). Le simulateur génère une paire de clé $(\mathsf{psk}_i, \mathsf{ppk}_i)$ pour l'utilisateur u_i puis joue le protocole interactif $\mathcal{SG}$.Rejoins de la signature de groupe avec l'adversaire. Il stock ensuite la clé usk_i et le certificat cert_i correspondant.
– $\mathcal{O}$.Modifie$(\mathsf{m}, \Sigma, \mathsf{MOD})$. Le simulateur vérifie que MOD correspond à ADM et, si ce n'est pas le cas, il retourne $\perp$. Il exécute ensuite le protocole $\mathcal{NT}$-$\mathcal{SGC}$.Modifie en utilisant l'oracle $\mathcal{O}.\mathcal{HC}$.Forge pour obtenir r' et obtenir ainsi la collision souhaitée sur le haché caméléon. Il retourne ensuite la signature de groupe caméléon :

$$\Sigma' = (\sigma, \mathsf{m}_{\mathsf{or}}, \mathsf{r}_{\mathsf{or}}, \mathsf{ADM}, \mathsf{r}', \mathsf{MOD}).$$

A la fin de l'expérience, $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{u}\mathcal{P}}$ retourne un triplet $(\pi^*_{\mathsf{or/mod}}, \mathsf{m}^*, \Sigma^*)$ avec Σ^* une signature sur le message m^* et $\pi^*_{\mathsf{or/mod}}$ une preuve qu'il s'agit d'une paire modifiée ou originale. Le simulateur calcule la paire $(\mathsf{u}, \pi_{\mathsf{u}})$ qui correspond à la paire (m^*, Σ^*) grâce à l'algorithme $\mathcal{SGC}$.Ouvre et à la clé d'ouverture osk.

L'algorithme $\mathcal{SGC}$.Groupe-Juge retourne $\mathsf{Délégué}$ seulement si la signature Σ^* est une signature de groupe caméléon valide de m^*. Dans ce cas Σ^* est composée de la façon

suivante : $\Sigma^* = (\sigma^*, \mathsf{m}^*_{\mathsf{or}}, \mathsf{r}^*_{\mathsf{or}}, \mathsf{ADM}^*_{\mathsf{or}}, \mathsf{r}^*, \mathsf{ADM}^*)$ et $\forall j \in [1, \mathsf{n}]$, nous avons un des deux cas suivants.

1. $(\mathsf{m}^*, \mathsf{u}) \neq (\mathsf{m}'_j, \mathsf{u}_j)$.
   Supposons que $\mathsf{m}^* \neq \mathsf{m}'_j$, alors $\bar{\mathsf{m}}_j \neq \bar{\mathsf{m}}^*$ car $\bar{\mathsf{m}}_j = \mathsf{m}'_j \| T_{j_1} \| \cdots \| T_{j_\tau} \| \mathsf{m}_{\mathsf{or}}$ et Σ^* est une signature de groupe caméléon valide de m^*.
   Sinon, nous avons $\mathsf{u}^* \neq \mathsf{u}_j$. Les certificats d'appartenance aux groupes impliqués dans σ^* et σ_j sont donc différents. Or pour un schéma de chiffrement consistant, deux chiffrés sont égaux uniquement si les clairs sont eux-aussi égaux. Ainsi, nous avons $\{T^*_i\}_{i\in[1,n]} \neq \{T_{j_i}\}_{i\in[1,n]}$. Or $\bar{\mathsf{m}}_j = \mathsf{m}'_j \| T_{j_1} \| \cdots \| T_{j_\tau} \| \mathsf{m}_{\mathsf{or}}$ et Σ^* est une signature de groupe caméléon valide de m^*. Donc $\bar{\mathsf{m}}_j \neq \bar{\mathsf{m}}^*$.
2. $(\mathsf{m}^*, \mathsf{u}) = (\mathsf{m}'_j, \mathsf{u}_j)$ avec $\mathsf{m}^*_{\mathsf{or}} \neq \mathsf{m}_{j_{\mathsf{or}}}$. Comme $\bar{\mathsf{m}}_j = \mathsf{m}'_j \| T_{j_1} \| \cdots \| T_{j_\tau} \| \mathsf{m}_{j_{\mathsf{or}}}$ et que Σ^* est une signature de groupe caméléon valide de m^* nous avons, ici aussi, $\bar{\mathsf{m}}_j \neq \bar{\mathsf{m}}^*$.

En conclusion, $\forall j \in [1, \mathsf{n}], \bar{\mathsf{m}}_j \neq \bar{\mathsf{m}}^*$ donc $\bar{\mathsf{m}}^*$ est une collision qui n'a jamais été demandé à l'oracle $\mathcal{O}.\mathcal{HC}.$FORGE. Et le simulateur gagne l'expérience contre la **résistance aux collisions** de la fonction caméléon en retournant $(\mathsf{dpk}, \bar{\mathsf{m}}^*, \mathsf{r}^*, \bar{\mathsf{m}}_{\mathsf{or}}, \mathsf{r}_{\mathsf{or}})$.
Le succès d'un adversaire contre la del**Protection** du schéma est :

$$\mathsf{Succ}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathsf{del}\mathcal{P}}(\lambda) \leq \mathsf{Succ}^{\mathcal{HC}}_{\mathbf{collRes},\mathcal{A}}(\lambda)$$

□

Anonymat Nous utilisons l'anonymat du schéma de signature de groupe $\mathcal{SG}$ comme défini dans le modèle BSZ.
Plus formellement, nous montrons que s'il existe un adversaire contre l'**Anonymat** du schéma, que nous notons $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathbf{anon}}$, alors nous pouvons l'utiliser pour construire un adversaire, noté $\mathcal{A}^{\mathcal{SG}}_{\mathbf{anon}}$, contre l'**Anonymat** du schéma de signature de groupe $\mathcal{SG}$.
Au début de l'expérience, le simulateur reçoit du challengeur la clé publique du groupe gpk et la clé secrète de gestionnaire de groupe ggsk qu'il fournit à l'adversaire $\mathcal{A}^{\mathcal{SG}}_{\mathbf{anon}}$. Il utilise directement les oracles du challengeur pour simuler les réponses aux oracles $\mathcal{O}.$FAISREJOINDRE et $\mathcal{O}.$CORROMPS. L'oracle $\mathcal{O}.$PROUVE n'a pas d'algorithme équivalent ici et n'est donc pas considéré. Pour les deux oracles restant, le simulateur réagit de la façon suivante.

- $\mathcal{O}.\textsc{Signe}(\mathsf{m}, \mathsf{u}_i, \mathsf{dpk}, \mathsf{ADM})$. Le simulateur effectue l'algorithme $\mathcal{NT}\text{-}\mathcal{SGC}.$SIGNE en remplaçant les appels à la signature de groupe par les appels correspondants aux sous-oracles $\mathcal{O}.\mathcal{SG}.\textsc{Signe}_1$ et $\mathcal{O}.\mathcal{SG}.\textsc{Signe}_2$. Le simulateur retourne la signature de groupe caméléon $\Sigma = (\sigma, \mathsf{r}, \mathsf{ADM})$.
- $\mathcal{O}.\textsc{Challenge}_b(\mathsf{u}_0, \mathsf{u}_1, \mathsf{m}, \mathsf{dpk}, \mathsf{ADM})$. Le simulateur effectue $\mathcal{NT}\text{-}\mathcal{SGC}.$SIGNE pour l'utilisateur u_b en remplaçant les appels à la signature de groupe par les appels correspondants aux sous-oracles $\mathcal{O}.\mathcal{SG}.\textsc{Challenge}_{b1}(\mathsf{u}_0, \mathsf{u}_1)$ et $\mathcal{O}.\mathcal{SG}.\textsc{Challenge}_{b2}(\tilde{\mathsf{m}})$. Le simulateur retourne la signature de groupe caméléon $\Sigma = (\sigma, \mathsf{r}, \mathsf{ADM})$.

A la fin de l'expérience, $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathbf{anon}}$ retourne un bit b^*. Il gagne l'expérience si $b^* = b$, c'est-à-dire si c'est u_{b^*} qui est à l'origine de la signature Σ émise par l'oracle de challenge $\mathcal{O}.\textsc{Challenge}_b$. Supposons que l'adversaire $\mathcal{A}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathbf{anon}}$ a gagné son expérience.
L'utilisateur u_{b^*} est à l'origine de la signature Σ seulement si u_{b^*} est à l'origine de la signature de groupe σ émise par l'oracle de challenge $O.\mathcal{SG}.$CHALLENGE. Le simulateur retourne simplement b^* pour gagner l'expérience contre l'**Anonymat** de la signature de groupe.
En conclusion, le succès d'un adversaire contre l'**Anonymat** du schéma est :

$$\mathsf{Succ}^{\mathcal{NT}\text{-}\mathcal{SGC}}_{\mathbf{anon},\mathcal{A}}(\lambda) \leq \mathsf{Succ}^{\mathcal{SG}}_{\mathbf{anon},\mathcal{A}}(\lambda).$$

□

8.3 Principes des autres solutions

Je vais à présent donner les principes généraux sous-tendant chacune de nos autres solutions. Par soucis de lisibilité, je ne donnerai pas les définitions formelles de ces schémas.

8.3.1 Principe pour une solution non transparente avec intimité

À partir du schéma $\mathcal{NT}$-$\mathcal{SGC}$ que je viens de décrire, il est possible d'atteindre la notion d'intimité en utilisant un engagement de Pedersen pour masquer le message original de manière efficace et engageante.

Soit $\mathbb{G}$ un groupe d'ordre $p = 2q+1$ avec p et q premiers. Et soient g et h deux générateurs de ce groupe choisis aléatoirement et publiés durant l'initialisation du schéma. Le signataire cache son message original de la façon suivante.

1. Pour chaque partie modifiable, le signataire choisit aléatoirement un aléa.

$$\forall i \in \mathsf{ADM}, \mathsf{r}_i \in \{0,1\}^\lambda$$

2. Puis, pour chaque partie du message, le signataire génère :

$$\forall i \in [1,\mathsf{t}], \hat{\mathsf{m}}_i = \begin{cases} g^{\mathsf{m}_i} h^{\mathsf{r}_i} & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i & \text{sinon} \end{cases}$$

3. Il obtient ainsi le message $\hat{\mathsf{m}} = \hat{\mathsf{m}}_1 \| \cdots \| \hat{\mathsf{m}}_\mathsf{t}$.

Le signataire agit ensuite avec le message $\hat{\mathsf{m}}$ comme s'il s'agissait de son message original m. Nous notons $\mathsf{R} = \{\mathsf{r}_i\}_{i \in \mathsf{ADM}}$ l'ensemble des aléas. R est alors ajouté à la signature originale, ce qui permet de reconstituer $\hat{\mathsf{m}}$ et donc de vérifier la signature. Lors d'une modification, R est retiré de la signature.

Le reste du schéma est simplement adapté en conséquence afin de prendre en compte que le message original est préparé avant d'être signé et que seule la version $\hat{\mathsf{m}}$ du message est utilisée dans la signature.

Sécurité du schéma. Ce schéma respecte les propriétés d'AO**Protection**, de del**Protection**, d'**Anonymat** pour les même raisons que le schéma sans intimité.

En ce qui concerne l'u**Protection**, le schéma atteint cette propriété de manière similaire au schéma initial à une exception prêt. En effet nous devons considérer le cas où l'attaque cible l'engagement $g^{\mathsf{m}_i} h^{\mathsf{r}_i}$. Dans ce cas, l'adversaire exhibe une paire $(\mathsf{m}_i^*, \mathsf{r}_i^*)$ telle que $g^{\mathsf{m}_i^*} h^{\mathsf{r}_i^*} \neq g^{\mathsf{m}_{j_i}} h^{\mathsf{r}_{j_i}}$ pour $\mathsf{m}_i^* \neq \mathsf{m}_{j_i}$. Or cela implique que l'adversaire a obtenu le logarithme discret de h en base g. En effet, en posant $x = (\mathsf{r}_i^* - \mathsf{r}_{j_i})(\mathsf{m}_{j_i} - \mathsf{m}_i^*)^{-1} \bmod p$ nous avons $h = g^x$. Comme h a été choisi aléatoirement, l'adversaire peut être utilisé pour résoudre le problème du logarithme discret dans un groupe d'ordre premier fort.

Enfin, l'**Intimité** repose sur le fait que l'engagement de Pedersen que nous utilisons est parfaitement **Indistinguable**. En effet s'il existe un adversaire capable de distinguer si une signature modifiée provient d'un message $\mathsf{m}_{\mathsf{or}_0}$ ou d'un message $\mathsf{m}_{\mathsf{or}_1}$ alors celui-ci pourra être utilisé pour construire un adversaire capable de distinguer quel message a été engagé parmi deux de son choix. Les aléas n'étant plus fournis dans une signature modifiée, distinguer qu'un message modifié provient d'un message $\mathsf{m}_{\mathsf{or}_0}$ ou d'un message $\mathsf{m}_{\mathsf{or}_1}$ qui ne différent que sur les parties engagées revient à distinguer si $\hat{\mathsf{m}}_i = g^{\mathsf{m}_{\mathsf{or}_i}} h^{\mathsf{r}_{\mathsf{or}_i}}$ ou si $\hat{\mathsf{m}}_i = g^{\mathsf{m}_{1_i}} h^{\mathsf{r}_{1_i}}$. Or sachant que pour $\hat{\mathsf{m}}_i$, les deux aléas existent et que r_{b_i} a été choisi aléatoirement et n'est pas divulgué, ceci est parfaitement impossible.

8.3.2 Principe pour une solution "fortement" transparente

Nous nous intéressons à présent à un schéma pour lequel un couple message-signature original ne peut être distingué d'un couple message-signature modifié que par le signataire ou le délégué impliqué dans la signature.

L'idée de cette construction est de combiner le principe du schéma de signature caméléon sûr dans le modèle de Brzuska *et al.* que nous avons présenté au Chapitre 6 avec un schéma de signature de groupe sûr dans le modèle [BSZ05]. Nous donnons ici les grandes lignes de notre schéma.

Cette construction suppose que chaque signataire se comporte comme un membre d'un groupe dans une signature de groupe. Nous considérons donc qu'ils se sont dûment enregistrés auprès du gestionnaire de groupe et qu'ils possèdent donc une clé secrète d'utilisateur usk_i et un certificat d'appartenance au groupe cert_i. Nous supposons, d'autre part, que la table de registre correspondante a été mise en place et que celle-ci est accessible en lecture à l'autorité d'ouverture.

Signe. Pour signer un message m, le signataire commence par effectuer la première phase de la signature de groupe afin d'obtenir le chiffré, pour l'autorité d'ouverture, de son certificat d'appartenance, c'est-à-dire les valeurs $\{T_i\}_{i\in[1,\tau]}$. Il génère ensuite un TAG pour son message de manière similaire au schéma de signature caméléon décrit au Chapitre 6. Il choisit aléatoirement $\mathsf{Nonce} \in \{0,1\}^\lambda$, calcule $\mathsf{x} = \mathcal{PRF}(\kappa_i, \mathsf{Nonce})$ et obtient $\mathsf{TAG} = \mathcal{PRG}(\mathsf{x})$.

Ensuite, le signataire choisit $|\mathsf{ADM}| + 1$ aléas dans $\{0,1\}^\lambda$. Nous notons cet ensemble $\mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}, \mathsf{r}_c\}$. Le signataire exécute alors une *procédure de reconstruction* sur le message m.

Procédure de reconstruction :
Cette procédure publique utilise TAG, les aléas R, les valeurs $\{T_i\}_{i\in[1,\tau]}$, la clé publique du délégué dpk et celle du groupe gpk :

1. Pour chaque partie on calcule une valeur intermédiaire $\tilde{\mathsf{m}}_i$.
$$\forall i \in [1,\mathsf{t}], \tilde{\mathsf{m}}_i = \begin{cases} \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{m}_i||i, \mathsf{r}_i) & \text{si } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i||i & \text{sinon} \end{cases}$$
2. On pose $\bar{\mathsf{m}} = \mathsf{TAG}\|\mathsf{m}\|T_1\|\cdots\|T_\tau$.
3. Puis on calcule la partie finale supplémentaire.
$$\mathsf{h}_c = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \bar{\mathsf{m}}, \mathsf{r}_c)$$

Le message reconstitué est $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1\|\cdots\|\tilde{\mathsf{m}}_\mathsf{t}\|\mathsf{h}_c\|\mathsf{dpk}$.

Il exécute ensuite les phases 2 et 3 de la signature de groupe afin d'obtenir la signature $\sigma = \mathcal{SG}.\textsc{Signe}(\tilde{\mathsf{m}}, (\mathsf{ssk}_i, \mathsf{cer}_i), \mathsf{gpk})$ sur le message $\tilde{\mathsf{m}}$. La signature de groupe caméléon est

$$\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}) \text{ avec } \mathsf{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_{|\mathsf{ADM}|}, \mathsf{r}_c\}.$$

La signature Σ et le message correspondant m sont ajoutés à la base de données de l'utilisateur BD_i.

$$\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}) \leftarrow \mathcal{SGC}.\textsc{Signe}(\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{dpk}, \mathsf{gpk}, \mathsf{ADM})$$

Modifie. Un délégué souhaitant modifier le message agit de manière similaire au schéma de signature caméléon classique. Il vérifie que MOD correspond à ADM et, si ce n'est pas le cas, il s'arrête en retournant $\perp$. Il choisit ensuite aléatoirement Nonce' et TAG' dans $\{0,1\}^\lambda$ avant de construire les collisions sur les parties qu'il souhaite modifier, ainsi que sur la partie finale, en utilisant sa clé secrète de délégué. La signature modifiée est $\Sigma' = (\sigma, \mathsf{TAG}', \mathsf{Nonce}', \mathsf{R}', \mathsf{ADM}')$.

$$(\Sigma' = (\sigma, \mathsf{TAG}', \mathsf{Nonce}', \mathsf{R}', \mathsf{ADM}'), \mathsf{m}') \leftarrow \mathcal{SGC}.\textsc{Modifie}(\mathsf{m}, \Sigma, \mathsf{gpk}, \mathsf{dsk}, \mathsf{MOD})$$

Prouve. De manière analogue aux signatures caméléons classiques, c'est le signataire à l'origine d'une signature qui put émettre une preuve indiquant si une signature est originale ou non. En considérant qu'il s'est assuré de la validité des signatures et des preuves qui lui sont fournies, il utilise sa base de données pour prouver si une paire n'est pas originale. L'objectif est alors d'exhiber une paire (m_k, Σ_k) telle que :

- $\mathcal{SGC}.\textsc{Vérifie}(\mathsf{m}_k, \Sigma_k, \mathsf{gpk}, \mathsf{dpk})$ retourne vrai ;
- $\mathsf{m}_k \neq \mathsf{m}$;
- $\mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG} \| \mathsf{m} \| T_1 \| \cdots \| T_\tau, \mathsf{r}_c) = \mathcal{HC}.\textsc{Hache}(\mathsf{dpk}, \mathsf{TAG}_k \| \mathsf{m}_k \| T_{k1} \| \cdots \| T_{k\tau}, \mathsf{r}_{c_k})$ avec $\mathsf{TAG}_k = \mathcal{PRG}(x_k)$ pour $x_k = \mathcal{PRF}(\kappa_i, \mathsf{Nonce}_k)$.

S'il trouve une telle paire alors la preuve est le triplet $\pi_{\mathsf{or/mod}} = (\mathsf{m}_k, x_k, \Sigma_k)$ sinon $\pi_{\mathsf{or/mod}} = \perp$.

$$\pi_{\mathsf{or/mod}} \leftarrow \mathsf{u}\text{-}\mathcal{SGC}.\textsc{Prouve}(\mathsf{m}, \Sigma, \mathsf{u}_i, \pi_\mathsf{u}, \mathsf{usk}_i, \mathsf{gpk}, \mathsf{dpk}, \mathsf{BD} = \{(\mathsf{m}_k, \Sigma_k)\}_{k \in [1,\mathsf{q}]})$$

Remarque. L'architecture de la procédure de reconstruction que nous utilisons pour ce schéma permet d'appliquer les différentes extensions que nous avons proposées au chapitre précédent. Il est ainsi possible de construire un schéma de signature de groupe caméléon pour lequel les parties modifiables sont limitées dans un ensemble ou pour lequel le signataire peut contrôler le nombre de versions de sa signature ou le nombre de parties modifiables.

Sécurité du schéma. Ce schéma respecte les propriétés d'AO**Protection** et d'**Anonymat** de manière similaire à notre schéma $\mathcal{NT}$-$\mathcal{SGC}$.

L' u**Protection** de ce schéma repose sur la propriété de non-diffamation de la signature de groupe $\mathcal{SG}$ et sur le fait que les parties non modifiables, la clé publique du délégué autorisé et le message intermédiaire sont signés. Ainsi, si le délégué indiqué par dpk^* n'a jamais été autorisé, qu'une partie non modifiable du message a été altérée ou que la preuve honnêtement construite retourne u alors que u n'a pas généré cette signature, c'est que l'attaquant a, au minimum, réussi à construire une signature de groupe s'ouvrant sur un utilisateur n'ayant jamais signé ce message.

La propriété de del**Protection** s'appuie sur la résistance aux collisions de la fonction de hachage caméléon $\mathcal{HC}$ et sur le fait que $\mathcal{PRG}$ soit une fonction à sens unique de manière similaire à notre schéma de signature caméléon.

Enfin, la **Transparence** forte provient du fait que les seules différences entre une signature originale et une signature modifiée résident dans la construction du TAG et des aléas de R de manière identique à la transparence de notre schéma de signature caméléon classique présenté au Chapitre 6.

8.3.3 Principe pour une solution à transparence de groupe

Le développement d'un schéma à transparence de groupe est, pour l'instant, au stade du travail en cours. Je présenterai ici comment construire une solution partielle au problème. Dans ce cas, un signataire peut faire en sorte que toutes les signatures modifiées produites à partir de

ses signatures originales soient considérées comme des originaux. Notons que la responsabilité finale, c'est-à-dire après levée d'anonymat, repose toujours sur le signataire.

L'idée est de se baser sur les schémas que nous avons présentés au cours de ce chapitre. Le schéma de signature de groupe caméléon à transparence forte est le plus proche, car il assure déjà une forme de transparence. Cependant il faut y apporter les modifications nécessaires et suffisantes pour que tout membre du groupe puisse répondre à une requête de preuve.

Une première méthode est de faire en sorte que tous les membres du groupe aient accès aux bases de données à jour des autres signataires. Il faudrait alors considérer que les valeurs x permettant de construire les TAG y soient inscrites.

Une seconde solution implique que chaque signature comporte les éléments nécessaires à un membre du groupe pour se convaincre que la signature est originale ou non et prouver cette conviction. Seuls les membres du groupe devant pouvoir accéder à ces informations, nous proposons l'utilisation d'un chiffrement à clé publique déchiffrables par tous les membres et uniquement par eux. Le principe généralement utilisé pour prouver si une signature est originale ou non est d'exhiber la signature originale, nous obtenons donc la construction suivante.

Supposons que $\Sigma = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM})$ est la signature que nous aurions obtenue pour le message original $\mathsf{m_{or}}$ avec notre solution $\mathcal{FT}$-$\mathcal{SGC}$. Nous chiffrons la signature originale, que nous notons c_σ, et le message associé, que nous notons $\mathsf{c}_{\mathsf{m_{or}}}$. Une signature de groupe caméléon à transparence de groupe est alors composée de la signature Σ comme construite précédemment et des deux chiffrés, c'est-à-dire $\Sigma_{GT} = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}, \mathsf{c}_\sigma, \mathsf{c}_{\mathsf{m_{or}}})$. Le problème est que, dans ce cas, un signataire peut tricher sur ce qu'il chiffre et un délégué ne souhaitant pas assumer un message pourrait remplacer le chiffré par un aléa au bon format. Les signataires ne peuvent donc pas se convaincre que le délégué ou le signataire est à l'origine du chiffré mal formé et sont donc incapable de prouver quoi que se soit.

Une solution consiste à adapter ce principe de telle manière que les chiffrés soient signés par le signataire originale. Par exemple on chiffre le message original $\mathsf{m_{or}}$, la variable $\mathsf{ADM_{or}}$, les aléas $\mathsf{R_{or}}$, le triplet $(\mathsf{TAG}, \mathsf{Nonce}, x)$ et, enfin, le chiffré du certificat d'appartenance $\{T_i\}_{i\in[1,\tau]}$. On note c_{GT} le chiffré obtenu. Le message intermédiaire est modifié par l'ajout de ce chiffré. La signature de groupe σ est ainsi une signature de groupe sur le message intermédiaire $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_\mathsf{t} \| \mathsf{h}_c \| \mathsf{dpk} \| \mathsf{c}_{GT}$. Enfin, la signature de groupe caméléon obtenue est $\Sigma_{GT} = (\sigma, \mathsf{TAG}, \mathsf{Nonce}, \mathsf{R}, \mathsf{ADM}, \mathsf{c}_{GT})$.

Les membres du groupe des signataires peuvent ainsi déchiffrer c_{GT} et reconstituer le couple message-signature original. c_{GT} étant simplement signé par le signataire, le délégué est obligé de garder cette élément intact pour conserver une signature valide. Ainsi, si le chiffré est mal formé, le signataire devient responsable de tous les messages modifiés car le groupe pourra prouver qu'il a correctement déchiffré les éléments et que ceux-ci proviennent du chiffré correctement signé. On considère alors que le signataire a choisi d'endosser cette responsabilité en n'indiquant pas de message original.

Nous travaillons encore à la mise en place d'une solution qui respecte la transparence de groupe mais qui assure qu'un signataire ne soit pas capable d'endosser la responsabilité de toutes les signatures modifiées.

Conclusion

Nous avons vu au cours de ce chapitre qu'il est possible d'utiliser les signatures caméléons dans le contexte d'un groupe de signataires et ainsi permettre à ceux-ci de bénéficier des propriétés d'anonymat des signatures de groupe. J'ai décrit ici un nouveau modèle ainsi qu'un ensemble

de schémas qui permettent de s'adapter à différentes configurations en fonction de l'importance des notions de groupe et de transparence pour nos signatures.

Cette partie consacrée aux signatures caméléons a commencé sur les signatures caméléons simple au premier chapitre, j'ai alors montré les limites de l'état de l'art dans le modèle classique pour ce type de signature puis proposé une solution sûre et efficace. Au second chapitre, nous avons vu comment modéliser et construire des schémas permettant au signataire d'encadrer les modifications du délégué. Et j'ai montré ici mes travaux pour modéliser et développer les signatures caméléons dans le contexte des signatures de groupe. Je travaille à présent, avec S.Canard, à l'approfondissement des signatures de groupes caméléons mais aussi aux moyens de construire de tels schémas pour lesquels un signataire défini non pas un délégué mais un groupe de délégués. Cette problématique implique non seulement un nouveau type de modèle mais aussi un changement radical en ce qui concerne l'architecture des schémas qui considère jusqu'à présent qu'un délégué est coupable seulement si les signataires prouvent leur innocence.

Quatrième partie

Applications

Chapitre 9

Facturation anonyme

Sommaire

La concept de facturation anonyme propose d'ajouter aux systèmes de facturations de services classiques les moyens de respecter la vie privée des utilisateurs.

Le contexte général de la facturation considère trois entités : le consommateur (aussi appelé utilisateur), le fournisseur de services et le fournisseur de facture. Le consommateur obtient auprès du fournisseur de services un certain nombre de services payants au cours d'une période de temps déterminée. Ce fournisseur est capable de rendre plusieurs services ayant éventuellement un coût différent d'un service à un autre. A la fin de chaque période, le fournisseur de facture envoie une facture au consommateur afin d'obtenir le paiement pour les services.

Cette architecture représente un modèle économique courant dans lequel les deux fournisseurs peuvent représenter différents services d'une même entreprise ou deux entreprises distinctes. Le second cas se retrouve, par exemple, lors de paiement de services sur facture téléphonique. Le consommateur acquiert des services auprès de différents fournisseurs ayant un contrat avec un opérateur, le paiement de ceux-ci s'effectue en fin de mois sur la facture téléphonique.

Nous souhaitons intégrer à cette architecture un haut niveau de respect de la vie privée des utilisateurs. Le fournisseur de services a besoin de connaitre la liste des services souhaités et de s'assurer que l'utilisateur s'engage à payer. Cependant il n'est pas nécessaire qu'il ait accès à l'identité de l'utilisateur. Dans un cas idéal, l'utilisateur sera donc anonyme et intraçable vis-à-vis du fournisseur de services. Par contre, lorsque l'utilisateur commande un service, il ne doit pas pouvoir nier qu'il est à l'origine de sa commande. Le fournisseur de facture a, pour sa part, besoin de connaitre l'identité de l'utilisateur et les données comptables mais il n'est pas nécessaire qu'il connaisse le détail des services achetés. Nous souhaitons donc que celui-ci soit incapable de les retrouver. Enfin, en cas de problème, l'utilisateur doit pouvoir contester la facture reçue.

L'objectif est donc de fournir une facture cohérente et correcte au consommateur sans pour autant que le fournisseur de services connaisse son identité réelle, que le fournisseur de facture apprenne quels services ont été achetés ou que le consommateur puisse nier avoir consommé ces services.

Dans ce chapitre, je présente la solution à laquelle j'ai travaillé avec S.Canard et E.Malville. Cette solution a fait l'objet du brevet [CJM09]. Chronologiquement, cette application est à l'origine de mes travaux sur les signatures de groupes caméléons présentés au Chapitre 8. En effet, nous verrons qu'une solution efficace à notre problème est d'utiliser une signature de groupe caméléon non-transparente. Cette solution a été proposée, en tant qu'application des signatures de groupe caméléons, dans un article avec S.Canard en cours de soumission [CJ] (voir Annexe B).

9.1 Définition et modèle

Nous considérons un fournisseur de services FS. Chaque service s est payant, on note α_s le code de facturation correspondant. Pour l'ensemble S_k des services s_i ayant le même tarif, nous avons $\forall i, j \in S_k, \alpha_{s_i} = \alpha_{s_j}$.

Soient un utilisateur u et un fournisseur de services FS, l'objectif est que l'utilisateur obtienne le service s souhaité auprès du fournisseur FS en restant anonyme vis-à-vis de celui-ci. Ensuite, le fournisseur FS fourni au fournisseur de facture FF le code de facturation α_s afin d'être payé pour le service rendu. Nous supposons que le code de facturation ne révèle aucune donnée sur le service choisi, chaque code pouvant se référer à un ensemble de services ayant la même tarification. Le fournisseur de facture FS ayant accès à l'identité de l'utilisateur u et aux montants à payer peut prendre contact avec l'utilisateur pour les aspects financiers sans pour autant connaitre quels services ont été achetés. Enfin, en cas de problème un juge peut retrouver et prouver comment le protocole s'est déroulé et quelle entité a effectué quelle action.

Par abus de langage, nous parlerons dans ce chapitre du service s lorsqu'il sera question du service ayant pour identifiant unique s. Et, afin de simplifier la lecture, dés que le contexte le permet, nous noterons i le service s_i et α_i le tarif correspondant au service i.

9.1.1 État de l'art

Les solutions de monnaie électronique [CHL05, CDG+09] et de coupons [CES+05, CGH06] ne sont pas considérées dans cette étude. En effet, les travaux sur la monnaie électronique considèrent que l'utilisateur est anonyme non seulement vis-à-vis du marchand mais aussi vis-à-vis de la banque chez qui le marchand va déposer l'argent. Ainsi, ce système n'est pas adapté à la mise en place de factures. D'autre part, les solutions de coupons considèrent un modèle de prépaiement dans lequel l'utilisateur acquiert d'abord un ensemble de coupons avant de les utiliser pour accéder à un service de façon anonyme.

La solution W-HA [WHA], le système de facturation d'Orange, propose au consommateur de créer, auprès d'un tiers, un compte qui lui permet ensuite d'effectuer des dépenses de façon anonyme auprès du fournisseur de service. Cependant, cette solution n'atteint pas l'ensemble des propriétés que nous souhaitons puisque toutes les transactions sont centralisées au niveau du tiers de confiance. Il connaît l'identité du client, l'identité du fournisseur de service, ainsi que les produits achetés.

Enfin, la solution SET [Mas96] considère le paiement électronique anonyme. Dans ce cas, le consommateur utilise un pseudonyme pour commander des services auprès d'un marchand. Le marchand contacte ensuite sa banque qui est capable, avec l'aide de la banque du consommateur, de retrouver l'identité de l'utilisateur et d'exécuter le paiement. Bien que permettant une forme

d'anonymat des utilisateurs, cette solution n'atteint pas l'anonymat total que nous souhaitons ici. En effet, un consommateur utilise le même pseudonyme pour tous ses achats : il n'est donc pas identifiable mais peut être tracé.

9.1.2 Définition

Définition 69 (Facturation anonyme).
Un schéma de facturation anonyme, noté $\mathcal{FA}$, est tout d'abord constitué d'une phase d'initialisation du système et de mise en place des acteurs.

- Init permet d'initialiser tous les paramètres du schéma. Il permet, d'autre part, aux fournisseurs d'obtenir leurs paires de clés. Enfin, nous considérons que la mise en place du catalogue de services $\mathsf{cs} = \{(i, \alpha_i)\}_{i\in[1,q]}$ se fait durant cette phase et que la description de celui-ci est inclue dans la clé publique du schéma gpk.

$$((\mathsf{fssk}, \mathsf{fspk}), \mathsf{ffsk}, \mathsf{ggsk}, \mathsf{gpk}) \leftarrow \textsc{Init}(1^\lambda)$$

- uInit est la phase de mise en place des utilisateurs. Durant cette phase, chaque utilisateur, éventuellement muni d'une paire de clés "personnelle", effectue les préparatifs nécessaires au schéma, ce qui comprend une phase d'enregistrement interactive auprès du fournisseur de facture. À la fin de cette étape, l'utilisateur est, lui-aussi, muni de sa paire de clés de consommateur.

$$(\mathsf{u} : (\mathsf{usk}_i, \mathsf{cert}_i); \mathsf{GG} : \mathsf{reg}) \leftarrow \textsc{uInit}(\mathsf{u} : (\mathsf{psk}, \mathsf{ppk}), \mathsf{gpk}; \mathsf{GG} : \mathsf{ggsk}, \mathsf{gpk})$$

Les trois étapes clés de la vie d'une instance de facturation anonyme sont ensuite représentées par les requêtes suivantes.

- RequeteService permet à un utilisateur enregistré u de générer une requête pour un service s à destination du fournisseur FS.

$$\mathsf{req_s} \leftarrow \textsc{RequeteService}(\mathsf{usk}_i, \mathsf{cer}_i, \mathsf{fspk}, \mathsf{gpk}, s)$$

- RequeteFacture permet au fournisseur de services de produire une requête, à destination du fournisseur de facture FF, sollicitant la génération de la facture qui correspond à l'achat d'un service s ayant pour code de facturation α_s.

$$\mathsf{req_f} \leftarrow \textsc{RequeteFacture}(\mathsf{req_s}, \mathsf{gpk}, \mathsf{fssk})$$

- RequetePaiement est utilisé par le fournisseur de facture FF pour retrouver l'utilisateur u concerné par une requête de facture et générer la requête de paiement correspondante.

$$\mathsf{req_p} \leftarrow \textsc{RequetePaiement}(\mathsf{req_f}, \mathsf{ffsk}, \mathsf{gpk}, \mathsf{fspk}, \mathsf{reg})$$

*

Un tel schéma est considéré comme sûr s'il assure les propriétés de sécurité suivantes.

Anonymat. Le fournisseur de services FS ne doit pas pouvoir retrouver quel utilisateur lui achète un service.
Ainsi, dans l'expérience correspondante, l'adversaire $\mathcal{A}$ peut corrompre le fournisseur de services et former une coalition en corrompant des utilisateurs. Il choisit ensuite deux utilisateurs honnêtes u_0 et u_1 ainsi qu'un service s. Le challengeur choisit aléatoirement un bit b puis retourne une requête pour le service s provenant de l'utilisateur u_b.
L'adversaire gagne l'expérience s'il retourne un bit b^* tel que $b^* = b$ avec une probabilité non-négligeablement supérieure à $1/2$.

Confidentialité des services. Le fournisseur de facture FF ne doit pas pouvoir apprendre quel service l'utilisateur a acheté parmi deux services ayant le même code de facturation. Dans l'expérience correspondante, l'adversaire $\mathcal{A}$ peut corrompre le fournisseur de facture ainsi qu'autant d'utilisateurs qu'il le souhaite, à condition qu'il y ait toujours au moins un utilisateur honnête dans le système. L'adversaire choisit deux services s_0 et s_1 qui ont le même code de facturation, $\alpha_0 = \alpha_1$, ainsi qu'un utilisateur honnête u. Le challenger choisit aléatoirement un bit b et retourne la requête de l'utilisateur u pour le service s_b.
L'adversaire gagne l'expérience s'il retourne un bit b^* tel que $b^* = b$ avec une probabilité non-négligeablement supérieure à $1/2$.

Non-Diffamation. Aucune coalition d'utilisateurs, même avec l'aide du fournisseur de service, ne peut produire une requête de facturation valide telle que le fournisseur de facture considère qu'elle concerne un utilisateur honnête qui ne fait pas parti de la coalition.
Dans l'expérience correspondante, l'adversaire $\mathcal{A}$ peut corrompre le fournisseur de services ainsi qu'autant d'utilisateurs qu'il le souhaite à condition qu'il y ait toujours au moins un utilisateur honnête dans le système. L'adversaire retourne une requête pour un utilisateur honnête u et un service s.
L'adversaire gagne s'il n'a jamais demandé de requête à l'utilisateur u pour le service s et que la requête retournée est une requête valide concernant l'achat du service s par utilisateur u.

Non Répudiation. Aucune coalition d'utilisateurs, même en connaissant les clés du fournisseur de facture, n'est capable ni de construire une requête de facturation valide du fournisseur de services FS, ni de produire une requête de service ou de facturation telle que le fournisseur de facture ne retrouve pas l'utilisateur concerné, ou ne soit pas capable de le prouver.
Dans l'expérience correspondante, l'adversaire $\mathcal{A}$ peut obtenir la clé secrète du fournisseur de facture FF et corrompre tous les consommateurs. L'adversaire retourne, à la fin de l'expérience, soit une requête de facturation différente de toutes celles qu'il a pu obtenir par l'oracle personnifiant le fournisseur de facture, soit une requête de service.
L'adversaire gagne l'expérience si la requête est une requête de facturation valide ou si la requête (de facturation ou de service) est telle que le fournisseur de facture FF est incapable de retrouver ou de prouver quel utilisateur elle concerne.

9.2 Vers une première solution de facturation anonyme

Je vais à présent décrire, étape par étape, comment construire une première solution répondant à l'ensemble de ces propriétés de sécurité.

9.2.1 Obtenir l'anonymat de l'utilisateur.

Nous souhaitons que l'utilisateur soit anonyme et non-traçable vis-à-vis du fournisseur de services FS. Pour cela, nous pouvons utiliser un système d'authentification à base de signatures de groupe (voir Chapitre 2). Nous avons vu au cours de ce mémoire que ce type de signatures permet à un utilisateur d'être anonyme au sein d'un groupe, tout en permettant à une autorité d'ouverture de retrouver l'auteur d'une signature en cas de problème.

Dans notre contexte, l'utilisateur peut signer les services voulus. Le fournisseur de service, agissant comme un vérifieur, n'est pas capable de retrouver l'auteur. Le fournisseur de facture, jouant le rôle de l'autorité d'ouverture, peut de son côté retrouver l'identité de l'utilisateur.

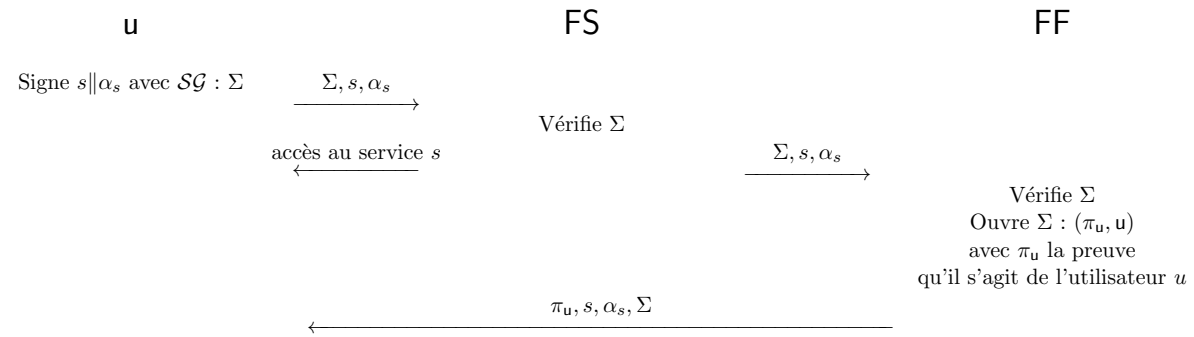

FIGURE 9.1 – Solution 1 : obtenir l' anonymat de l'utilisateur

Cette solution n'atteint cependant pas toutes les propriétés. En effet, la signature de groupe doit pouvoir être vérifiée par le fournisseur de service. Ceci implique qu'il connait les services souscris. Cette solution ne respecte donc pas la confidentialité des services. De plus, il est possible pour un consommateur corrompu de gagner l'expérience contre la non-répudiation en retournant simplement une requête de service.

9.2.2 Obtenir la confidentialité des services.

Une solution pour obtenir la propriété de Confidentialité des services est que l'utilisateur chiffre les services demandés de telle manière que le fournisseur de services puisse déchiffrer mais que le fournisseur de facture en soit incapable. Un chiffrement à clé secrète implique le partage de la clé de chiffrement/déchiffrement. Il y a alors deux configurations possibles. La première implique que tous les utilisateurs partagent la même clé avec le fournisseur de service. Et il suffit alors au fournisseur de facture de corrompre un utilisateur pour pouvoir déchiffrer tous les messages. La seconde configuration suppose que chaque paire utilisateur-fournisseur de services a une clé commune. Dans ce cas, l'utilisateur n'est plus anonyme car il est reconnaissable par la clé qu'il utilise. Nous utilisons donc un chiffrement à clé publique. L'utilisateur chiffre le (ou les) services souhaités avec la clé publique du fournisseur de service. Puis signe, en tant que membre du groupe, le chiffré obtenu et le code de facturation.

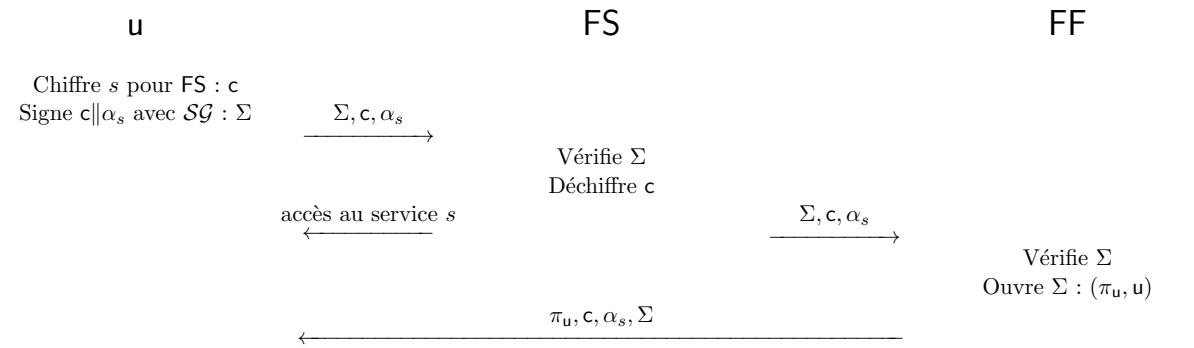

FIGURE 9.2 – Solution 2 : obtenir la confidentialité des services

Cependant, en pratique, les utilisateurs n'ont pas nécessairement accès aux codes de facturation. De plus, une requête de service et une requête de facturation étant identiques, la propriété de non-répudiation n'est pas atteinte. Une requête de service et une requête de facturation étant identique, un utilisateur corrompu peut gagner l'expérience contre la non-répudiation en retournant simplement une requête de service.

9.2.3 Obtenir la non répudiation

Afin d'obtenir cette propriété, nous impliquons le fournisseur de services dans la génération des requêtes de facturation. Une solution est qu'il signe la requête de service émise par l'utilisateur ainsi que le code de facturation qui y correspond. Cette solution a pour avantage qu'elle ne suppose pas que l'utilisateur connaisse les codes de facturation.

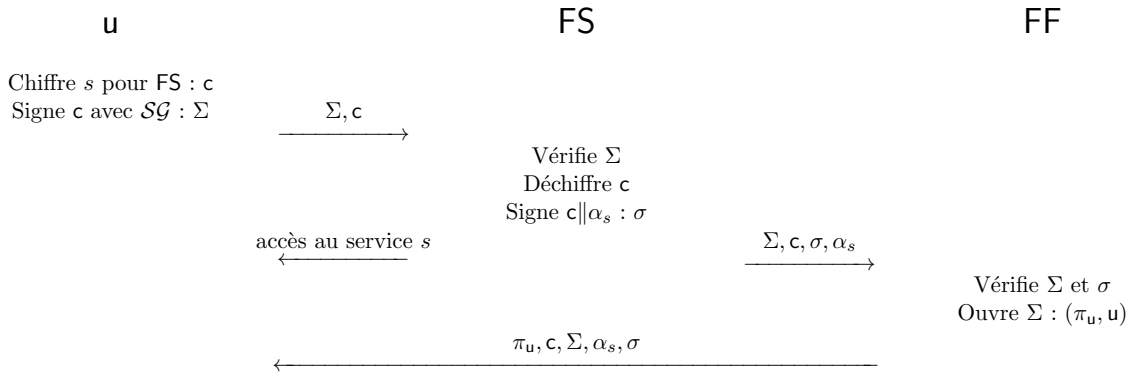

FIGURE 9.3 – Solution 3 : obtenir la non répudiation

Nous avons ici une première solution respectant toutes les propriétés de sécurité. Il est possible d'obtenir une solution équivalente mais plus efficace en utilisant une brique moins classique : un schéma de signature de groupe caméléon (voir Chapitre 8). Je vais à présent décrire cette solution en détail.

9.3 Une nouvelle solution de facturation anonyme plus efficace

Nous proposons à présent une seconde solution basée sur un schéma de signature de groupe caméléon (voir Chapitre 8). Les utilisateurs sont les membres du groupe et peuvent produire des signatures de groupe caméléon originales. Le fournisseur de services est un délégué et peut ainsi modifier le message afin d'y inclure le code de facturation correspondant au(x) service(s). Enfin, le fournisseur de facture joue le rôle de l'autorité d'ouverture et peut ainsi retrouver l'utilisateur à l'origine de la signature.

9.3.1 Le schéma

Dans ce contexte, il est utile de pouvoir publiquement distinguer un couple message-signature original, qui correspond à une requête de service, d'un couple modifié, qui correspond à une requête de facture. Nous utilisons donc un schéma non-transparent. Il y a ici deux possibilités. La première consiste en l'utilisation d'un schéma de signature de groupe non transparent avec

intimité qui empêche que le message original ne puisse être retrouvé à partir d'un couple modifié. Ce qui dans notre cas correspond à garantir que le fournisseur de facture est incapable de retrouver le service. La seconde possibilité, que nous développons ici, propose de chiffrer le service s, pour le fournisseur de service, avant de le signer avec un schéma de signature de groupe non transparent sans intimité. Dans notre cas où seule une partie du message ne doit pas être retrouvé, cette solution est plus rapide. De plus, elle ne suppose pas que l'utilisateur et le fournisseur de services partagent un canal secret pour les requêtes de services. Nous utilisons donc le schéma de signature de groupe non-transparent sans intimité proposé au Chapitre 8 en Section 8.2.

Informellement, nous obtenons le schéma suivant.

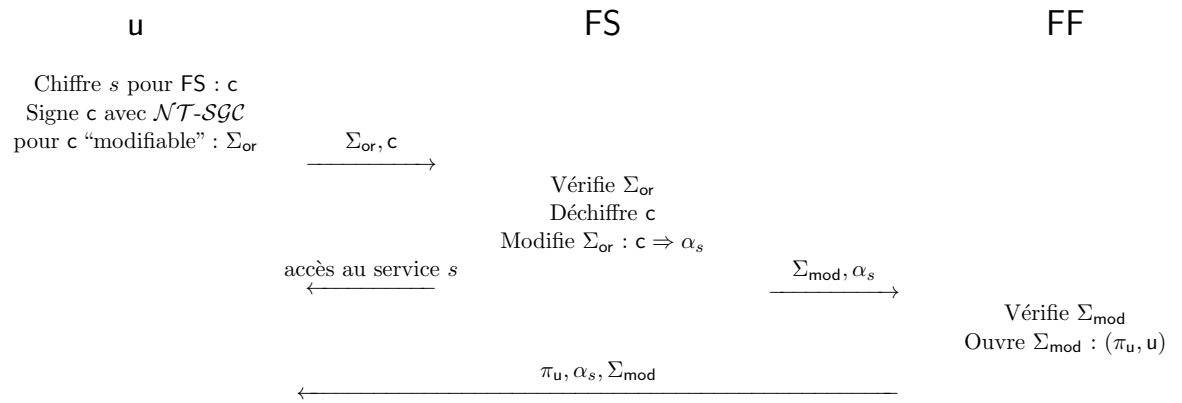

FIGURE 9.4 – Nouvelle solution de facturation anonyme

Plus formellement, notre schéma fonctionne de la façon suivante.

Construction 26 (Facturation anonyme).
Notre schéma de facturation anonyme, noté $\mathcal{FA}$, utilise un schéma de signature de groupe caméléon non-transparent $\mathcal{NT}$-$\mathcal{SGC}$ sûr dans le modèle décrit au Chapitre 8 ainsi qu'un schéma de chiffrement **IND − CPA** sûr (voir Chapitre 2).

- $\mathcal{FA}$.INIT(1^λ) Cet algorithme prend en entrée 1^λ avec λ un paramètre de sécurité et initialise le système. Il initialise le schéma de chiffrement en exécutant l'algorithme $\mathcal{Ch}$.INIT(1^λ) et obtient les paramètres $\mathcal{Ch}$.param. Il utilise ensuite $\mathcal{SGC}$.INIT(1^λ) qui retourne la clé secrète du gestionnaire du groupe ggsk, celle de l'autorité d'ouverture osk et enfin la clé publique du groupe gpk. Le rôle du gestionnaire de groupe peut être joué par le fournisseur de facture ou par une entité indépendante. Nous le considérons donc, ici, comme une entité à part. L'autorité d'ouverture correspond au fournisseur de facture, qui obtient donc la clé secrète ffsk = osk.
  Le fournisseur de services utilise l'algorithme $\mathcal{SGC}$.SANGÉNCLÉ($\mathcal{SGC}$.param) pour générer sa paire de clés de délégué dans le schéma de signature de groupe caméléon, que nous notons (dsk, dpk). Il utilise ensuite l'algorithme $\mathcal{Ch}$.GÉNCLÉ($\mathcal{Ch}$.param) et obtient une paire de clés de chiffrement (cpk, csk). Le fournisseur de services a donc la paire de clés (fssk = {dsk, csk}, fspk = {dpk, cpk}). Nous considérons que les paramètres du schéma de chiffrement sont inclus dans la clé publique cpk.
- $\mathcal{FA}$.uINIT(u_i : (psk, ppk), gpk; GG : ggsk, gpk) L'utilisateur u_i, qui est éventuellement muni d'une paire de clés personnelle (psk_i, ppk_i), se joint au groupe des signataires en utilisant le

protocole interactif $\mathcal{SGC}$.Rejoins(u_i : gpk, (psk_i, ppk_i); GG : gpk, ggsk) avec le gestionnaire de groupe. Il obtient ainsi une clé secrète d'utilisateur usk_i ainsi qu'un certificat d'appartenance au groupe cer_i. D'autre part, le gestionnaire de groupe ajoute une nouvelle entrée dans sa table de registre $\mathsf{reg}[i]$ qui correspond à l'utilisateur u_i. Notons, que la table de registre est accessible en lecture à l'autorité d'ouverture.

Le protocole de facturation en lui-même est composé des trois requêtes suivantes.

$\mathcal{FA}$.RequeteService($\mathsf{usk}_i, \mathsf{cer}_i, \mathsf{fspk}, \mathsf{gpk}, s$). L'utilisateur chiffre le service s qu'il souhaite acquérir en exécutant $\mathcal{C}h$.Chiffre(s, cpk), il obtient ainsi le chiffré c.

Il génère ensuite sa requête $\mathsf{m} = \mathsf{m}_g \| \mathsf{c} \| \mathsf{m}_d$, les parties m_g et m_d étant optionnelles. Il construit la variable ADM. Cette variable indique la taille de chaque partie et précise que la partie comprenant c est modifiable par le délégué.

L'utilisateur exécute ensuite $\mathcal{SGC}$.Signe($\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{dpk}, \mathsf{gpk}, \mathsf{ADM}$) qui retourne la signature Σ_{or}.

Enfin, la requête correspond à $\mathsf{req_s} = (\mathsf{m}, \Sigma_{\mathsf{or}})$.

$\mathcal{FA}$.RequeteFacture($\mathsf{req_s} = (\mathsf{m}, \Sigma_{\mathsf{or}}), \mathsf{gpk}, \mathsf{fssk}$). Le fournisseur de services vérifie si Σ est une signature valide du message m. Si $\mathcal{SGC}$.Vérifie($\mathsf{m}, \Sigma_{\mathsf{or}}, \mathsf{gpk}, \mathsf{dpk}$) = faux, il s'interrompt en retournant $\perp$. Sinon, il décompose le message m en $\mathsf{m}_g \| \mathsf{c} \| \mathsf{m}_d$ afin d'obtenir c. Il déchiffre ensuite c en utilisant l'algorithme $\mathcal{C}h$.Déchiffre(c, csk) et obtient ainsi le service s. Il répond à la requête de l'utilisateur en délivrant le service puis définit la variable de modification MOD en indiquant qu'il souhaite remplacer c par le code de facturation α_s qui correspond au service s.Enfin, il exécute l'algorithme $\mathcal{SGC}$.Modifie($\mathsf{m}, \Sigma_{\mathsf{or}}, \mathsf{gpk}, \mathsf{dsk}, \mathsf{MOD}$) et obtient ainsi la signature modifiée Σ_{mod} sur le message $\mathsf{m_{mod}} = \mathsf{m}_g \| \alpha_s \| \mathsf{m}_d$.

La requête de facturation est $\mathsf{req_f} = (\mathsf{m_{mod}}, \Sigma_{\mathsf{mod}})$.

$\mathcal{FA}$.RequetePaiement($\mathsf{req_f} = (\mathsf{m_{mod}}, \sigma'), \mathsf{ffsk}, \mathsf{gpk}, \mathsf{fspk}, \mathsf{reg}$). Le fournisseur de facture vérifie la signature Σ_{mod}. Si $\mathcal{SGC}$.Vérifie($\mathsf{m_{mod}}, \Sigma_{\mathsf{mod}}, \mathsf{gpk}, \mathsf{dpk}$) retourne faux, il s'arrête en retournant $\perp$. Sinon, il utilise l'algorithme $\mathcal{SGC}$.Ouvre($\mathsf{m_{mod}}, \Sigma_{\mathsf{mod}}, \mathsf{osk}, \mathsf{gpk}, \mathsf{dpk}$) afin de retrouver l'utilisateur à l'origine de la signature, il obtient ainsi $(\mathsf{u}, \pi_{\mathsf{u}})$.

La requête de paiement est $\mathsf{req_p} = (\pi_{\mathsf{u}}, \mathsf{m_{mod}}, \Sigma_{\mathsf{mod}})$. ⟲

9.3.2 Sécurité du schéma

Théorème 12. *Sous l'hypothèse que le schéma de signature de groupe caméléon non transparent et sans intimité $\mathcal{SGC}$ est sûr dans le modèle défini au Chapitre 8 et que le schéma de chiffrement $\mathcal{C}h$ est **IND − CPA**, le schéma de facturation est sûr.* ♢

Principe de la démonstration. Prouver que notre schéma est sûr revient à montrer qu'il respecte les propriétés d'**Anonymat**, de **confidentialité des services**, de **non-diffamation** et de **non répudiation**.

Anonymat. Dans cette expérience, l'adversaire peut corrompre le fournisseur de service.

L'adversaire gagne, s'il retourne l'identité du consommateur ayant effectué la requête de challenge avec une probabilité non négligemment supérieure à 1/2. Une telle requête est une signature de groupe caméléon originale, un adversaire contre cette propriété peut donc être utilisé, de façon évidente, pour casser la propriété d'**Anonymat** de la signature $\mathcal{SGC}$. □

Confidentialité des services. Pour attaquer cette propriété, l'attaquant peut corrompre le fournisseur de service.

Pour gagner, l'adversaire doit pouvoir distinguer quel service a été demandé par le fournisseur de service, parmi deux de son choix, avec une probabilité non négligemment su-

périeure à 1/2. Or les services n'apparaissent dans les requêtes que chiffrés. Ainsi, un tel adversaire peut être utilisé contre la propriété d'indistinguabilité du schéma de chiffrement **IND − CPA**. □

Non-diffamation. Dans l'expérience correspondante, l'attaquant peut corrompre le fournisseur de service.

Les requêtes req_s et req_f sont des signatures de groupe caméléons (originale pour une requête de service et modifiée pour une requête de paiement). Donc un adversaire capable de construire une requête telle que le fournisseur de facture FF prouve qu'elle concerne un utilisateur honnête est capable de générer une signature de groupe caméléon telle que l'autorité d'ouverture indique un signataire honnête. Un tel adversaire peut donc être utilisé pour construire un adversaire contre la **uProtection** de la signature $\mathcal{SGC}$. □

Non Répudiation. Dans l'expérience correspondante, l'attaquant accès à la clé secrète du fournisseur de facture et peut corrompre autant de consommateurs qu'il le souhaite.

Si un adversaire est capable de construire une requête de facturation valide différente de toutes celles auxquelles il a pu accéder par l'oracle simulant le fournisseur de service, alors cet adversaire est capable de créer une nouvelle signature modifiée différente de toutes celles obtenues via oracles. Un tel adversaire peut donc être utilisé pour construire un adversaire contre la propriété de **delProtection** de la signature $\mathcal{SGC}$. Pour rappel, le schéma $\mathcal{NT}$-$\mathcal{SGC}$ est non-transparent donc la distinction entre une signature originale est modifiée est publique, cela est donc suffisant pour distinguer publiquement une requête de service d'une requête de facturation.

L'adversaire peut, d'autre part, gagner en retournant une requête de facture (ou de service) valide telle que le fournisseur de facture FF est incapable de savoir de qui elle provient ou est incapable de le prouver. Une telle requête est une signature de groupe caméléon $\mathcal{NT}$-$\mathcal{SGC}$ et le fournisseur de facture agit en tant qu'autorité d'ouverture. En conséquence un tel adversaire peut être utilisé pour construire un adversaire contre l'**AOProtection** du schéma de signature $\mathcal{NT}$-$\mathcal{SGC}$. □

Conclusion

J'ai présenté dans ce chapitre deux solutions de facturation permettant au consommateur d'être anonyme vis-à-vis du fournisseur de services et telles que le fournisseur de facturation ne soit pas capable de connaitre les services commandés. La première solution n'utilise que des outils cryptographiques considérés aujourd'hui comme classique, tandis que la seconde solution, la plus efficace pour le fournisseur de service, est une application naturelle des signatures de groupe caméléon décrite au Chapitre 8. Cette solution, qui est chronologiquement à l'origine de mes travaux sur les signatures de groupe caméléons, a fait l'objet du brevet [CJM09] déposé avec S.Canard et E.Malville et a été soumise avec S.Canard comme application des signatures de groupe caméléons (voir Annexe B).

Nous allons à présent nous intéresser à une problématique connexe : l'abonnement anonyme.

Chapitre 10

Abonnement anonyme et profilage

Sommaire

De nos jours de plus en plus de services sont disponibles en ligne. La mise à disposition de ces services suit différents modes de fonctionnement : une inscription ou un paiement par utilisation, un carnet de tickets permettant un nombre d'utilisations donné du service ou encore un système d'abonnement (éventuellement limité dans le temps) qui permet aux utilisateurs d'accéder au service autant qu'il le souhaite. Nous avons vu au chapitre précédent une solution permettant de répondre au premier cas tout en respectant l'anonymat des utilisateurs. Le second cas correspond aux schémas étudiant les coupons anonymes. Plusieurs solutions ont été proposées pour répondre à cette problématique, notamment [CES+05, CGH06]. Dans le cas de l'abonnement, seule une première réponse a, à ma connaissance, été proposée par M.Blanton [Bla08]. Cette solution permet aux utilisateurs d'accéder aux services auxquels ils sont abonnés de manière anonyme et non traçable vis-à-vis du fournisseur de service, cependant la procédure d'abonnement en elle-même n'est pas anonyme. De plus ce schéma utilise l'évaluation de couplages de manière intensive, son efficacité est donc limitée par le coût élevé de cette opération.

Dans ce chapitre, je présenterai trois solutions brevetées [CJ09] et publiées [CJ10b] en collaboration avec S. Canard à la conférence Trustbus 2010. Pour ces solutions, nous considérons l'anonymat et la non-traçabilité des utilisateurs lors de l'utilisation de services comme un minimum. Nous nous intéressons ensuite à la phase d'abonnement. L'utilisation étant anonyme et

non-traçable, cette phase est la seule utilisable par les fournisseurs de service pour profiler leurs clients, c'est-à-dire analyser leurs consommateurs et déterminer un ensemble de caractéristiques communes dans l'objectif de proposer des services ou des promotions mieux ciblées. Nous allons voir au court de ce chapitre comment nos solutions permettent d'obtenir différent compromis entre profilage des utilisateurs et respect de leur vie privée, ce qui correspondra, au minimum, à une utilisation anonyme et non-traçable des services auxquels ils sont abonnés et, au mieux, à une utilisation *et* un abonnement anonyme et non-traçable.

Il est à noter que l'utilisation d'un schéma dans lequel l'utilisateur est anonyme, voir même intraçable, durant tous le processus (abonnement et utilisation) implique de choisir un moyen de paiement ayant les mêmes critères, par exemple un schéma de monnaie électronique de type "E-Cash" [CHL05, CDG+09]. Nous considérons dans ce chapitre qu'une telle solution est utilisée pour le paiement et nous nous concentrons uniquement sur les phases de souscription et d'utilisation de services.

10.1 Définition et modèle

Un schéma d'abonnement à un ensemble de services est composé d'un fournisseur de service FS, qui propose un ensemble de services, et d'un ensemble d'utilisateurs u souhaitant souscrire et utiliser ces services. Un utilisateur u est abonné à un service s'il possède un certificat d'abonnement à ce service. L'utilisateur est anonyme et intraçable lorsqu'il utilise un service. Le compromis permettant le profilage est introduit uniquement dans la phase de souscription.

10.1.1 Définition

Par abus de langage, nous parlerons dans ce chapitre du service s lorsqu'il sera question du service ayant pour identifiant unique s.

Définition 70 (Abonnement à un ensemble de services).
Un schéma d'abonnement à un ensemble de services, noté $\mathcal{AS}$, est composé des algorithmes suivants.

- $\mathcal{AS}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système $\mathcal{AS}.\mathsf{param}$. Nous considérerons dans la suite que λ est inclus dans les paramètres.

$$\mathcal{AS}.\mathsf{param} \leftarrow \mathcal{AS}.\textsc{Init}(1^\lambda)$$

- $\mathcal{AS}$.FSInit est un algorithme, prenant $\mathcal{AS}.\mathsf{param}$ en entrée, qui permet au fournisseur de service FS de générer une paire de clés de fournisseur de service $(\mathsf{fssk}, \mathsf{fspk})$ ainsi qu'un ensemble S de f identifiants de services $s_1, \ldots, s_f$ représentant tous les services du catalogue du fournisseur de service FS. Nous considérerons dans la suite que $\mathcal{AS}.\mathsf{param}$ est inclus dans la clé publique fspk du fournisseur de service.

$$((\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}) \leftarrow \mathcal{AS}.\mathsf{FS}\textsc{Init}(\mathcal{AS}.\mathsf{param})$$

- $\mathcal{AS}$.uInit permet à chaque utilisateur u de générer une paire de clé personnelle $(\mathsf{upk}, \mathsf{usk})$ en fonction des paramètres du système $\mathcal{AS}.\mathsf{param}$.

$$(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathcal{AS}.\mathsf{u}\textsc{Init}(\mathcal{AS}.\mathsf{param})$$

- $\mathcal{AS}$.Souscris est un protocole entre un utilisateur u et le fournisseur de service FS. L'utilisateur prend en entrées sa paire de clés $(\mathsf{usk}, \mathsf{upk})$, l'ensemble des services auxquels il

souhaite s'abonner $\mathsf{S_u} \subset \mathsf{S}$ et la clé publique du fournisseur de service fspk. Tandis que le fournisseur de service prend comme entrées sa paire de clés $(\mathsf{fssk}, \mathsf{fspk})$, et S. À la fin du protocole, l'utilisateur u obtient un certificat d'abonnement cert aux services de $\mathsf{S_u}$ tandis que le fournisseur enregistre la trace du protocole dans une base de données notée Tr.

$$(\mathsf{u} : \mathsf{cert}; \mathsf{FS} : \mathsf{Tr}) \leftarrow \mathcal{AS}.\textsc{Souscris}(\mathsf{u} : (\mathsf{usk}, \mathsf{upk}), \mathsf{S_u}, \mathsf{fspk}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S})$$

– $\mathcal{AS}.\textsc{AjouteAbo}$ est un protocole entre un utilisateur u et le fournisseur de service FS permettant à un utilisateur d'ajouter des services à son certificat d'abonnement. L'utilisateur prend en entrées sa paire de clés $(\mathsf{usk}, \mathsf{upk})$, l'ensemble des services auxquels il souhaite souscrire $\tilde{\mathsf{S}}_\mathsf{u} \subset \mathsf{S}$, la clé publique du fournisseur de service fspk, et le certificat cert correspondant aux services $\mathsf{S_u}$ auxquels il est déjà abonné. Le fournisseur de service FS prend comme entrées sa paire de clés $(\mathsf{fssk}, \mathsf{fspk})$, S l'ensemble des services de son catalogue et la base de données Tr des traces de protocoles qu'il a effectué jusque là. A la fin du protocole, l'utilisateur obtient un nouveau certificat d'abonnement $\tilde{\mathsf{cert}}$ correspondant à l'ensemble des services auxquels il est à présent abonné $\mathsf{S_u^*} = \tilde{\mathsf{S}}_\mathsf{u} \cup \mathsf{S_u}$. Le fournisseur enregistre la trace du protocole dans la base de données Tr.

$$(\mathsf{u} : \tilde{\mathsf{cert}}; \mathsf{FS} : \mathsf{Tr}) \leftarrow \mathcal{AS}.\textsc{Souscris}(\mathsf{u} : (\mathsf{usk}, \mathsf{upk}), \tilde{\mathsf{S}}_\mathsf{u}, \mathsf{fspk}, \mathsf{cert}, \mathsf{S_u}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}, \mathsf{Tr})$$

– $\mathcal{AS}.\textsc{Utilise}$ est un protocole entre un utilisateur u et un fournisseur de service FS. u prend en entrées son certificat d'abonnement cert, sa paire de clés $(\mathsf{usk}, \mathsf{upk})$, la clé publique du fournisseur de service fspk et un service $s \in \mathsf{S_u}$ auquel il souhaite accéder. Tandis que le fournisseur de service utilise sa paire de clés $(\mathsf{fssk}, \mathsf{fspk})$ et l'ensemble des services de son catalogue S. Ce protocole retourne 1 si l'utilisateur a le droit d'accéder au service s et 0 sinon.

$$\{0,1\} \leftarrow \mathcal{AS}.\textsc{Souscris}(\mathsf{u} : \mathsf{cert}, (\mathsf{usk}, \mathsf{upk}), \mathsf{fspk}, s; \quad \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S})$$

*

10.1.2 Propriétés de sécurité

Il y a principalement trois propriétés de sécurité que l'on souhaite obtenir dans ce type de schéma. Nous utilisons les définitions de [Bla08] qui se présentent comme suit.

– **Consistance**. Tout abonné doit être capable, grâce au protocole $\mathcal{AS}.\textsc{Utilise}$ avec FS, d'utiliser les services auxquels il s'est abonné en utilisant les protocoles $\mathcal{AS}.\textsc{Souscris}$ ou $\mathcal{AS}.\textsc{AjouteAbo}$.
– **validité**. Même une coalition d'abonnés doit être incapable d'accéder à des services auxquels ils n'ont pas souscris. Dans l'expérience correspondante, l'adversaire doit faire en sorte que le protocole $\mathcal{AS}.\textsc{Utilise}$ retourne 1 sur un service s_i auquel aucun utilisateur qu'il représente n'a souscrit auprès du fournisseur de service par les protocoles $\mathcal{AS}.\textsc{Souscris}$ ou $\mathcal{AS}.\textsc{AjouteAbo}$.
– **Anonymat** : Même le fournisseur de service FS doit être incapable de distinguer si deux exécutions du protocole $\mathcal{AS}.\textsc{Utilise}$ ont été effectuées avec le même abonné. Plus formellement, il doit être impossible pour un adversaire, pouvant jouer le rôle du fournisseur de service, de décider entre deux utilisateurs u_0, u_1 qu'il choisit lequel participe à un protocole $\mathcal{AS}.\textsc{Utilise}$ sur un service s donné.

De plus, [Bla08] définit la propriété d'efficacité suivante.

– **Compacité** : La taille du certificat d'abonnement cert ne doit pas dépendre du nombre de services auxquels l'utilisateur est abonné.

L'objectif de nos solutions est de définir différents compromis entre profilage et respect de la vie privée, nous déterminons donc à présent quelques repères pour chacun de ces deux critères.

Vie Privée.

En termes de respect de la vie privée, toutes nos solutions, ainsi que la solution de [Bla08], permettent l'anonymat et l'intraçabilité de l'utilisateur durant l'utilisation de ses services. Nous considérons ceci comme acquis et différentions trois niveaux considérant la prise en compte de ces principes lors des protocoles $\mathcal{AS}$.Souscris et $\mathcal{AS}$.AjouteAbo.

– **[U0. Pas d'anonymat]** L'utilisateur est reconnu par le fournisseur de service lors des protocoles $\mathcal{AS}$.Souscris et $\mathcal{AS}$.AjouteAbo.
– **[U1. Anonyme mais traçable]** L'utilisateur est anonyme vis-à-vis du fournisseur de service. Cependant, FS est capable de reconnaitre si un protocole $\mathcal{AS}$.Souscris et un $\mathcal{AS}$.AjouteAbo (ou deux $\mathcal{AS}$.AjouteAbo) sont effectués avec le même utilisateur.
– **[U2. Anonyme et intraçable]** L'utilisateur est anonyme et intraçable par le fournisseur de service durant les protocoles $\mathcal{AS}$.Souscris et $\mathcal{AS}$.AjouteAbo.

Profilage.

En ce qui concerne le profilage des utilisateurs nous définissons les trois niveaux suivants.

– **[P0. Aucun Profilage]** Le fournisseur de service est incapable de distinguer si deux services ont été souscris par le même utilisateur.
– **[P1. Profilage par abonnement]** Le fournisseur de service connait l'ensemble des services souscris par un utilisateur lors d'une instance du protocole $\mathcal{AS}$.Souscris ou $\mathcal{AS}$.AjouteAbo.
– **[P2. Profilage total]** Le fournisseur de service connait tous les services souscris par un même utilisateur.

10.1.3 Intuition de nos schémas

Dans les solutions que je présenterai dans ce chapitre, nous utiliserons de manière intensive le concept de signature "Camenisch-Lysyanskaya" [CL02, CL04] qui a été décrit en détail au Chapitre 2. Pour rappel, ces schémas de signatures, notés $\mathcal{CL}$, possèdent les trois propriétés suivantes.

– Il est possible de signer, avec l'algorithme de signature $\mathcal{CL}$.Signe, un message m décomposé en blocs $\mathsf{m} = \mathsf{m}_0 \| \cdots \| \mathsf{m}_\ell$.
– Il existe un algorithme, noté $\mathcal{CL}$.CSigne, permettant au signataire de signer un engagement de Pedersen sans pour autant connaître les valeurs engagées.
– Il existe un algorithme, noté $\mathcal{CL}$.Prouve, qui génère une preuve de connaissance $\mathcal{ZKPK}$ d'une signature sur un message sans révéler ni la signature, ni les valeurs signées.

De plus, chaque service proposé par le fournisseur de service FS est associé à un générateur h_i ainsi qu'à deux scalaires s_i et n_i. Le premier est l'identifiant du $i$ème service, le second indique la non-souscription au service s_i.

Durant la phase d'abonnement, correspondant au protocole $\mathcal{AS}$.Souscris, le fournisseur de service génère une signature Camenisch-Lysyanskaya sur les services auxquels l'utilisateur s'abonne $(s_1, \cdots, s_k)$, ceux qu'il n'a pas choisit $(n_{k+1}, \cdots, n_f)$, ainsi que sur une clé secrète usk

reliée à l'utilisateur. Cette dernière permet de s'assurer que seul l'abonné peut utiliser le certificat d'abonnement. Lors de la procédure $\mathcal{AS}$.AjouteAbo, le fournisseur de service ajoute les nouveaux services à la signature. Pour cela nous modifions le schéma de signature afin d'ajouter une nouvelle procédure permettant de mettre à jour une signature en ajoutant de nouveaux messages. Enfin, le protocole $\mathcal{AS}$.Utilise correspond à la génération, par l'utilisateur, d'une preuve de connaissance $\mathcal{ZKPK}$ d'une signature Camenisch-Lysyanskaya sur le service auquel il souhaite accéder et cela sans révéler la signature ni les autres services auxquels il est abonné.

10.2 Abonnement non-anonyme : U0-P2

10.2.1 Notre solution U0-P2

Notre premier schéma, instancié avec le schéma $\mathcal{CL}$ basé sur q-SDH, fonctionne de la manière suivante.

Construction 27 (Schéma U0-P2).

$\mathcal{AS}$.Init(1^λ) Cet algorithme consiste en l'exécution de l'algorithme $\mathcal{CL}$.Init(1^λ) afin de générer les paramètres du schéma $\mathcal{CL}$: $\mathcal{CL}$.param qui contiennent la description de l'environnement bilinéaire asymétrique $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ que nous utiliserons ici. Il choisit ensuite aléatoirement $g, h \in \mathbb{G}_1$. Nous notons $\mathcal{AS}.\mathsf{param} = (\mathcal{CL}.\mathsf{param}, g, h)$.

$\mathcal{AS}$.FSInit($\mathcal{AS}.\mathsf{param}$) Pour chacun des f services proposés par le fournisseur de service, on choisit un scalaire $s_i \in \mathbb{Z}_p$ qui représente l'état "abonné", un scalaire $n_i \in \mathbb{Z}_p$ qui représente l'état "non-abonné" et un élément de groupe $h_i \in \mathbb{G}_1$. Il est possible d'ajouter de nouveaux services au catalogue a posteriori, il suffira alors de générer ces trois variables pour chaque nouveau service.

Le fournisseur de service FS génère sa paire de clés en utilisant $\mathcal{CL}$.GénClé($\mathcal{CL}.\mathsf{param}$). Il obtient ainsi $\mathsf{fssk} = \gamma$ et $\mathsf{fspk} = (w = g_2^\gamma)$.

$\mathcal{AS}$.uInit($\mathcal{AS}.\mathsf{param}$) Chaque utilisateur génère une paire de clés long terme $(\mathsf{usk}, \mathsf{upk})$ telle que $\mathsf{upk} = g^{\mathsf{usk}}$.

$\mathcal{AS}$.Souscris($\mathsf{u} : (\mathsf{usk}, \mathsf{upk}), \mathsf{S_u}, \mathsf{fspk}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}$) L'utilisateur u souhaite s'abonner à $k \leq f$ services $s_{i_1}, \cdots, s_{i_k} \in \mathsf{S} = \{s_1, \cdots, s_f\}$. Nous notons $\mathsf{F} = [1, f]$, $\mathsf{I} = \{i_1, \cdots, i_k\} \subset \mathsf{F}$. L'utilisateur u et le fournisseur de service effectuent le protocole interactif qui correspond à la génération d'une signature Camenisch-Lysyanskaya sur : la clé secrète de l'utilisateur usk, $\{s_j\}_{j\in\mathsf{I}}$ l'ensemble des services auxquels il s'abonne et $\{n_j\}_{j\in\mathsf{F}\setminus\mathsf{I}}$ l'ensemble correspondant aux services que l'utilisateur n'a pas choisi. La clé secrète est engagée par l'utilisateur tandis que les valeurs de services s_j ou n_j sont engagées par le fournisseur de service.

Plus formellement nous avons les étapes suivantes.

1. L'utilisateur choisit aléatoirement $s' \in \mathbb{Z}_p$ puis engage sa clé secrète.

$$C' = h^{s'} h_0^{\mathsf{usk}}$$

2. Il génère ensuite $\pi_{C'}$ une preuve de connaissance $\mathcal{ZKPK}$ que u connait les valeurs engagées dans C' et que sa clé secrète usk reliée à la clé publique upk en fait partie.

$$\pi_{C'} = \textsc{Pok}(s', \mathsf{usk} : C' = h^{s'} h_0^{\mathsf{usk}} \wedge \mathsf{upk} = g^{\mathsf{usk}})$$

L'utilisateur envoie à FS l'engagement C', sa clé publique upk, la preuve $\pi_{C'}$ ainsi que la liste des services auxquels il souhaite s'abonner $s_{i_1}, \cdots, s_{i_k}$.

3. Pour chaque service, FS ajoute à l'engagement C' soit la valeur s_j si le service est souhaité soit n_j s'il n'a pas été choisi par l'utilisateur. Pour cela, il choisit aléatoirement $s'' \in \mathbb{Z}_p$ puis calcule l'engagement suivant.

$$C = C' h^{s''} \prod_{j \in \mathsf{I}} h_j^{s_j} \prod_{j \in \mathsf{F} \setminus \mathsf{I}} h_j^{n_j}$$

4. Enfin FS signe l'engagement C. Ainsi, il choisit aléatoirement $x \in \mathbb{Z}_p^*$ puis calcule

$$A = (g_1 C)^{\frac{1}{\gamma + x}}.$$

Il sauvegarde dans sa base de données Tr la trace du protocole, c'est-à-dire les engagements C, C', la preuve $\pi_{C'}$, l'ensemble des services souscris $\mathsf{S_u} = \{s_{i_1}, \cdots, s_{i_k}\}$ et la clé publique de l'utilisateur upk.

Enfin, il envoie à l'utilisateur son certificat d'abonnement $\mathsf{cert} = \sigma\big(= (A, x)\big)$ ainsi que la valeur s".

L'utilisateur obtient ainsi un certificat cert d'abonnement aux services souhaités $\mathsf{S_u}$. De plus, connaissant les valeurs s', s", $\{s_j\}_{i \in \mathsf{I}}$ et $\{n_j\}_{j \in \mathsf{F} \setminus \mathsf{I}}$, l'utilisateur peut reconstituer l'engagement C. Pour cela, il lui suffit de poser $s = s' + s$" et de calculer

$$C = h^s h_0^{\mathsf{usk}} \prod_{j \in \mathsf{I}} h_j^{s_j} \prod_{j \in \mathsf{F} \setminus \mathsf{I}} h_j^{n_j}.$$

$\mathcal{AS}$.AjouteAbo($\mathsf{u} : (\mathsf{usk}, \mathsf{upk}), \tilde{\mathsf{S}}_\mathsf{u}, \mathsf{fspk}, \mathsf{cert}, \mathsf{S_u}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}, \mathsf{Tr}$) Nous supposons que l'utilisateur u a déjà souscrit à k services. Il a donc un certificat d'abonnement cert correspondant à une signature $\sigma = (A, x)$ sur le message $(s, \mathsf{usk}, \{s_j\}_{j \in \mathsf{I}}, \{n_j\}_{j \in \mathsf{F} \setminus \mathsf{I}})$. L'utilisateur souhaite à présent ajouter à son abonnement l nouveaux services $s_{i_{k+1}}, \cdots, s_{i_\ell}$ avec $\ell = k + l$. Nous notons $\tilde{\mathsf{I}} = \{i_{k+1}, \cdots, i_\ell\} \cup \mathsf{I}$ le nouvel ensemble. L'utilisateur u et le fournisseur de service FS, effectue le protocole suivant.

1. L'utilisateur envoie au fournisseur de service sa clé publique upk, la variable C ainsi que la preuve de connaissance suivante.

$$\pi_{abo} = \textsc{Pok}(s, \mathsf{usk} : C / \prod_{j \in \mathsf{I}} h_j^{-s_j} \prod_{j \in \mathsf{F} \setminus \mathsf{I}} h_j^{-n_j} = h^s h_0^{\mathsf{usk}} \wedge \mathsf{upk} = g^{\mathsf{usk}})$$

2. Le fournisseur de service connaissant l'identité de l'utilisateur peut ajouter les nouveaux services à l'abonnement de l'utilisateur. Pour cela, il retrouve dans sa base de données Tr, la trace du protocole précédent $(C, s_{i_1}, \cdots, s_{i_k}, \mathsf{upk})$. Il vérifie la preuve π_{abo}. Si la preuve est invalide il retourne $\perp$ et s'arrête. Sinon, il choisit aléatoirement $\tilde{s}' \in \mathbb{Z}_p$ puis ajoute à l'engagement C les valeurs $s_{i_{k+1}}, \cdots, s_{i_\ell}$ de la façon suivante.

$$\tilde{C} = C h^{\tilde{s}'} \prod_{j \in \tilde{\mathsf{I}} \setminus \mathsf{I}} h_j^{s_j - n_j} = h^{\tilde{s}} h_0^{\mathsf{usk}} \prod_{j \in \tilde{\mathsf{I}}} h_j^{s_j} \prod_{j \in \mathsf{F} \setminus \tilde{\mathsf{I}}} h_j^{n_j}$$

3. Enfin, FS signe le nouvel engagement $\tilde{C}$. Il choisit aléatoirement $\tilde{x} \neq x$ dans $\mathbb{Z}_p$ puis calcule

$$\tilde{A} = (g_1 \tilde{C})^{\frac{1}{\gamma + \tilde{x}}}.$$

Il remplace dans sa base de données Tr les anciennes valeurs correspondant à l'utilisateur upk par : le nouvel engagement $\tilde{C}$, le nouvel ensemble de services $\mathsf{S_u} = \{s_{i_1}, \cdots, s_{i_\ell}\}$ et la clé publique de l'utilisateur upk.

Enfin, il envoie à l'utilisateur son nouveau certificat d'abonnement $\mathsf{cert} = \tilde{\sigma} = (\tilde{A}, \tilde{x})$ ainsi que la valeur $\tilde{s}'$.

L'utilisateur obtient ainsi un certificat cert d'abonnement mise à jour sur le nouvel ensemble de services $\mathsf{S_u}$. De plus, connaissant les valeurs $s, \tilde{s}', \{s_j\}_{i\in\tilde{\mathsf{I}}}$ et $\{n_j\}_{j\in\mathsf{F}\setminus\tilde{\mathsf{I}}}$, l'utilisateur peut reconstituer le nouvel engagement $\tilde{C}$. Pour cela, il lui suffit de poser $\tilde{s} = s + \tilde{s}'$ et de calculer

$$\tilde{C} = h^{\tilde{s}} h_0^{\mathsf{usk}} \prod_{j\in\tilde{\mathsf{I}}} h_j^{s_j} \prod_{j\in\mathsf{F}\setminus\tilde{\mathsf{I}}} h_j^{n_j}.$$

$\mathcal{AS}$.Utilise($\mathsf{u} : \mathsf{cert}, (\mathsf{usk}, \mathsf{upk}), \mathsf{fspk}, s; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}$) L'utilisateur u, abonné à l'ensemble des services $\mathsf{S_u} = \{s_{i_1}, \cdots, s_{i_\ell}\}$, souhaite utiliser le service $s = s_{i_u} \in \mathsf{S_u}$. Ce protocole est basé sur le fait que le certificat cert est une signature Camenisch-Lysyanskaya $\sigma = (A, x)$ sur le message (s, usk, $\{s_j\}_{j\in\mathsf{I}}$, $\{n_j\}_{j\in\mathsf{F}\setminus\mathsf{I}}$) avec $\mathsf{I} = \{i_1, \cdots, i_\ell\}$. Nous pouvons donc utiliser l'algorithme $\mathcal{CL}$.Prouve afin de prouver la connaissance d'une signature valide sur le message sans révéler ni la signature, ni le message à l'exception du bloc correspondant au service s_{i_u}.

Dans le cas d'une signature q-SDH, l'utilisateur calcule d'abord $C_1 = Ah^r$ et $C_2 = g^r h^u$ puis génère la preuve suivante.

$$\pi_{util} = \textsc{Pok}(\tilde{s}, \mathsf{usk}, \{s_j\}_{j\in\mathsf{I}\setminus\{i_u\}}, \{n_j\}_{j\in\mathsf{F}\setminus\tilde{\mathsf{I}}}, x, rx, r, s, sx : C_2 = g^r h^s \wedge 1 = C_2^x g^{-rx} h^{-sx} \wedge$$
$$e(g_1, g_2) e(h_j, g_2)^{s_{i_u}} / e(C_1, w) = e(C_1, g_2)^x e(h, g_2)^{-rx} e(h, w)^{-r} \prod_{j\in\tilde{\mathsf{I}}} e(h_j, g_2)^{-s_j} \prod_{j\in\mathsf{F}\setminus\tilde{\mathsf{I}}} e(h_j, g_2)^{-n_j})$$

Le fournisseur de service vérifie la validité de la preuve π_{util}. Si celle-ci est valide, il retourne 1 et l'utilisateur peut accéder au service, sinon il retourne 0. ↺

10.2.2 Sécurité du schéma

Nous avons vu au début de ce chapitre qu'un bon schéma d'abonnement, dans le modèle que nous considérons, devait suivre quatre propriétés. Nous allons à présent voir comment se place ce premier schéma face à ces quatre points.

Consistance : Un utilisateur ayant un certificat valide, et donc une signature Camenisch-Lysyanskaya valide, sera capable de produire la preuve du protocole $\mathcal{AS}$.Utilise.

validité : La signature Camenisch-Lysyanskaya étant **Infalsifiable** (dans notre cas sous l'hypothèse q-SDH) alors un adversaire sera incapable de générer une signature sur un nouveau message, et cela même en ayant accès au protocole de vérification et à un oracle de signature. De plus l'engagement choisit étant **résistant aux collisions**, il n'est pas possible d'y ajouter de nouvelles valeurs sans en modifier la valeur.

Anonymat : Cette propriété est obtenue par l'utilisation de preuve de connaissance $\mathcal{ZKPK}$ durant la procédure $\mathcal{AS}$.Utilise. Il est ainsi impossible, même pour le fournisseur de service, de relier une instance $\mathcal{AS}$.Souscris à une instance $\mathcal{AS}$.Utilise sans casser le schéma d'engagement ou la preuve de connaissance.

Compacité : La taille de notre certificat est celle d'une signature Camenisch-Lysyanskaya, c'est-à-dire un couple (A, x) comprenant un élément de groupe et un scalaire. Elle est donc bien indépendante du nombre de services souscris.

10.2.3 Schéma U0-P2 : Profilage et Vie Privée

Avec ce premier schéma, nous atteignons un premier niveau d'intraçabilité de l'utilisateur. Le fournisseur de service ignore l'identité de l'utilisateur qui accède à ses services et n'est pas capable de faire le lien entre une souscription et son utilisation. Avec un tel système, le fournisseur de service peut effectuer un profilage précis : il connait l'ensemble des services auxquels chacun de ses utilisateurs a souscrit. Il peut effectuer les statiques qu'il souhaite sur les habitudes de ses utilisateurs.

Cependant, un utilisateur pourrait souhaiter un plus au niveau de protection par rapport à son fournisseur de service. Par exemple en ne donnant pas son identité au fournisseur lorsqu'il souscrit à un abonnement. Nous allons voir comment nous pouvons mettre en place une solution respectant l'anonymat des utilisateurs tout en permettant un certain profilage de la part du fournisseur de service.

10.3 Abonnement anonyme : U1-P2

Dans le premier schéma que nous venons de voir, l'utilisateur prouve qu'il a le droit de souscrire à un service en utilisant une preuve de connaissance $\mathcal{ZKPK}$ d'une clé secrète correspondant à sa clé publique. Cette solution assure la non-répudiation de la part de l'utilisateur, elle l'empêche de revenir sur l'engagement qu'il a prit en souscrivant à un abonnement, cependant elle permet au fournisseur de service de relier la clé upk à l'identité de l'utilisateur, cette solution n'est donc pas anonyme.

Une solution pour obtenir l'anonymat des utilisateurs est de les cacher dans un groupe : le groupe des personnes étant autorisé à s'abonner aux services proposés par FS. De plus, nous avons besoin de nous assurer que les utilisateurs ne puissent pas répudier leurs demandes d'abonnement. Pour cela, il nous faut être capable de lever l'anonymat en cas de problème entre l'utilisateur et le fournisseur de service. Tout ceci est possible grâce aux signatures de groupe introduites au Chapitre 2. Cependant, nous avons besoin d'une propriété supplémentaire : la **Compacité**. Pour cela, le fournisseur de service doit pouvoir reconnaitre un utilisateur lors d'un abonnement à des services supplémentaires ($\mathcal{AS}$.AjouteAbo), afin de pouvoir agréger les nouveaux services sur l'ancien certificat. Nous utiliserons en conséquence une variante des signatures de groupes introduite au Chapitre 2 : les signatures de liste.

Afin de simplifier la lecture, je vais à présent décrire le schéma 2 par rapport au schéma 1, ce qui nous permettra de mieux visualiser les différences. Ainsi, je ne décrirai pas une seconde fois les procédures qui restent inchangées et ne donnerai que les modifications à effectuer et non une description complète de chaque algorithme ou protocole.

10.3.1 Notre solution U1-P2

Construction 28 (Schéma U1-P2).
Nous utilisons ici le schéma de signature Camenisch-Lysyanskaya, noté $\mathcal{CL}$, basé sur q-SDH ainsi qu'un schéma de signature de liste, noté $\mathcal{SL}$, basé sur celui-ci.

$\mathcal{AS}$.Init(1^λ) Comme dans le Schéma 1, on génère les paramètres de la signature Camenisch-Lysyanskaya $\mathcal{CL}.\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. On utilise ensuite ces paramètres pour le schéma de signature de liste $\mathcal{SL}$, et on génère ses clés grâce à $\mathcal{SL}$.GénClé($\mathcal{CL}.\mathsf{param}$). Le gestionnaire de liste, qui peut être le fournisseur de service puisque l'anonymat du schéma est aussi respecté vis-à-vis du gestionnaire de liste, obtient ainsi la clé secrète glsk. Le gestionnaire d'ouverture obtient la clé secrète $\mathsf{osk} = (\zeta_1, \zeta_2)$. La clé publique de la liste est $\mathsf{lpk} = (k, w, h, g)$.

Nous notons $\mathcal{AS}.\mathsf{param} = (\mathcal{CL}.\mathsf{param}, \mathsf{lpk})$.

$\mathcal{AS}.\mathsf{FSInit}(\mathcal{AS}.\mathsf{param})$ Cette procédure reste identique à celle du schéma U0-P2.

$\mathcal{AS}.\mathsf{uInit}(\mathcal{AS}.\mathsf{param})$ Chaque utilisateur u rejoint la liste des clients potentiels en utilisant le protocole $\mathcal{SL}.\textsc{Rejoins}$ avec le gestionnaire de groupe. Il obtient ainsi une clé secrète sk ainsi qu'un certificat d'appartenance à la liste reliée à cette clé $(A_l = (g_1 h_0^{\mathsf{sk}})^{\frac{1}{\mathsf{gisk}+x_l}}, x_l)$. La clé secrète de l'utilisateur est $\mathsf{usk} = (\mathsf{sk}, A_l, x_l)$, la clé publique correspondante étant la clé publique de la liste lpk.

$\mathcal{AS}.\textsc{Souscris}(\mathsf{u} : (\mathsf{usk}, \mathsf{lpk})\mathsf{S_u}, \mathsf{fspk}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S})$ Dans ce schéma, l'utilisateur ne prouve plus que C' est un engagement valide de sa clé secrète reliée à sa clé publique mais que c'est un engagement valide de sa clé sk reliée à un certificat d'appartenance à la liste valide (A_l, x_l). On utilise ainsi le même processus d'abonnement que dans le Schéma $U0-P2$ en remplaçant la preuve $\pi_{C'}$ par d'une part le chiffrement de A_l en utilisant le double chiffrement Elgamal, et d'autre part la génération d'une nouvelle preuve $\pi_{C'}$. Cette nouvelle preuve correspond à la partie à prouver de la signature de connaissance effectuée dans une signature de liste. Plus formellement, l'utilisateur choisit aléatoirement $\alpha, \beta \in \mathbb{Z}_p^*$ puis calcule $T_1 = k^\alpha$, $T_2 = A_l h^\alpha$, $T_3 = k^\beta$ et $T_4 = A_l g^\beta$. Il génère ensuite la preuve $\mathcal{ZKPK}$ suivante.

$$\pi_{C'} = \textsc{Pok}\big(s', \alpha, \beta, x_l, x_l\alpha, \mathsf{sk} : \quad C' = h^{s'} h_0^{\mathsf{sk}} \wedge T = h_{\mathsf{FS}}^{\mathsf{sk}} \wedge T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge e(T_2, g_2)^{x_l} e(h, w)^{-\alpha} e(h, g_2)^{-x_l\alpha} e(h_0, g_2)^{-\mathsf{sk}} = e(g_1, g_2)/e(T_2, w)\big)$$

Les autres étapes du protocole ne sont pas modifiées. L'utilisateur obtient ainsi, à la fin du protocole, l'engagement $C = h^s h_0^{\mathsf{sk}} \prod_{j\in\mathsf{I}} h_j^{s_j} \prod_{j\in\mathsf{F}\setminus\mathsf{I}} h_j^{n_j}$ ainsi que son certificat d'abonnement $\mathsf{cert} = \sigma$ qui est, comme dans le premier schéma, une signature $\sigma = (A, x)$ du message $(s, \mathsf{sk}, \{s_j\}_{j\in\mathsf{I}}, \{n_j\}_{j\in\mathsf{F}\setminus\mathsf{I}})$.

D'autre part, le fournisseur de service conserve dans sa base de données Tr la trace du protocole : la valeur T, l'ensemble des services souscris $\mathsf{S_u} = \{s_{i_1}, \cdots, s_{i_k}\}$ et la valeur C.

$\mathcal{AS}.\textsc{AjouteAbo}(\mathsf{u} : (\mathsf{usk}, \mathsf{lpk}), \tilde{\mathsf{S}}_\mathsf{u}, \mathsf{fspk}, \mathsf{cert}, \mathsf{S_u}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}, \mathsf{Tr})$ Ce protocole est modifié de manière similaire au protocole $\mathcal{AS}.\textsc{Souscris}$. On remplace ainsi la preuve que $\mathsf{upk} = g^{\mathsf{usk}}$ par la preuve que sk correspond au certificat (A_l, x_l) de la signature de liste. Ce qui correspond à la preuve suivante.

$$\textsc{Pok}\big(\alpha, \beta, x_l, x_l\alpha, \mathsf{sk} : \quad T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge e(T_2, g_2)^{x_l} e(h, w)^{-\alpha} e(h, g_2)^{-x_l\alpha} e(h_0, g_2)^{-\mathsf{sk}} = e(g_1, g_2)/e(T_2, w)\big)$$

Il doit d'autre part prouver qu'il connait la clé sk contenue dans C et que la valeur $T = h_{\mathsf{FS}}^{\mathsf{sk}}$ de la signature de liste est bien construite. La preuve de connaissance π_{abo} du protocole $\mathcal{AS}.\textsc{AjouteAbo}$ est donc finalement remplacée par la preuve suivante.

$$\pi_{abo} = \textsc{Pok}\big(s, \alpha, \beta, x_l, x_l\alpha, \mathsf{sk} : C/\prod_{j\in\mathsf{I}} h_j^{-s_j} \prod_{j\in\mathsf{F}\setminus\mathsf{I}} h_j^{-n_j} = h^s h_0^{\mathsf{sk}} \wedge T = h_{\mathsf{FS}}^{\mathsf{sk}} \wedge T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge e(T_2, g_2)^{x_l} e(h, w)^{-\alpha} e(h, g_2)^{-x_l\alpha} e(h_0, g_2)^{-\mathsf{sk}} = e(g_1, g_2)/e(T_2, w)\big)$$

Le fournisseur de service est capable de retrouver dans sa base de données Tr la trace du protocole précédent grâce à la valeur $T = h_{\mathsf{FS}}^{\mathsf{sk}}$. Il peut donc agréger les nouveaux abonnements à l'ancien certificat comme dans le schéma U0-P2.

$\mathcal{AS}$.UTILISE(u : cert, (usk, lpk), fspk, s; FS : (fssk, fspk), S) Ce protocole est identique à celui du premier schéma. En effet, l'utilisateur n'a pas à prouver le lien entre sk et son certificat d'appartenance à la liste.

10.3.2 Schéma U1-P2 : Profilage et Vie Privée

Dans ce second schéma, la vie privée de l'utilisateur est mieux prise en compte. En effet l'utilisateur est, ici, anonyme par rapport au fournisseur de service lors de l'utilisation mais aussi lors de la souscription. De plus, l'utilisation des signatures de liste permet au fournisseur de service de conserver la possibilité de relier un abonnement et un réabonnement. On peut ainsi conserver le même niveau de profilage que pour le schéma U0-P2.

Enfin, comme pour tous les schémas que je présente ici, l'utilisateur est anonyme et intraçable lors de l'utilisation de ses abonnements, c'est-à-dire lorsqu'il exécute le protocole $\mathcal{AS}$.UTILISE.

10.3.3 Le cas des signatures de groupe

Il est possible d'utiliser une signature de groupe plutôt qu'une signature de liste dans le schéma que nous venons de voir. Cependant les signatures de groupe étant non-traçables, il est alors nécessaire de procurer au fournisseur de service un moyen de relier un abonnement à un réabonnement. Une telle solution serait donc équivalente à la solution que nous venons de proposer.

Nous allons nous intéresser, à présent, à l'intraçabilité dans la phase d'abonnement.

10.4 Intraçabilité des services : U2-P1

Nous nous intéressons maintenant à la possibilité d'obtenir l'anonymat **et** l'intraçabilité de l'utilisateur durant la phase d'abonnement. Souhaitant conserver une possibilité de profilage des utilisateurs, nous proposons une solution permettant de connaitre l'ensemble des services auxquels un utilisateur souscrit lors d'un abonnement mais qui rend impossible de relier deux sessions d'abonnements entre elles.

Une solution serait de simplement remplacer la signature de liste dans le second schéma par une signature de groupe. Nous obtiendrions dans ce cas une solution intraçable et anonyme. Cependant, celle-ci ne serait pas compacte. En effet, le fournisseur de service serait dans l'incapacité d'ajouter de nouveaux services au certificat d'abonnement existant. L'utilisateur aurait alors un certificat par ensemble d'abonnement, ce que nous souhaitons éviter. Pour contourner ce problème, nous proposons de permettre à l'utilisateur de cacher l'engagement obtenu sur les services auxquels il est abonné afin que le fournisseur de service puisse y ajouter de nouveaux services sans être capable de faire le lien avec les abonnements précédents. L'utilisateur obtient ainsi un certificat d'abonnement compact.

Plus précisément, cela implique les changements suivants par rapport au schéma U1-P2.

10.4.1 Notre solution U2-P1

Construction 29 (Schéma U2-P1).
Nous utilisons ici le schéma de signature Camenisch-Lysyanskaya basé sur q-SDH, que nous notons $\mathcal{CL}$, ainsi que le schéma de signature de groupe [DP06] présentée au Chapitre 2.

$\mathcal{AS}$.Init(1^λ) Comme dans le Schéma U1-P2, l'algorithme génère les paramètres du schéma de signature Camenisch-Lysyanskaya $\mathcal{CL}.\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathsf{e})$. Ces même paramètres seront ensuite utilisés pour le schéma de signature de groupe $\mathcal{SG}$. L'algorithme génère ainsi les clés de la signature de groupe en utilisant l'algorithme $\mathcal{SG}$.GénClé($\mathcal{CL}.\mathsf{param}$). Le gestionnaire de groupe obtient ainsi la clé secrète $\mathsf{ggsk} = \gamma$, le gestionnaire d'ouverture obtient la clé secrète $\mathsf{osk} = (\zeta_1, \zeta_2)$. La clé publique du groupe est $\mathsf{gpk} = (k, w, h, g)$.

Nous notons $\mathcal{AS}.\mathsf{param} = (\mathcal{CL}.\mathsf{param}, \mathsf{gpk})$.

$\mathcal{AS}$.FSInit($\mathcal{AS}.\mathsf{param}$) Cette procédure reste identique à celles des schémas U0-P2 et U1-P2.

$\mathcal{AS}$.uInit($\mathcal{AS}.\mathsf{param}$) Chaque utilisateur rejoint le groupe des clients potentiels en utilisant le protocole $\mathcal{SG}$.Rejoins avec le gestionnaire de groupe et obtient ainsi une clé secrète d'utilisateur du groupe usk ainsi qu'un certificat d'appartenance au groupe $(A_j = (g_1 h_0^{\mathsf{sk}})^{\frac{1}{\gamma+x}}, x_b)$. Nous notons $\mathsf{usk} = (\mathsf{sk}, A_j, x_b)$ la clé secrète de l'utilisateur u.

$\mathcal{AS}$.Souscris($\mathsf{u} : (\mathsf{usk}, \mathsf{gpk})\mathsf{S_u}, \mathsf{fspk}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}$) Ce protocole est similaire au schéma U1-P2, excepté que nous utilisons ici une signature de groupe. La variable T n'a donc plus d'utilité et est donc retirée. Le chiffrement de A_g s'effectue de façon analogue au chiffrement de A_l dans le schéma U1-P2. L'utilisateur choisit donc aléatoirement $\alpha, \beta \in \mathbb{Z}_p^*$ puis calcule $T_1 = k^\alpha$, $T_2 = A_g h^\alpha$, $T_3 = k^\beta$ et $T_4 = A_g g^\beta$. Il prouve ensuite d'une part la connaissance de la clé secrète sk comprise dans l'engagement C' et d'autre part que cette clé est reliée au certificat A_g. Cela correspond à la preuve $\mathcal{ZKPK}$ suivante.

$$\begin{aligned}\pi_{C'} \;=\; & \textsc{Pok}\big(s', \alpha, \beta, x_g, x_g\alpha, \mathsf{usk} : C' = h^{s'} h_0^{\mathsf{sk}} \wedge \\ & T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge \\ & e(T_2, g_2)^{x_g} e(h, w)^{-\alpha} e(h, g_2)^{-x_g\alpha} e(h_0, g_2)^{-\mathsf{sk}} = e(g_1, g_2)/e(T_2, w)\big)\end{aligned}$$

L'utilisateur obtient ainsi, à la fin du protocole, son certificat d'abonnement cert qui est, comme dans les deux premiers schémas, une signature $\sigma = (A, x)$ du message $(s, \mathsf{sk}, \{s_j\}_{j\in\mathsf{I}}, \{n_j\}_{j\in\mathsf{F}\setminus\mathsf{I}})$.

$\mathcal{AS}$.AjouteAbo($\mathsf{u} : (\mathsf{usk}, \mathsf{gpk})\tilde{\mathsf{S}}_\mathsf{u}, \mathsf{fspk}, \mathsf{cert}, \mathsf{S_u}; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S}, \cdot$) Comme l'utilisateur n'est pas reconnu par le fournisseur de service, nous devons adapter ce protocole et, en particulier, la preuve π_{abo} qu'il a bien un certificat d'abonnement valide. L'utilisateur envoie toujours un engagement sur les services auxquels il a déjà souscris mais il doit maintenant attester que celui-ci est bien formé, sans révéler les services $s_{i_1}, \cdots, s_{i_k}$ auxquels il est déjà abonné. De plus, afin que le fournisseur de service ne puisse reconnaître l'engagement, l'utilisateur ne doit pas simplement utiliser l'engagement $C = h^s h_0^{\mathsf{sk}} \prod_{j\in\mathsf{I}} h_j^{s_j} \prod_{j\in\mathsf{F}\setminus\mathsf{I}} h_j^{n_j}$ obtenu durant son précédent abonnement.

Pour cela, il ajoute de l'aléa à l'engagement afin de le rendre méconnaissable. De manière plus formelle, l'utilisateur choisit aléatoirement $\hat{s} \in \mathbb{Z}_p^*$ et génère $\hat{C} = h^{\hat{s}} C$. Il devra donc prouver qu'il connait tous les éléments constituants $\hat{C}$ et que la clé sk correspond à un certificat d'appartenance au groupe (A_g, x_g) valide ce qui correspond à la sous-preuve suivante.

$$\begin{aligned}\textsc{Pok}(s, \hat{s}, \alpha, \beta, x_g, x_g\alpha, \mathsf{sk}, \mathsf{S_u}, \{n_j\}_{j\in\mathsf{F}\setminus\mathsf{I}} : \hat{C} = h^s h^{\hat{s}} h_0^{\mathsf{sk}} \textstyle\prod_{j\in\mathsf{I}} h_j^{s_j} \prod_{j\in\mathsf{F}\setminus\mathsf{I}} h_j^{n_j} \wedge \\ T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha/g^\beta \wedge \\ e(T_2, g_2)^x e(h, w)^{-\alpha} e(h, g_2)^{-x\alpha} e(h_0, g_2)^{-\mathsf{sk}} = e(g_1, g_2)/e(T_2, w)\end{aligned}$$

Il doit ensuite prouver qu'il connait un certificat d'abonnement valide sur les services contenus dans celui-ci, c'est à dire qu'il connait une signature Camenisch-Lysyanskaya (A, x) sur les éléments contenus dans C sans la révéler. Il choisit aléatoirement r et u dans $\mathbb{Z}_p$ puis calcule

$C_1 = Ah^r$ et $C_2 = g^r h^u$. Il pourra ainsi prouver la connaissance de A en ajoutant la sous-preuve suivante.

$$\textsc{Pok}(r, u, x, s, \hat{s}, \mathsf{sk}, \mathsf{S_u}, \{n_j\}_{j\in\mathsf{F}\setminus\mathsf{I}} : C_2 = g^r h^u \wedge 1 = C_2^x g^{-rx} h^{-ux} \wedge$$

$$(g_1, g_2)e(h, w)^r / e(C_1, w) = e(C_1, g_2)^x e(h, g_2)^{-rx} e(h, g_2)^s e(h_0, g_2)^{\mathsf{sk}} \textstyle\prod_{j\in\mathsf{I}} e(h_j, g_2)^{-s_j} \prod_{j\in\mathsf{F}\setminus\mathsf{I}} e(h_j, g_2)^{-n_j})$$

En combinant ces différentes sous preuves, nous obtenons la preuve de connaissance $\mathcal{ZKPK}$ suivante.

$$\pi_{abo} = \textsc{Pok}(r, u, x, s, \hat{s}, \alpha, \beta, x_g, x_g\alpha, \mathsf{sk}, \mathsf{S_u}, \{n_j\}_{j\in\mathsf{F}\setminus\mathsf{I}} : \hat{C} = h^s h^{\hat{s}} h_0^{\mathsf{sk}} \prod_{j\in\mathsf{I}} h_j^{s_j} \prod_{j\in\mathsf{F}\setminus\mathsf{I}} h_j^{n_j} \wedge$$

$$C_2 = g^r h^s \wedge 1 = C_2^x g^{-rx} h^{-sx} \wedge T_1 = k^\alpha \wedge T_3 = k^\beta \wedge T_2/T_4 = h^\alpha / g^\beta \wedge$$

$$e(T_2, g_2)^x e(h, w)^{-\alpha} e(h, g_2)^{-x\alpha} e(h_0, g_2)^{-\mathcal{SG}.\mathsf{usk}} = e(g_1, g_2)/e(T_2, w) \wedge$$

$$e(g_1, g_2)e(h, w)^r / e(C_1, w) = e(C_1, g_2)^x e(h, g_2)^{-rx} e(h, g_2)^s e(h_0, g_2)^{\mathsf{sk}} \textstyle\prod_{j\in\mathsf{I}} e(h_j, g_2)^{-s_j} \prod_{j\in\mathsf{F}\setminus\mathsf{I}} e(h_j, g_2)^{-n_j})$$

Le vérifieur s'assure de la validité de π_{abo}, il choisit ensuite aléatoirement $\tilde{s}' \in \mathbb{Z}_p$ et calcule $\tilde{C} = \hat{C} h^{\tilde{s}'} \prod_{j\in\tilde{\mathsf{I}}\setminus\mathsf{I}} h_j^{s_j - n_j}$. Il signe ensuite l'engagement $\tilde{C}$ avec $\mathcal{CL}.\textsc{Signe}$ et obtient $\tilde{\sigma} = (\tilde{A}, \tilde{x})$. Il peut enfin envoyer le nouveau certificat d'abonnement $\mathsf{cert} = \tilde{\sigma}$ qui constitue une signature sur le message $(s, \mathcal{SG}.\mathsf{usk}, \{s_j\}_{j\in\tilde{\mathsf{I}}}, \{n_j\}_{j\in\mathsf{F}\setminus\tilde{\mathsf{I}}})$.

$\mathcal{AS}.\textsc{Utilise}(\mathsf{u} : \mathsf{cert}, (\mathsf{usk}, \mathsf{upk}), \mathsf{fspk}, s; \mathsf{FS} : (\mathsf{fssk}, \mathsf{fspk}), \mathsf{S})$ Ce protocole est identique à ceux des schémas U0-P2 et U1-P2.

10.4.2 Schéma U2-P1 : Profilage et Vie Privée

Dans ce troisième schéma l'utilisateur est anonyme et intraçable (entre deux abonnements) vis-à-vis du fournisseur de service. On obtient ainsi une solution encore plus respectueuse de la vie privée, dans laquelle le fournisseur de service est capable d'effectuer un profilage : il peut étudier l'ensemble des services souscris par l'utilisateur lors d'un protocole $\mathcal{AS}.\textsc{Souscris}$ ou $\mathcal{AS}.\textsc{AjouteAbo}$, cependant il sera incapable de savoir si deux ensembles souscris lors de deux instances distinctes proviennent du même utilisateur.

Remarque. Il est encore possible de monter d'un cran dans le respect de la vie privée des utilisateurs en abandonnant la possibilité d'effectuer tout profilage. Pour cela il suffit d'utiliser le schéma U2-P1 en ne souscrivant aux services qu'un par un. De cette façon, la vie privée de l'utilisateur est parfaitement respectée mais aucun profilage n'est plus possible.

10.5 Comparaison des constructions

Nous allons à présent comparer les solutions que nous avons vu au cours de ce chapitre en termes de respect de la vie privée des utilisateurs, profilage et efficacité. Afin de mieux cerner le travail que j'ai effectué sur ces solutions avec S.Canard dans [CJ10b], j'ajouterai à cette comparaison la solution proposée par M.Blanton dans [Bla08].

Efficacité.

Nous considérons ici l'efficacité d'un point de vue utilisateur. En effet nous supposons que le fournisseur de service à accès à des ordinateurs ayant une forte puissance de calcul. Nous évaluons ici l'efficacité des différentes solutions d'un point de vue pratique. Nous utilisons ainsi les résultats d'optimisation d'implémentation de A.J.Devegili, M.Scott et R.Dahab [DSD07] sur les courbes Barreto-Naehrig pour les couplages ainsi que les résultats des tests de performance effectués par le réseau d'excellence européen en cryptographie ECRYPT II [ECR08] pour les autres opérations.

Nous utilisons pour les calculs de représentation de la forme $c = \prod_{i=1}^{n} g_i^{e_i}$ l'astuce proposée dans [Möl01] permettant de baisser sa complexité. Grâce à elle, une telle exponentiation ne correspond plus qu'à (environ) $\frac{2^{n+1}-1}{3\times 2^{n-1}}$ exponentiations modulaires.

Nous comparons les temps nécessaires à chaque schéma pour s'abonner à 3 services puis le temps nécessaire pour ajouter 4 services à l'abonnement et, enfin, le temps pour utiliser un service. Le Schéma 4 ayant un principe différent des autres (il implique de s'abonner aux services un par un), nous adaptons le nombre de services initialement souscris. Le $\mathcal{AS}$.Souscris correspondant ne permet de souscrire qu'à un seul service donc $k = 1$. Puis nous adaptons le nombre de services ajoutés a posteriori, pour obtenir un total de 7 services comme pour les autres schémas, nous posons donc $l = 6$.

Comparaison

En utilisant les remarques et les notations que nous venons de voir, nous pouvons établir le tableau de comparaison des schémas de la Figure 10.1. On remarque que les schémas $2, 3$ et 4 intégrant des conditions de respect de la vie privée sont moins efficaces que le Schéma 1. Le schéma 4 permettant le plus haut niveau de protection est, logiquement, peu efficace vis-à-vis des autres schémas que nous présentons. Cependant tous nos schémas sont bien plus efficaces que la solution de M.Blanton [Bla08], ce qui s'explique par le fait que cette solution utilise un grand nombre d'évaluations de couplages, qui est une opération coûteuse.

Méthode	Vie Privée	Profilage	Efficacité $\mathcal{AS}$.Souscris $f = 8, k = 3$		Efficacité $\mathcal{AS}$.AjouteAbo $f = 8, k = 3, l = 4$		Efficacité $\mathcal{AS}$.Utilise $\ell = 7$
M.Blanton [Bla08]	$U0$	$P2$	900.7 ms		n.a.		1428.96 ms
Schéma U0-P2	$U0$	$P2$	C' : 3.5 ms $\pi_{C'}$: 6.5 ms σ : 50.4 ms C : 4 ms	total : 64.4 ms	π_{abo} : 6.5 ms $\tilde{\sigma}$: 50.4 ms $\tilde{C}$: 3.9 ms	total : 60.8 ms	64.2 ms
Schéma U1-P2	$U1$	$P2$	C' : 3.5 ms $\pi_{C'}$: 43 ms σ : 50.4 ms C : 4 ms	total : 100.9 ms	π_{abo} : 43 ms $\tilde{\sigma}$: 50.4 ms $\tilde{C}$: 3.9 ms	total : 97.3 ms	64.2 ms
Schéma U2-P1	$U2$	$P1$	C' : 3.5 ms $\pi_{C'}$: 40 ms σ : 50.4 ms C : 4 ms	total : 97.9 ms	π_{abo} : 81.5 ms $\tilde{\sigma}$: 50.4 ms $\tilde{C}$: 3.9 ms	total : 135.9 ms	64.2 ms

Figure 10.1 – Comparaison Globale

Conclusion

J'ai présenté dans ce chapitre trois solutions issues de mes travaux avec S.Canard qui répondent au problème de l'abonnement anonyme tout en permettant au fournisseur de service de profiler ses utilisateurs. Ces solutions sont plus efficaces que l'état de l'art et permettent d'atteindre différents équilibres entre vie privée et profilage afin de s'adapter au mieux au contexte. Ces trois solutions ont fait l'objet d'un brevet [CJ09] et d'une publication à la conférence Trustbus 2010 [CJ10b].

Nous allons à présent nous intéresser à une dernière application permettant de gérer les droits d'accès à des contenus dans un groupe hiérarchique, tout en respectant la vie privée du groupe.

Chapitre 11

Gestion de contenus dans un groupe hiérarchisé

Sommaire

Dans une société de plus en plus orientée vers la communication et l'échange, la protection des contenus est devenue un enjeu majeur. Cette protection peut se présenter sous de multiples facettes, comme la confidentialité et l'intégrité des données, le contrôle parental, la protection des droits d'auteurs, etc.

Dans ce contexte, il existe de nombreuses situations dans lesquelles une approche de type "groupe" est particulièrement pertinente. Cette approche permet de traiter le cas d'un ensemble d'individus ou de terminaux, représentant par exemple la famille, un cercle d'amis ou encore une communauté d'intérêts. Il est alors intéressant d'intégrer les considérations de hiérarchie de droit d'accès dans le groupe. Enfin, il est important de prendre en compte la problématique du respect de la vie privée des membres du groupe, en particulier s'il s'agit d'un groupe familial, en permettant, par exemple, une gestion du groupe interne. Les solutions existantes dans ce

domaine sont bien adaptées à un ensemble de membres ayant tous accès aux mêmes contenus, mais s'appliquent mal au cas d'un groupe incluant une hiérarchie interne.

Nous nous intéressons dans ce chapitre à la gestion de contenus dans un groupe hiérarchisé, par exemple familial, dans lequel le gestionnaire de la famille met en place une hiérarchie de droit, par exemple, en fonction de l'âge (enfants, adolescents, parents, ...) et de la position vis-à-vis de la famille (membre de la famille, amis, extérieurs). Les travaux que je présente ici ont pour objectif d'adapter un outil classique de gestion de droit, les DRM (Digital Right Management), à une utilisation dans le cadre familial au sens large. Cette application, mise au point en collaboration avec S.Canard et M.Milhau, a donné lieu au brevet [CJM07]. L'outil cryptographique que nous utiliserons pour notre solution, la dérivation de clés dans un graphe hiérarchisé, est à la source de l'article [CJ08] que j'ai co-rédigé avec S.Canard.

11.1 Gestion de contenus dans un groupe hiérarchisé

Nous commencerons par une description de l'existant dans le domaine, je donnerai une description succincte du fonctionnement des DRM. Enfin, je décrirai les principes de notre solution qui permet la gestion de contenus dans un groupe hiérarchisé tout en respectant la vie privée des membres du groupe.

11.1.1 État de l'art

Le consortium OMA pour *Open Mobile Alliance*, qui définit des spécifications DRM pour les réseaux mobiles, propose une solution au problème du groupe de terminaux dans OMA DRM version 2 [OMA] : les Domaines. Un Domaine est un ensemble de terminaux qui partagent une même clé de Domaine, celle-ci étant délivrée par un fournisseur de licence. Les différents terminaux du Domaine peuvent partager des licences et donc s'échanger et lire n'importe quel contenu protégé couvert par ces licences. Le fournisseur de licence définit le domaine, gère les clés de domaines et contrôle quel terminal entre ou sort du Domaine. Les spécifications des Domaines faites par OMA DRM ont cependant des limitations gênantes dans les applications que nous souhaitons aborder.

- Premièrement le domaine est géré par un fournisseur de licence qui se trouve à l'extérieur du Domaine. Ce qui implique que les paramètres du domaine ne peuvent être modifiés par aucun membre du groupe.
- Deuxièmement, OMA gère les Domaines mais ne permet qu'une gestion des droits uniforme dans le groupe, ce qui n'est pas possible dans une famille : tous les contenus ne doivent pas être accessibles aux adolescents ou aux enfants.
- Pour finir le fournisseur de licence connaît énormément d'informations sur le groupe. Ce qui ne respecte pas la vie privée de la famille.

Une autre solution issue des travaux sur la domotique [PCTK04] est représenté par les Domaines Autorisés (ou *Authorized domain*). L'idée est de permettre aux différents éléments d'une maison de s'échanger des contenus sans restriction. Depuis, ce concept a été développé pour une application à la télévision sur IP [ADFD04, KKLV05, DVB]. Très proche des Domaines d'OMA DRM, cette solution ne prévoit pas la gestion de différents niveaux de droits dans le groupe.

La *Digital Living Network Alliance* [DLN] est le fruit de la collaboration d'industriels de l'électronique, de l'industrie informatique et d'opérateurs mobiles. Les membres de DLNA ont pour but l'élaboration de réseaux personnels sans-fils totalement interopérables. Le principe est de permettre aux utilisateurs de stocker, transporter ou utiliser leurs contenus n'importe où et

sur n'importe quel terminal. D'autre part, les fournisseurs de contenus souhaitent que ceux-ci soient protégés. La solution choisie est basée sur les DRM. Cette solution ne correspond pas non plus à nos exigences car elle ne gère pas de hiérarchie interne au groupe.

Enfin, différentes entreprises ont proposé des solutions propriétaires à une partie de notre problème. Cependant aucune solution n'existe, à notre connaissance, répondant à l'ensemble de nos exigences.

11.1.2 Principe des DRM

La technologie DRM est utilisée par différents acteurs, comme les créateurs, les éditeurs, les distributeurs de contenus numériques, les magasins en ligne et enfin les consommateurs. Bien que les DRM soient principalement connues pour leur application aux contenus multimédia, elles peuvent être appliquées à la protection de tout type de contenu numérique (textes, images, vidéos, ...).

Le contenu numérique est la cible de la protection et du contrôle effectué par les DRM. Le but étant de pouvoir définir des droits de lecture, de gravure et/ou de copie en fonction d'un contenu et du propriétaire du contenu. Ceci effectué grâce à l'utilisation de licence.

Définition 71 (Licence).
Une licence est un document indiquant les règles d'utilisation d'un contenu numérique. Elle est écrite en utilisant un langage d'expression de droit, noté REL (*Right Expression Langage*), qui définit la syntaxe et la sémantique des permissions et des contraintes qui régissent l'utilisation du contenu numérique. Elle est composée de différents champs dont les principaux sont :

- les identifiants de licence et de contenu,
- la CEK chiffrée avec la clé publique de l'utilisateur ayant demandé la licence,
  La CEK (*Ciphered Encryption Key*) est la clé que l'on utilise pour chiffrer le contenu. Celle-ci est généralement différente d'un contenu à un autre mais identique pour tous les utilisateurs souhaitant accéder à un même contenu.
- les droits de la licence (droits de lecture, gravure, etc.),
- la signature, sur l'ensemble des éléments de la licence, du fournisseur de licence ayant délivré la licence.

*

L'architecture standard des DRM permet à une entité A de protéger des œuvres numériques et de créer des licences pour ces contenus. Ainsi, une entité B s'étant procurée le contenu protégé devra demander une licence afin de pouvoir l'utiliser. Un tel système est composé des trois acteurs suivants.

- Le conditionneur de contenus, noté CC, protège le contenu numérique en le chiffrant avec la CEK. Il y ajoute, d'autre part, un entête contenant un identifiant ainsi qu'un ensemble de métadonnées parmi lesquelles l'URL du distributeur de licence, la date du conditionnement de la licence ainsi que des informations sur le contenu. On appelle contenu protégé un contenu numérique obtenu à la sortie du conditionneur de contenus. Après avoir protégé un contenu, CC fait parvenir l'identifiant du contenu et la CEK utilisée au fournisseur de licences à travers un canal sécurisé.
- Le fournisseur de licences, noté FL, génère et délivre les licences pour les contenus protégés. La livraison des licences peut se faire soit en réponse à une requête de l'agent DRM d'un client lors de la première utilisation d'un contenu, soit en amont (par exemple, lors du téléchargement d'un contenu protégé).

– L'agent DRM de l'utilisateur, noté $\mathsf{A_u}$, interprète et applique la licence. Il valide les droits et déchiffre le contenu. Un lecteur de contenu est souvent intégré à cette entité afin de lire les contenus numériques sans qu'ils ne sortent de l'agent DRM.

Informellement, un système DRM "classique" fonctionne de la façon suivante.

1. Le conditionneur de contenus utilise un algorithme de chiffrement symétrique pour chiffrer le contenu avec la CEK. Il stocke ensuite la clé de chiffrement ainsi qu'un identifiant de contenu dans une base de données accessible au fournisseur de licences.
2. Le consommateur récupère le contenu mais est dans l'impossibilité de le lire. Son agent DRM demande la licence pour son contenu en précisant la clé publique du consommateur.
3. Le fournisseur de licences crée la licence demandée. La CEK est chiffrée avec la clé publique du consommateur (afin qu'il soit le seul à pouvoir la retrouver et donc à lire le contenu grâce à cette licence). Cette clé publique est ensuite ajoutée aux métadonnées de la licence. Enfin, le fournisseur de licences fait parvenir la licence au consommateur.
4. Grâce à la licence et à sa clé privée, l'agent DRM du consommateur retrouve la clé symétrique CEK utilisée pour chiffrer le contenu. Il peut ainsi le déchiffrer et y accéder. On notera que le contenu est toujours stocké chiffré sur le terminal du client.

11.1.3 Notre solution

Nous souhaitons à présent adapter ce système au cas du groupe familial en permettant un haut niveau de respect de la vie privée. Nous avons besoin, d'une part, de pouvoir construire un groupe d'utilisateurs (ou de terminaux) ayant accès aux mêmes contenus sous réserve de l'autorisation du responsable du groupe (c'est-à-dire la possibilité du point de vue DRM, de créer une seule licence pour un groupe avec un contrôle supplémentaire interne) et, d'autre part, de pouvoir gérer une hiérarchie à l'intérieur de notre groupe. Pour cela, nous introduisons une nouvelle entité : le fournisseur de licences privé FLP. Celui-ci agit comme intermédiaire entre les membres du groupe et le fournisseur de licence. Le groupe apparait ainsi comme un utilisateur particulier et son architecture n'a pas besoin d'être divulguée à l'extérieur du groupe. Lors de l'acquisition d'un contenu, les membres du groupe interagissent uniquement avec le fournisseur FLP, celui-ci se chargeant des interactions avec FL. Informellement, le FLP transforme les "Licences de Groupes" provenant du fournisseur de licences FL en une licence compréhensible par les Agents DRM des membres du groupe (une "Licence Privée").

D'autre part, nous rassemblons les membres du groupe en sous-groupe en fonction de leurs droits d'accès. Tous les membres d'un sous-groupe ont ainsi les mêmes droits. Ces différents sous-groupes sont ensuite hiérarchisés en fonction des attributs choisis, FLP étant placé à la racine. Ces attributs nous permettent de déterminer le sous-groupe approprié à chaque membre du groupe. En ce qui concerne la famille, nous développons deux critères, l'âge et la relation à la famille, chacun est divisé en trois niveaux ordonnés : les adultes, les adolescents et les enfants pour l'âge et les membres de la famille, les amis et les extérieurs pour la relation à la famille. En croisant ces deux critères nous obtenons neuf sous-groupes qui peuvent être ordonnés en fonction de leurs droits. Les adultes de la famille ont accès à tous les contenus, tandis que les enfants de la famille n'ont accès qu'aux contenus pour enfant, etc. En considérant que les flèches représentent les relations entre les sous-groupes (en allant du sous-groupe père vers le sous-groupe fils) et qu'un sous-groupe père est capable de lire tous les contenus accessibles à ses sous-groupes fils, nous obtenons la représentation de la figure 11.1.

Nous supposons qu'une clé secrète peut être affectée à chaque sous-groupe de telle manière que tous membres d'un sous-groupe ascendant, et seulement eux, sont capables de la retrouver.

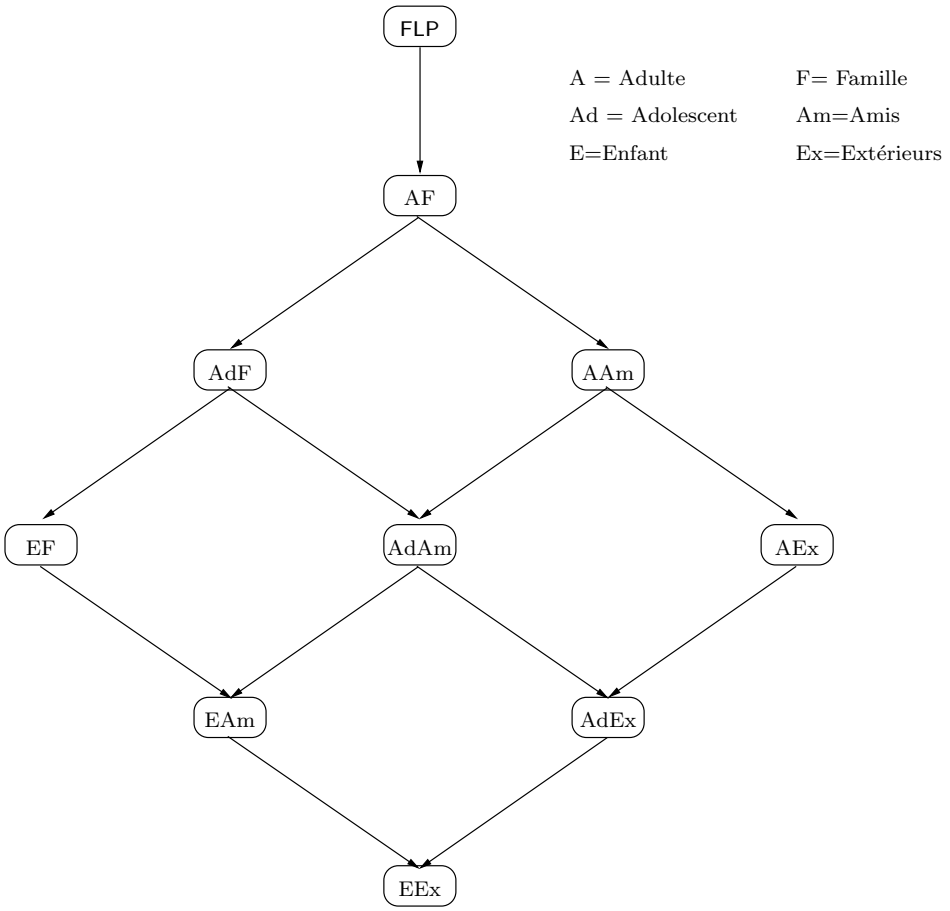

FIGURE 11.1 – Représentation par un Graphe

Nous pouvons alors adapter le fonctionnement du système DRM classique de la façon suivante. Pour chaque nouveau contenu, le responsable du groupe indique un niveau d'accès au fournisseur de licence privé FLP. Il indiquera, par exemple EF pour englober seulement les membres de la famille, AEx pour cibler uniquement les adultes ou encore EEx pour un contenu accessible à tous membre du groupe. La CEK comprise dans une licence de groupe est chiffrée avec la clé publique du FLP de la famille. Ainsi, celui-ci est capable de la déchiffrer. Il lui suffit ensuite de chiffrer le clair obtenu avec la clé sk_{cible} du sous-groupe cible, puis de remplacer, dans la Licence de Groupe, le chiffré de la clé CEK pour FLP, noté $\mathsf{c}_{CEK,\mathsf{FLP}}$, par le chiffré de la clé CEK avec la clé sk, $\mathsf{c}_{CEK,\mathsf{sk}}$. FLP signe ensuite la licence et obtient la Licence Privée qu'il publie. Une seconde solution, implique que le fournisseur de Licence FL utilise une signature caméléon, comme décrite au Chapitre 6, en indiquant FLP comme délégué et $\mathsf{c}_{CEK,\mathsf{FLP}}$ comme partie modifiable. Dans ce cas le fournisseur FLP remplace simplement le chiffré et adapte la signature du fournisseur FL.

Remarque. Les clés des différents sous-groupes sont des clés symétriques. D'après les spécifications OMA DRM version 2, la CEK doit être chiffrée avec RSA. Il y a ici deux possibilités. La première est de modifier les spécifications afin de permettre, dans notre cas spécifique, l'utilisation d'un algorithme symétrique. La deuxième est de construire une paire de clés RSA à partir de chaque clé symétrique. Il y a plusieurs solutions pour cela, on peut, par

exemple, utiliser la clé symétrique comme graine d'un générateur pseudo-aléatoire, la sortie obtenue étant utilisée à la place des aléas initialisant l'algorithme de génération de clé. Nous considérerons qu'une de ces deux possibilités est mise en place.

Pour développer une telle application, nous avons supposé qu'une clé secrète était affectée à chaque sous-groupe de telle manière que tous membres d'un sous-groupe ascendant, et seulement eux, soient capables de la retrouver. Un tel schéma est appelé un schéma de dérivation de clés dans un graphe hiérarchique.

11.2 Gestion de clés dans une hiérarchie

Nous allons à présent nous intéresser à la gestion de clés dans une hiérarchie quelconque. Ce cas inclut ainsi, entre autres, la hiérarchie présentée en section précédente. Et, en particulier, à un système dans lequel un groupe d'utilisateurs est réparti en sous-groupes en fonction d'une hiérarchie donnée. Le principe est alors qu'un membre d'un sous-groupe plus élevé dans la hiérarchie soit capable de retrouver la clé de ses sous-groupes subordonnés, l'inverse devant être impossible. Ce genre de hiérarchie dans un groupe est utilisée dans de nombreux domaines et il existe plusieurs façons de les représenter. Il est ainsi possible, dans certains cas, d'utiliser un arbre dans lequel les sommets représentent un sous-groupe d'utilisateurs ayant les mêmes droits mais cela ne permet pas de représenter une hiérarchie ayant plusieurs racines ou dans laquelle certains sous-groupes ont plusieurs supérieurs hiérarchiques directs (plusieurs sous-groupes pères) comme cela est le cas dans notre application. La méthode que je vais décrire ici permet de représenter la hiérarchie par un graphe orienté acyclique dont les sommets sont les sous-groupes d'utilisateurs ayant les mêmes droits. Cette représentation est cohérente avec la représentation d'une hiérarchie, le cas de l'arbre devient un cas particulier de notre représentation. Nous élargissons ainsi nos possibilités aux hiérarchies ayant plusieurs sous-groupes de plus haut niveau à leur tête ou ayant des sous-groupes avec plusieurs sous-groupes pères.

Nous commencerons par présenter le modèle de sécurité que nous utiliserons pour les schémas de gestion de clés dans une hiérarchie, nous nous intéresserons ensuite aux constructions proposées dans la littérature. Enfin, nous décrirons la solution de schéma de gestion de clés dans une hiérarchie que j'ai présenté à Indocrypt 2008 en collaboration avec S. Canard [CJ08].

11.2.1 Définitions et modèle

Avant de présenter notre modèle, nous devons définir quelques termes que nous empruntons à la théorie des graphes.

Graphe orienté Un graphe orienté G est un couple (S, A) avec S et A définis comme suit.
- $S = \{s_1, s_2, ..., s_{|S|}\}$ est l'ensemble des sommets de G. Les sommets sont notés s_i ou plus simplement i.
- $A = \{a_1, a_2, ..., a_{|A|}\}$ est l'ensemble de couples de sommets (i, j) appelé ensemble des arcs de G. Chaque arc (i, j) représente l'existence d'une arrête orientée du sommet i vers le sommet j.

Chemin Un chemin $C = \{a_1, \cdots, a_{|C|}\}$ est un ensemble de $|C|$ arcs tels que pour tout $i \in \{1, \cdots, |C| - 1\}$, si $a_i = (s_i, s_j)$ et $a_{i+1} = (s_k, s_l)$, alors $s_j = s_k$. C est alors un chemin entre a_1 et $a_{|C|}$.

Graphe orienté acyclique Un graphe orienté est dit acyclique s'il ne contient aucun cycle, c'est-à-dire s'il n'existe aucun chemin tel que deux sommets s_i, s_j avec $i \neq j$ soient identiques.

Sommet père et fils Soit $a = (i, j)$ un arc du graphe G. Le sommet i est appelé sommet père tandis que le sommet j est nommé sommet fils. Nous notons $P_i = \{p_i[1], \cdots, p_i[|P_i|]\}$ l'ensemble des $|P|$ pères du sommet i et $F_i = \{f_i[1], \cdots, f_i[|F_i|]\}$ l'ensemble des $|F|$ sommets fils du sommet i.

Sommet ascendant et descendant S'il existe un chemin entre le sommet i et le sommet j, nous dirons que i est un ascendant de j et j un descendant de i. Nous notons A_j l'ensemble des ascendant du sommet j et D_i l'ensemble des sommets descendants de i. Notons que $P_j \subset A_j$ et $F_i \subset D_i$.

Le modèle que nous présentons considère un graphe orienté acyclique $G = (S, A)$ dans lequel chaque sommet j représente un sous-groupe de membres partageant la même clé secrète sk_{s_j} (ou plus simplement sk_j) et la même valeur publique vp_{s_j} (ou plus simplement vp_j). La valeur publique est une information dépendante de la clé secrète qui sert à retrouver cette dernière à partir de la clé secrète d'un de ses pères. L'orientation du graphe représente la hiérarchisation des sommets. Un utilisateur membre d'un sous-groupe aura accès aux clés de tous ses descendants. Réciproquement, tous ses ascendants auront accès à sa clé secrète. Enfin, chaque sous-groupe est représenté lors de la génération des clés par un de ses utilisateurs. De manière plus formelle, nous définissons un protocole de gestion de clés dans un graphe de la façon suivante.

Définition 72 (Protocole de gestion de clés dans un graphe).
Soit $G = (S, A)$ un graphe orienté acyclique avec $S = \{s_1, s_2, ..., s_{|S|}\}$ l'ensemble de ses sommets et $A = \{a_1, a_2, ..., a_{|A|}\}$ l'ensemble de ses arcs. Un protocole de gestion de clés dans un graphe, noté $\mathcal{GKG}$, est un protocole entre un ensemble U d'utilisateurs notés $\mathsf{U} = \{\mathsf{u}_1, \cdots, \mathsf{u}_{|S|}\}$. Chaque utilisateur u_i représente un sommet s_i du graphe G dont la représentation est connue de tous. Le protocole est composé des cinq algorithmes et protocoles suivants.

$\mathcal{GKG}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système $\mathcal{GKG}.\mathsf{param}$ qui contiennent, entre autres, la description du graphe G. Nous considérerons dans la suite que λ est inclus dans les paramètres.

$$\mathcal{GKG}.\mathsf{param} \leftarrow \mathcal{GKG}.\textsc{Init}(1^\lambda)$$

$\mathcal{GKG}$.uInit permet à chaque utilisateur u_i de générer sa paire de clés long-terme $(\mathsf{upk}_i, \mathsf{usk}_i)$ en fonction des paramètres du système $\mathcal{GKG}.\mathsf{param}$. Nous considérerons dans la suite que param est inclus dans les clés publiques upk_i des utilisateurs.

$$(\mathsf{upk}_i, \mathsf{usk}_i) \leftarrow \mathcal{GKG}.\mathsf{uInit}(\mathcal{GKG}.\mathsf{param})$$

$\mathcal{GKG}$.GénClé est un protocole entre tous les utilisateurs $U = \{\mathsf{u}_1, \cdots, \mathsf{u}_{|U|}\}$ prenant en entrée leur paires de clés respectives $(\mathsf{upk}_i, \mathsf{usk}_i)$. Chaque utilisateur u_i génère la première version de la clé du sommet qu'il représente, notée $\mathsf{sk}_i[0]$. D'autre part, la première version des valeurs publiques du graphe $\mathsf{VP}[0]$ est publiée.

$$(\mathsf{VP}[0], \mathsf{u}_i : \mathsf{sk}_i[0]) \leftarrow \mathcal{GKG}.\textsc{GénClé}(\mathsf{u}_i : (\mathsf{upk}_i, \mathsf{usk}_i))$$

$\mathcal{GKG}$.MàJClé est un algorithme qui prend en entrée un sommet s_j dont la clé doit être mise à jour et lance un protocole entre un sous-ensemble des utilisateurs du système $\tilde{U} = \{\mathsf{u}_{i_1}, \cdots, \mathsf{u}_{i_k}\} \subset U$ tel que $u_j \in \tilde{U}$. Chaque utilisateur u_i prend en entrée le numéro de version courante ρ, sa clé de sommet correspondante $\mathsf{sk}_i[\rho]$ et la valeur publique correspondante $\mathsf{VP}[\rho]$. Chacun retourne secrètement sa nouvelle clé d'instance $\mathsf{sk}_i[\rho+1]$. D'autre part, la nouvelle version des valeurs publiques $\mathsf{VP}[\rho+1]$ est générée.

$$(\mathsf{VP}[0], \mathsf{u}_i : \mathsf{sk}_i[\rho+1]) \leftarrow \mathcal{GKG}.\textsc{MàJClé}(s_j, \mathsf{u}_i : (\rho, \mathsf{sk}_i[\rho], \mathsf{VP}[\rho]), \mathcal{GKG}.\mathsf{param})$$

$\mathcal{GKG}$.DérivClé prend en entrée un sommet s_j, un numéro de version ρ, la ρème version de la clé secrète d'un sommet s_i, notée $\mathsf{sk}_i[\rho]$, et la valeur publique correspondante $\mathsf{VP}[\rho]$. Il retourne soit un message d'erreur $\perp$ si $i \notin A_j$ (i n'est pas un ascendant de j) soit la ρème version de la clé secrète du sommet s_j, c'est-à-dire $\mathsf{sk}_j[\rho]$.

$$\{\perp, \mathsf{sk}_j[\rho]\} \leftarrow \mathcal{GKG}.\text{DérivClé}(s_j, \rho, \mathsf{sk}_i[\rho], \mathsf{VP}[\rho], \mathcal{GKG}.\mathsf{param}) \quad *$$

Remarque. L'efficacité de l'algorithme de rafraichissement de clés $\mathcal{GKG}$.MàJClé est essentielle dans le cas où les clés doivent être régulièrement changées, notamment si les membres des différents groupes changent au cours du temps. Il est alors nécessaire que lorsque la clé d'un sommet est mise à jour, le minimum de clés soient modifiées. Dans le meilleur des cas, on ne modifiera que la clé du sommet visé et les clés de ses sommets descendants car celles-ci peuvent, par définition, être retrouvées à partir de la clé du sommet ciblé. D'autre part, afin de simplifier les descriptions, notre modèle suppose qu'à un instant t toutes les clés partagent un même numéro de version. Informellement, cela revient à considérer qu'à chaque mise à jour, toutes les clés passent d'une version ρ à une version $\rho + 1$, les valeurs des clés pouvant rester identiques.

Enfin, nous considérons, ici, le cas d'un graphe statique pour lequel aucun sommet n'est ajouté ou retiré durant la vie du système. Nous verrons brièvement en dernière section comment notre construction peut être adaptée à ce cas.

11.2.2 Propriétés de sécurité

Contrairement aux constructions de mise en accord de clés dans un groupe, nous ne pouvons pas utiliser une expérience décisionnelle pour décrire la sécurité des schémas de dérivation de clés dans un graphe. Dans ce type d'expérience, l'adversaire doit distinguer une clé secrète d'un aléa et, dans notre cas, il suffirait à l'adversaire de corrompre un descendant du sommet cible pour distinguer la clé originale de l'aléa. En effet, la dérivation de la clé originale permet d'obtenir les clés secrètes des descendants.

Ainsi, nous considérons qu'un schéma de gestion de clés dans les graphes est sûr s'il respecte la propriété de sécurité suivante.

- **Récupération de clés**. Même une coalition d'utilisateurs doit être incapable de retrouver la clé d'un sommet qui ne fait pas parti de leurs descendants.

La définition formelle de cette propriété définit un adversaire $\mathcal{A}$ qui cherche à gagner une expérience contre un challengeur $\mathcal{C}$. Dans l'expérience, nous considérons que le graphe G^* est choisi par l'adversaire au début de l'expérience et qu'il est ensuite connu par toutes les parties. Une fois ce choix effectué, l'adversaire a accès aux six oracles suivants.

- L'oracle $\mathcal{O}$.Lance prend en entrée un graphe $G^* = (S, A)$ et 1^λ où λ est un paramètre de sécurité. Il initialise un gestionnaire de clé pour G^* en utilisant la procédure Init. La description du graphe G^* est alors comprise dans les paramètres publics du système $\mathcal{GKG}.\mathsf{param}$. $\mathcal{O}$.Lance retourne les paramètres du schéma $\mathcal{GKG}.\mathsf{param}$.

$$\mathcal{GKG}.\mathsf{param} \leftarrow \mathcal{O}.\text{Lance}(G^*, 1^\lambda)$$

- L'oracle $\mathcal{O}$.GénClé permet à l'adversaire $\mathcal{A}$ de lancer la génération des clés dans le graphe G^*. L'oracle retourne les échanges publics effectués entre les utilisateurs lors du protocole GénClé ainsi que la valeur publique initiale $\mathsf{VP}[0]$. D'autre part, une clé secrète initiale $\mathsf{sk}_i[0]$ est secrètement attribuée à chaque sommet s_i du graphe G^*.

$$\mathsf{VP}[0] \leftarrow \mathcal{O}.\text{GénClé}(\mathcal{GKG}.\mathsf{param})$$

– $\mathcal{O}$.MÀJ permet à l'adversaire $\mathcal{A}$ de mettre à jour la clé du sommet s_i de son choix, ceci pouvant impliquer la mise à jour d'autre clés de manière similaire à l'algorithme MÀJCLÉ. Chaque sommet change de numéro de version (de ρ vers ρ+1). L'oracle retourne la nouvelle version des valeurs publiques de l'instance $\mathsf{VP}[\rho+1]$.

$$\mathsf{VP}[\rho+1] \leftarrow \mathcal{O}.\text{MÀJ}(s_i)$$

– L'oracle $\mathcal{O}$.ENVOIE prend en entrée un sommet s_i et un message m. Il retourne la réponse rep de l'utilisateur u_i au message m.

$$rep \leftarrow \mathcal{O}.\text{ENVOIE}(i, \mathsf{m})$$

– L'oracle $\mathcal{O}$.CORROMPS prend en entrée un sommet s_i et un identifiant de version ρ. Il retourne la version ρ de la clé secrète sk_i du sommet s_i du graphe G^*, c'est-à-dire $\mathsf{sk}_i[\rho]$. Les clés des sommets descendant du sommet s_i sont indirectement données à l'adversaire. En effet, celui-ci est capable de les retrouver grâce à l'algorithme de dérivation de clés DÉRIVCLÉ. On note $SC[\rho]$ l'ensemble des sommets corrompus directement ou indirectement à la version ρ, nous avons donc si $s_i \in SC[\rho]$ alors $\forall j \in D_i, s_j \in SC[\rho]$.

$$\mathsf{sk}_i[\rho] \leftarrow \mathcal{O}.\text{CORROMPS}(s_i, \rho)$$

– Enfin l'oracle $\mathcal{O}.\mathsf{Ft}-\text{CORROMPS}$ prend en entrée un sommet s_i et retourne la clé secrète long-terme usk_i de l'utilisateur u_i. Lorsqu'un utilisateur est fortement corrompu, nous considérons que le sommet, et tous ses descendants, sont corrompus pour toutes les versions passées et à venir : $\forall \rho, s_i \in SC[\rho]$ et $\forall j \in D_i, \forall \rho, s_j \in SC[\rho]$.

$$\mathsf{usk}_i \leftarrow \mathcal{O}.\mathsf{Ft}-\text{CORROMPS}(s_i)$$

Pour des raisons de lisibilité des expériences, les entrées choisies par l'adversaire seront omises dans les oracles, de même le $\mathcal{O}$. sera sous-entendu lorsqu'il est évident. Par exemple, l'oracle $\mathcal{O}$.CORROMPS(i, ρ) sera abusivement noté CORROMPS.

Définition 73 (Récupération de clés).
Soient λ un paramètre de sécurité. Soit $\mathcal{A}$ un attaquant qui dans un premier temps choisit un graphe G^* à attaquer puis dans un deuxième temps, en ayant accès aux six oracles que nous venons de voir : $\mathcal{O}$.LANCE$(G^*, 1^\lambda)$, $\mathcal{O}$.GÉNCLÉ$(\mathcal{GKG}.\mathsf{param})$, $\mathcal{O}$.MÀJ(s_i), $\mathcal{O}$.ENVOIE(i, m), $\mathcal{O}$.CORROMPS(s_i, ρ) et $\mathcal{O}.\mathsf{Ft}-\text{CORROMPS}(s_i)$ et retourne un triplet (ρ^*, k^*, s^*) avec ρ^* le numéro de version, k^* une clé et s^* un sommet. Nous définissons l'expérience contre la **récupération de clés** comme suit.

$\text{EXP}^{\mathcal{GKG}}_{\textbf{RécClé},\mathcal{A}}(\lambda)$:
$(G^*) \leftarrow \mathcal{A}$
$(\rho^*, k^*, s^*) \longleftarrow \mathcal{A}^{\text{LANCE,GÉNCLÉ,MÀJ,ENVOIE,CORROMPS,}\mathsf{Ft}-\text{CORROMPS}}(G^*)$
Retourne 1 si
– $\forall j \in A_{s^*}, \forall \rho, j \notin SC[\rho]$
– $\forall \rho, s^* \notin SC[\rho]$
– $k^* = \mathsf{sk}_{s^*}[\rho^*]$.
Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{GKG}}_{\textbf{RécClé},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{GKG}}_{\textbf{RécClé},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{GKG}}_{\textbf{RécClé},\mathcal{A}}(\lambda)].$$

Un schéma de gestion de clés dans les graphes $\mathcal{GKG}$ respecte la propriété de **récupération de clés** s'il n'existe pas d'adversaire polynomial $\mathcal{A}$ tel que pour ϵ non négligeable :

$$\mathsf{Succ}^{\mathcal{GKG}}_{\textbf{RécClé},\mathcal{A}}(\lambda) > \epsilon.$$

*

Nous allons à présent définir la version sélective de cette propriété que nous utiliserons pour prouver la sécurité de la nouvelle solution que nous proposerons. La *sélective*-**récupération de clés** définit un adversaire $\mathcal{A}$ qui cherche à gagner une expérience contre un challengeur $\mathcal{C}$. Comme pour la version non sélective, nous considérons que le graphe G^* est choisi par l'adversaire au début de l'expérience. Cependant, dans la version sélective, le sommet ciblé s^* est lui aussi choisi par l'adversaire mais cette fois-ci au début de l'expérience. D'autre part, nous modifions légèrement les oracles permettant de corrompre un utilisateur, et donc le sommet qu'il représente, afin de rendre impossible la corruption du sommet cible ou de ses ascendants.

Définition 74 (*sélective*-récupération de clés).
Soient λ un paramètre de sécurité. Soit $\mathcal{A}$ un attaquant qui dans un premier temps choisit un graphe G^* et un sommet s^* de celui-ci à attaquer puis, dans un deuxième temps, en ayant accès aux six oracles $\mathcal{O}.\text{Lance}(G^*, 1^\lambda)$, $\mathcal{O}.\text{GénClé}(\mathcal{GKG}.\mathsf{param})$, $\mathcal{O}.\text{MàJ}(s_i)$, $\mathcal{O}.\text{Envoie}(i, \mathsf{m})$, $\mathcal{O}.\text{Corromps}(s_i, \rho)$ et $\mathcal{O}.\mathsf{Ft} - \text{Corromps}(s_i)$, retourne un couple (ρ^*, k^*). Nous définissons l'expérience contre la *sélective*-**récupération de clés** comme suit.

$\text{Exp}^{\mathcal{GKG}}_{\textit{sél}-\textbf{RécClé},\mathcal{A}}(\lambda)$:
$(G^*, s^*) \leftarrow \mathcal{A}$
$(\rho^*, k^*) \longleftarrow \mathcal{A}^{\text{Lance,GénClé,MàJ,Envoie,Corromps,}\mathsf{Ft}-\text{Corromps}}(G^*)$
Retourne 1 si
$k^* = \mathsf{sk}_{s^*}[\rho^*]$.
Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{GKG}}_{\textit{sél}-\textbf{RécClé},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{GKG}}_{\textit{sél}-\textbf{RécClé},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{GKG}}_{\textit{sél}-\textbf{RécClé},\mathcal{A}}(\lambda)].$$

Un schéma de gestion de clés dans les graphes $\mathcal{GKG}$ respecte la propriété de *sélective*-**récupération de clés** s'il n'existe pas d'adversaire polynomial $\mathcal{A}$ tel que pour ϵ non négligeable :

$$\mathsf{Succ}^{\mathcal{GKG}}_{\textit{sél}-\textbf{RécClé},\mathcal{A}}(\lambda) > \epsilon.$$

*

Le modèle de sécurité défini par la propriété de **récupération de clés** et celui défini par la version sélective sont différents du modèle habituellement utilisé [AFB05]. En effet, nous laissons ici l'adversaire choisir le graphe et le sommet cible alors que dans [AFB05] ils sont déterminés au cours de l'expérience par un oracle. Notre adversaire ayant plus de pouvoir, on aurait tendance à croire que notre modèle est plus fort. Cependant nous considérons que le sommet cible est choisi au début de l'expérience, alors qu'il est proposé dans [AFB05] de le fixer au milieu de l'expérience, c'est-à-dire après d'éventuelles requêtes aux oracles.

11.2.3 État de l'art

Le premier article consacré au problème de la gestion des clés dans une hiérarchie fut publié en 1983 par S.G.Akl et D.Taylor [AT83]. Depuis, un grand nombre de travaux en rapport avec la dérivation des clés dans une hiérarchie définie par un graphe orienté ont été effectués [MTMA85, CT90, ZHP91, HZS93, KSCL99, BZNR01, Lin01, Zho02, RRN02, HFKY03, SFM04, ZW04, CLTW04, CLL04, AFB05, DSGP05, WW05, ZKB07, GHK$^+$08]. Il est à noté que toutes ces propositions ne peuvent décrire qu'une hiérarchie dite centralisée. En effet, la grande majorité d'entre eux ne s'intéressent qu'à une hiérarchie pouvant être représentée par un graphe à une seule racine. Les rares schémas s'intéressant au cas à plusieurs racines utilisent une autorité de confiance ce qui nous ramène au premier cas en considérant l'autorité comme la racine unique. En dehors de ce point commun, ces schémas peuvent être regroupés en quatre familles.

La famille initiale

Cette famille correspond au premier article proposant une solution pour ce type de hiérarchie de A.G.Akl et D.Taylor [AT83] et aux différentes améliorations de ce schéma [MTMA85, CT90, KSCL99, HFKY03, SFM04]. Cette famille utilise une autorité centrale qui génère les clés des sommets ainsi que les valeurs publiques qui seront nécessaires pour les déduire les unes des autres. L'idée de cette famille de schémas est la suivante.

Construction 30.
Solution [AT83]

- On choisit deux grands entiers p et q premier et on publie $n = pq$.
- On choisit aléatoirement une clé maître $\mathsf{sk}_0 \in \{0,1\}^\lambda$.
- Les clés des sommets sont déterminées grâce à un entier public, noté t_i qui est choisi de telle sorte que (i) pour tout sommet $s_k \in A_i$, t_i est un multiple de t_k et (ii) pour tout sommet s_i, t_i n'est pas un multiple du plus grand diviseur commun des t_j tels que $s_j \notin A_i$. La clé du sommet s_i est $\mathsf{sk}_i = (\mathsf{sk}_0)^{t_i} \pmod n$. ⟲

La première condition sur les t_i assure que $\frac{t_i}{t_k}$ est un entier et donc que l'on peut calculer $\mathsf{sk}_i = \mathsf{sk}_k^{t_i/t_k} \pmod n$. La deuxième condition permet de rendre ce calcul impossible pour les sommets n'appartenant pas aux ascendants du sommet cible et d'empêcher les collisions.

[AT83] propose de générer les t_i en affectant un nombre premier p_i à chaque sommet puis en calculant les t_i comme le produit des p_j tels que s_j ne soit pas un ascendant de s_i.

$$t_i = \prod_{j \notin D_i} p_j$$

Exemple.
Pour illustrer cette solution, nous prenons l'exemple décrit dans la Figure 11.2. Nous affectons à chaque sommet un nombre premier $p_1 = 2$, $p_2 = 3$, etc. Nous déterminons ensuite l'exposant pour chaque sommet. Celui-ci correspond au produit des p_i affectés aux sommets qui ne sont pas des descendants.

Pour le sommet racine, s_1 tous les autres sommets sont des descendants, donc $t_1 = 1$. La clé correspondante au sommet s_1 est $\mathsf{sk}_1 = \mathsf{sk}_0^{t_1} = \mathsf{sk}_0$.

Pour le sommet s_2, les sommets qui ne sont pas des descendants correspondent au sommet racine ainsi qu'aux sommets s_3, s_7 et s_8. Nous avons $t_2 = p_1.p_3.p_7.p_8 = 2.5.17.19 = 3230$. La clé du sommet s_2 est $\mathsf{sk}_2 = \mathsf{sk}_0^{t_2} = \mathsf{sk}_0^{3230}$.

Et ainsi pour chaque sommet, nous obtenons une clé $\mathsf{sk}_i = \mathsf{sk}_0^{t_i}$.

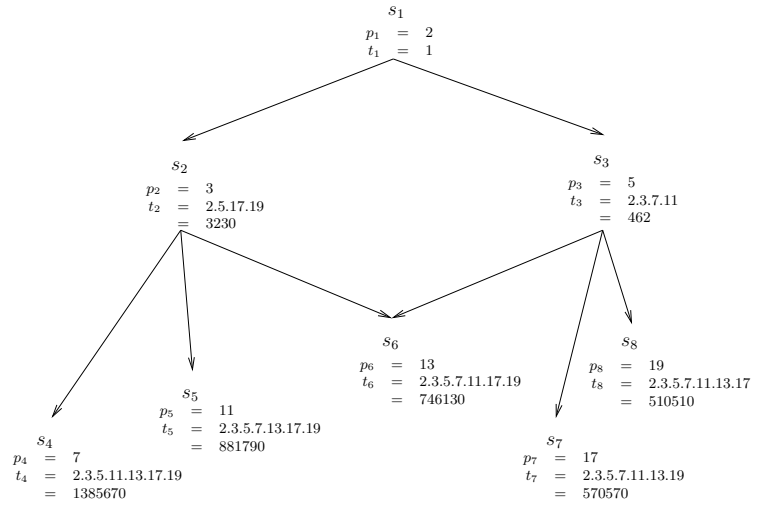

FIGURE 11.2 – Exemple de génération des t_i en fonction de [AT83]

Pour obtenir la clé du sommet s_4 à partir de la clé sk_2 du sommet s_2, il suffit d'utiliser les t_i qui ont été publiés.

$$\mathsf{sk}_4 = \mathsf{sk}_2^{\frac{t_4}{t_2}}$$

•

L'avantage principal de cette famille de solutions est qu'il s'agit d'une méthode directe. Ici, un ascendant s_k peut directement retrouver la clé d'un de ses descendants s_i en utilisant le bon rapport $\frac{t_i}{t_k}$, il lui suffit de calculer $\mathsf{sk}_i = \mathsf{sk}_k^{\frac{t_i}{t_k}}$. Cependant la taille des t_i augmente rapidement avec le nombre de sommets et la largeur du graphe. Dans notre exemple ne comprenant que 8 sommets et une largeur maximale de 5 sommets, on obtient déjà des t_i à 7 chiffres. Dans un graphe plus grand et surtout plus large, le problème peut atteindre des proportions contraignantes. [MTMA85, CT90, KSCL99, HFKY03, SFM04] ont amélioré cette solution mais elle reste cependant peu pratique. De plus, ces solutions utilisent toutes une autorité de confiance pour générer les p_i et les t_i. Enfin, il n'est pas toujours possible dans ces solutions de mettre à jour les clés des sommets, et lorsque cela est possible, une modification implique, au minimum, la mise à jour des clés d'une partie des ascendants.

La famille k-*SIFF*

La deuxième famille de solutions [ZHP91, HZS93] se base sur une famille de fonctions particulière les *k-SIFF* (ou *Sibling Intractable Function Family (SIFF)*) proposées par Y.Zheng, T.Hardjono et J.Pieprzyk [ZHP91]. Ils partent du constat que pour établir un schéma de génération et de délégation de clé indirecte, c'est-à-dire un schéma dans lequel on déduit la clé secrète d'un sommet en utilisant les clés secrètes de ses pères, on ne peut pas se contenter d'utiliser simplement une fonction de hachage pour déduire les clés. En effet, lorsqu'un sommet possède plusieurs pères, il faudrait utiliser une fonction de hachage permettant à tous les pères d'obtenir la même sortie tout en utilisant des entrées différentes car dépendantes de leurs clés secrètes respectives.

Exemple.
Si nous prenons l'exemple de la figure 11.3, chaque père $p_i[1], p_i[2]$ et $p_i[3]$ du sommet i a une

clé secrète distincte $\mathsf{sk}_{p_i[1]}, \mathsf{sk}_{p_i[2]}$ et $\mathsf{sk}_{p_i[3]}$. Il nous faudrait une fonction de hachage $\tilde{H}$ telle que

$$\tilde{H}(\mathsf{sk}_{p_i[1]}) = \tilde{H}(\mathsf{sk}_{p_i[2]}) = \tilde{H}(\mathsf{sk}_{p_i[3]}) = \mathsf{sk}_i$$

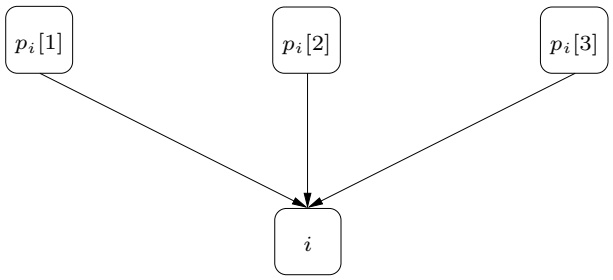

FIGURE 11.3 – Exemple de sommet ayant plusieurs pères

Or par définition une fonction de hachage doit être **résistante aux collisions**. [ZHP91] proposent dans cet objectif les *k-SIFF*, ces fonctions à sens unique permettent d'avoir pour un ensemble donné de k valeurs la même sortie et pour toutes autres valeurs une sortie distincte. Ces fonctions ont, en théorie, une efficacité et une sécurité similaire à une fonction de hachage exception faite de la collision exigée sur les k valeurs fixées. Cependant les travaux effectués dans [ZHP91, HZS93] prouvent uniquement l'existence théorique de telles fonctions mais ne proposent pas de construction. De plus, un tel schéma serait alors dépendant de l'efficacité de la construction proposée pour la génération de telles fonctions. En effet, à chaque rafraichissement d'une clé, la *k-SIFF* utilisée par les sommets pères devra être renouvelée. Enfin, cette solution implique l'utilisation d'une autorité de confiance qui connaitra les clés secrètes de tous les pères, ne serait-ce que pour générer la fonction *k-SIFF* correspondante.

La solution de J.Wu et R.Wei

La solution proposée par J.Wu et R.Wei dans [WW05] est basée sur le problème du logarithme discret. Cette solution fonctionne de la façon suivante.

Soit $\mathbb{Z}_p^*$ un groupe tel que $p = 2q + 1$ et p, q premiers. Le schéma se déroule dans un sous-groupe $\mathbb{G}$ de $\mathbb{Z}_p^*$ d'ordre q.

1. Pour chaque racine, une clé est aléatoirement choisie dans $[1, q]$.
2. À chaque sommet s_i qui n'est pas une racine, nous assignons publiquement un générateur g_i de $\mathbb{G}$ qui a été choisi aléatoirement.
3. La clé d'un sommet s_i dépend des clés de ses $|P_i|$ pères. Nous notons $\mathsf{sk}_{i,j}$ la clé du $j$ème père du sommet i.

$$\mathsf{sk}_i = \begin{cases} g_i^{\prod_{j=1}^{|P_i|} \mathsf{sk}_{i,j}} & \text{si } g_i^{\prod_{j=1}^{|P_i|} \mathsf{sk}_{i,j}} < q \\ p - g_i^{\prod_{j=1}^{|P_i|} \mathsf{sk}_{i,j}} & \text{sinon} \end{cases}$$

Cette solution utilise donc systématiquement les exponentiations modulaires et son efficacité est donc bien plus faible que les autres schémas de l'état de l'art. De plus, elle utilise une autorité de confiance pour l'initialisation du schéma et la génération des clés du système. Cependant cette solution fait partie des rares cas où les graphes à plusieurs racines sont réellement étudiés.

La famille des détectives

Le dernier groupe d'articles [BZNR01, Lin01, Zho02, ZW04, CLL04, CLTW04, DSGP05, AFB05, ZKB07] regroupe les solutions dans lesquelles les clés sont choisies aléatoirement par une autorité de confiance qui génère ensuite des indices publics pour passer d'une clé à l'autre en fonction des liens hiérarchiques. Un membre d'un sommet souhaitant obtenir la clé d'un de ses descendants choisira un chemin entre les deux sommets. Puis il déduira progressivement les clés en utilisant la clé du sommet précédent et les indices publiés par l'autorité afin d'obtenir la clé du sommet suivant. Différentes possibilités ont été proposées pour générer les indices. Par exemple [CLTW04, DSGP05, ZKB07] utilisent un système de polynôme d'interpolation, mais les solutions les plus efficaces utilisent uniquement des fonctions de hachages et/ou des opérations peu couteuses comme le "XOR" [BZNR01, Lin01, Zho02, ZW04, CLL04, AFB05].

11.3 Aperçu de notre solution

Je vais à présent introduire la solution de schéma de gestion de clés dans un graphe acyclique que j'ai proposé en collaboration avec S.Canard à INDOCRYPT 2008 [CJ08]. Cette solution fonctionne autant dans le cas centralisé où une entité (éventuellement la racine du graphe si elle est unique) génère toutes les clés puis les distribue, que dans un système décentralisé dans lequel tous les sommets participent à la génération des clés. Dans les deux cas, il est possible de construire les clés de la hiérarchie sans que tous les utilisateurs soient connectés en même temps.

Notre système de gestion de clés est basé sur deux méthodes. La première se concentre sur la dérivation des clés dans les cas où un sommet n'a qu'un seul sommet père (cas 1), la seconde prenant en compte les cas où un sommet à plusieurs pères (cas 2). Nous alternons ensuite ces deux méthodes afin de dériver les clés dans l'ensemble du graphe. En prenant un exemple, nous obtenons l'alternance de la figure 11.4.

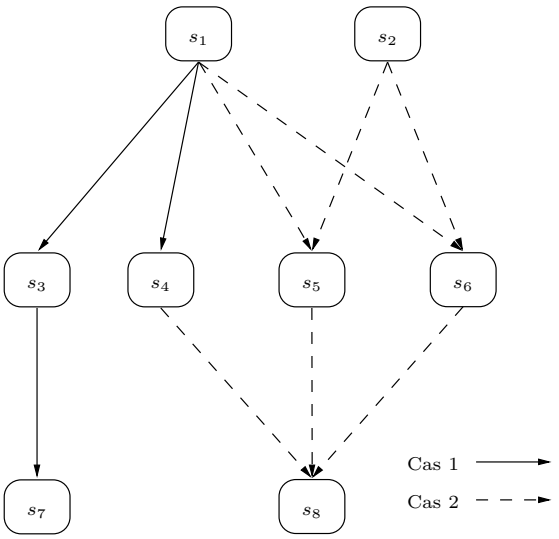

FIGURE 11.4 – Exemple de graphe hiérarchique

Remarque. Les méthodes que nous allons voir n'utilisent pas directement la clé affectée au sommet mais une sous clé distincte en fonction de la méthode utilisée. Un aléa est affecté à

chaque méthode, ces aléas étant ensuite publiés dans la valeur publique associée au sommet. Chaque sous clé est obtenue par une fonction pseudo-aléatoire sur l'aléa correspondant en utilisant la clé du sommet. Par souci de simplicité, nous mettons de coté cette subtilité dans cet aperçu et notons sk_{s_i} (ou plus simplement sk_i) les deux sous-clés du sommet s_i, le contexte indiquant la sous-clé utilisée.

Cette subtilité reflète d'autre part une réalité pratique. En effet nous utilisons une méthode spécifique pour chaque cas. Les deux méthodes peuvent ainsi utiliser des formats différents.

11.3.1 Cas d'un sommet à un seul père

Nous souhaitons dériver la clé du sommet fils de la clé du sommet père de façon la plus efficace possible. Nous avons donc choisi une méthode utilisant une fonction de code d'authentification de message basé sur une fonction de hachage (voir Chapitre 2 Section 2.2.4), notée $\mathrm{HMAC}(\mathsf{sk}, \mathsf{m})$.

D'autre part, il faut s'assurer que les clés des différents sommets fils d'un même père soient distinctes les unes des autres, pour cela nous utilisons un compteur spécifique au sommet s_i que nous notons s_i. Chaque valeur du compteur est utilisée une seule fois de la façon suivante. De plus, nous attribuons à chaque graphe un identifiant d'instance π choisit aléatoirement dans $\{0,1\}^\lambda$.

- On initialise à 0 le compteur s_i qui représente le nombre de clés générées à partir de la clé courante du sommet s_i.
- Les clés des sommets fils sont obtenues à partir de la clé du sommet père i de la façon suivante, avec π l'identifiant d'instance du graphe.

$$\forall j \in F_i, \quad \mathsf{s}_i = \mathsf{s}_i + 1, \quad \mathsf{sk}_{f_i[j]} = \mathrm{HMAC}(\mathsf{sk}_i, \pi \| \mathsf{s}_i)$$

Exemple.
Nous considérons le scénario décrit en Figure 11.5 dans laquelle sk_i est la clé du sommet i. Notre sommet i a trois fils $f_i[1]$, $f_i[2]$ et $f_i[3]$.

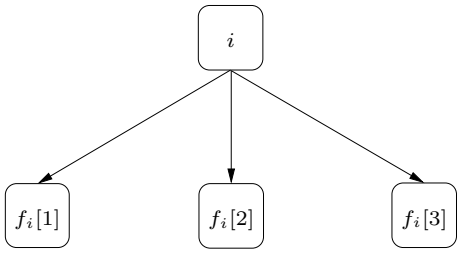

FIGURE 11.5 – Le cas à un seul père

Nous obtenons de cette façon les clés fils suivantes.

$$\mathsf{sk}_{f_i[1]} = \mathrm{HMAC}(\mathsf{sk}_1, \pi\|1) \quad \mathsf{sk}_{f_i[2]} = \mathrm{HMAC}(\mathsf{sk}_1, \pi\|2) \quad \mathsf{sk}_{f_i[3]} = \mathrm{HMAC}(\mathsf{sk}_1, \pi\|3) \qquad \bullet$$

11.3.2 Cas d'un sommet à plusieurs pères

Le cas où un sommet a plusieurs pères ne peut être traité de la même façon que le cas avec un seul père. En effet, comme vu pour les fonctions *k-SIFF*, la résistance à la collision de la fonction de hachage du HMAC empêche de prendre des entrées différentes dépendant des différentes clés secrètes des pères et de retourner une même sortie. Nous avons donc proposé une approche différente utilisant un schéma de mise en accord de clé dans un groupe non hiérarchique.

Cas simplifié. Nous souhaitons que tous les pères participent à la génération d'une clé commune. Pour cela nous utilisons un schéma de mise en accord de clé dans un groupe dans lequel la clé commune est initialisée lors d'un protocole entre les pères. De plus, les clés des sommets fils doivent pouvoir être recalculées de manière non-interactive par n'importe lequel des pères en utilisant uniquement la clé secrète de son sommet et un ensemble de paramètres et valeurs publiques : c'est ce que nous appelons un schéma *rejouable*. Enfin, la mise à jour des clés des pères doit impliquer la modification de la clé commune mais sans aucun impact sur les clés des autres pères : c'est ce que nous appelons un schéma *renouvelable*. Un schéma de mise en accord de clés ayant ces deux propriétés sera noté $\mathcal{RRGKA}$. Nous décrirons de manière plus formelle ce type de schéma dans la section suivante.

Nous observons deux points sensibles dans l'utilisation de tels schémas. Premièrement les solutions existantes qui respectent ces principes sont déterministes, donc un ensemble de pères utilisant les même clés de sommet obtiendra toujours la même clé commune. Ce qui implique nécessairement une adaptation dans le cas où un ensemble de pères ont en commun plusieurs fils. Une utilisation directe de ces schémas implique que pour le renouvellement de la clé fils, au moins un père doive changer de clé secrète ce qui est à éviter pour des raisons d'efficacité. Deuxièmement ces schémas ont une complexité bien supérieure au cas à un seul père, il faut donc les utiliser uniquement lorsque cela est indispensable, afin de ne pas trop alourdir la structure. Il est donc préférable de trouver une solution permettant d'obtenir un ensemble de clés différentes tout en utilisant le schéma $\mathcal{RRGKA}$ un nombre de fois minimum. Pour surmonter ces deux points, nous introduisons la notion de sommet virtuel.

Cas général : sommet virtuel. L'idée est de modifier le graphe en insérant un sommet entre l'ensemble des pères et le ou les fils qu'ils ont en commun. Ce sommet ne correspond à aucun sous groupe de la hiérarchie originale et la clé secrète correspondante ne sera utilisée que pour la dérivation de clé. Le schéma $\mathcal{RRGKA}$ est utilisé une fois pour l'obtention de la clé secrète sk_V de ce sommet virtuel. Les clés des sommets fils sont ensuite obtenues à partir du sommet virtuel en tant que seul sommet père de l'ensemble des sommets fils et donc en utilisant la première méthode à base de HMAC. Nous donnons à présent un petit exemple afin de mieux visualiser le fonctionnement de cette méthode.

Exemple.
Supposons que nous ayons trois sommets pères $p[1]$, $p[2]$ et $p[3]$ qui ont deux sommets fils en communs $f[1]$ et $f[2]$ comme décrit en Figure 11.6.

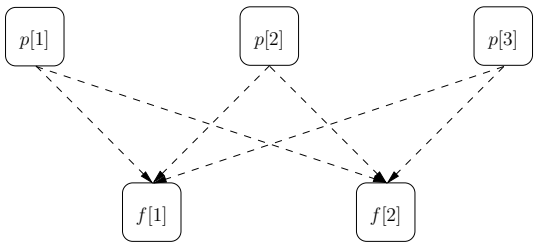

FIGURE 11.6 – Exemple de 3 sommets pères ayant 2 sommets fils en commun.

Dans un premier temps, notre méthode insère entre les sommets pères et fils un sommet virtuel que nous notons V. Nous obtenons ainsi le graphe de la Figure 11.7.

Nous pouvons ensuite dériver les clés des sommets fils. Nous utilisons d'abord le schéma $\mathcal{RRGKA}$ entre les pères $p[1], p[2]$ et $p[3]$ afin d'obtenir la clé sk_V du sommet virtuel V. Ce

que nous notons comme suit.

$$\mathsf{sk}_V = \mathcal{RRGKA}(p[1] : \mathsf{sk}_1, \quad p[2] : \mathsf{sk}_2, \quad p[3] : \mathsf{sk}_3)$$

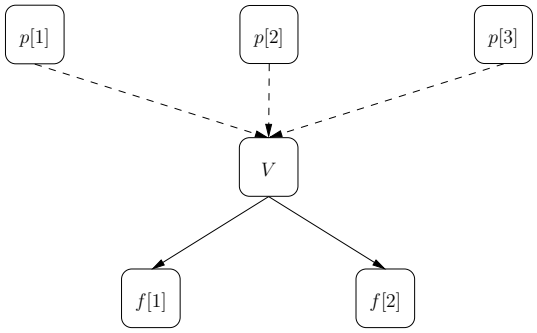

FIGURE 11.7 – Exemple après insertion du sommet virtuel

Nous déduisons ensuite les clés des sommets fils avec la méthode du cas 1 à partir de la clé secrète sk_V du sommet virtuel V.

$$\mathsf{sk}_{f_i[1]} = \mathrm{HMAC}(\mathsf{sk}_V, \pi \| 1) \quad \mathsf{sk}_{f_i[2]} = \mathrm{HMAC}(\mathsf{sk}_V, \pi \| 2)$$

•

Remarque. L'insertion d'un sommet virtuel peut être utilisée dans d'autres schémas de gestion de clés dans un graphe lorsque deux solutions distinctes sont utilisées en fonction du nombre de pères et que la solution à père unique est bien plus efficace. En particulier, les solutions utilisant les *k-SIFF* pourraient bénéficier de cette méthode.

Nous avons vu, dans cet aperçu, l'articulation de notre solution autour de deux méthodes : l'une basée sur un schéma de code d'authentification de message basé sur une fonction de hachage et la seconde sur un schéma de mise en accord de clés rejouable et renouvelable dans un groupe. La première méthode étant classiquement utilisée en cryptographie, une définition formelle de ce type de schéma a été présentée en première partie de ce mémoire. Par contre, la seconde est spécifique à notre solution, nous allons donc l'exposer de manière formelle avant de pouvoir décrire en détail notre méthode de dérivation de clés dans un graphe hiérarchisé.

11.4 Schéma rejouable et renouvelable de mise en accord de clés dans un groupe.

Un schéma de mise en accord de clés dans un groupe non hiérarchisé (ou *Group Key Agreement* en anglais), que nous notons $\mathcal{GKA}$, est un mécanisme qui permet à un groupe d'utilisateurs d'établir une clé secrète commune en se basant sur les contributions de chacun tout en ne communicant que sur un réseau public. La littérature propose de nombreux schémas $\mathcal{GKA}$.

Dans notre cas nous avons besoin d'un schéma ayant quelques propriétés supplémentaires qui sont déjà implicitement obtenues (ou qui peuvent être obtenues de manière triviale) par une grande partie des schémas existants [ITW82, STW00, LKKR03, KPT04b, KPT04a, BCP07a, BCP07b]. Nous introduisons ainsi les schémas de mise en accord de clés dans un groupe rejouable

et renouvelable (ou *Refreshable and Replayable Group Key Agreement* en anglais) noté $\mathcal{RRGKA}$. Cependant une adaptation des schémas traditionnels est à effectuer. En effet, l'information secrète habituellement choisie aléatoirement pour la clé secrète des participants est remplacée par une clé fixée en dehors du schéma. Un schéma $\mathcal{RRGKA}$ doit respecter les propriétés suivantes.

- Un $\mathcal{RRGKA}$ est un protocole *contributif* selon la définition de [KPT04b]. C'est-à-dire un schéma dans lequel tous les participants au protocole doivent apporter un élément secret nécessaire à la génération de la clé commune.
- Il est possible d'ajouter un algorithme déterministe permettant de rejouer la génération des clés après quelle soit générée. C'est-à-dire qu'une fois la clé commune initialisée grâce au protocole interactif entre les participants, n'importe quel représentant d'un sommet détenant la clé long-terme, la clé de version utilisée dans le protocole et un ensemble de valeurs publiques peut rejouer non-interactivement le protocole et retrouver la clé secrète de groupe.
- Le schéma contient une méthode de renouvellement de clé qui permet à un participant, ayant une clé corrompue, de renouveler sa clé de telle sorte que, en accord avec les autres participants, la clé commune soit modifiée mais sans nécessiter de changer la clé d'autres participants.

11.4.1 Définition d'un schéma rejouable et renouvelable de mise en accord de clés dans un groupe $\mathcal{RRGKA}$

Nous définissons un tel schéma de la façon suivante, adaptée de la définition de [BCP07a] pour les $\mathcal{GKA}$. On note $P = \{p_1, \ldots, p_{|P|}\}$ l'ensemble des participants potentiels.

Définition 75 (Schéma de mise en accord de clés rejouable et renouvelable).
Un schéma de mise en accord de clés dans un groupe rejouable et renouvelable, noté $\mathcal{RRGKA}$, est un schéma définit par les cinq algorithmes suivants.

$\mathcal{RRGKA}$.Init prend en entrée 1^λ où λ est un paramètre de sécurité. Il retourne les paramètres du système. On notera param ces paramètres. Nous considérons dans la suite que λ est inclus dans les paramètres.

$$\mathcal{GKA}.\mathsf{param} \leftarrow \mathcal{RRGKA}.\textsc{Init}(1^\lambda)$$

$\mathcal{RRGKA}.p$Init permet à chaque participant p_i de générer sa paire de clés long-terme, notée $(\mathsf{ppk}_i, \mathsf{psk}_i)$, en fonction des paramètres du système $\mathcal{GKA}.\mathsf{param}$. Nous considérons que $\mathcal{GKA}.\mathsf{param}$ est dorénavant inclus dans chacune des clés publiques ppk_i des utilisateurs.

$$(\mathsf{ppk}_i, \mathsf{psk}_i) \leftarrow \mathcal{RRGKA}.p\textsc{Init}(\mathcal{GKA}.\mathsf{param})$$

$\mathcal{RRGKA}$.GénClé est un protocole entre un ensemble de participants $\tilde{P} \subseteq P$. Chaque participant p_i prend en entrée sa paire de clés long-terme $(\mathsf{ppk}_i, \mathsf{psk}_i)$ et obtient une première version d'une clé secrète personnelle $\mathsf{sk}_{i,\tilde{P}}[0]$ ainsi que la première version de la clé commune, notée $\mathsf{sk}_{\tilde{P}}[0]$. D'autre part, la première version des valeurs publiques $\mathsf{VP}_{\tilde{P}}[0]$ est publiée.

$$(\{\mathsf{sk}_{i,\tilde{P}}[0]\}_{p_i \in \tilde{P}}, \mathsf{sk}_{\tilde{P}}[0], \mathsf{VP}_{\tilde{P}}[0]) \leftarrow \mathcal{RRGKA}.\textsc{GénClé}(p_i : (\mathsf{ppk}_i, \mathsf{psk}_i))$$

$\mathcal{RRGKA}$.MàJClé est un algorithme qui prend en entrée un ensemble de participants $\tilde{P} \subseteq P$ ayant une clé commune ainsi qu'un ensemble de participants $\tilde{P}^- \subseteq \tilde{P}$ dont la clé de version doit être remplacée. Si ρ est l'identifiant de version des clés, chaque participant $p_i \in \tilde{P}^-$

note $\mathsf{sk}_{i,\tilde{P}}[\rho+1]$ sa nouvelle clé secrète. Les participants $p_i \in \tilde{P} \setminus \tilde{P}^-$ conservent la même clé $\mathsf{sk}_{i,\tilde{P}}[\rho+1] = \mathsf{sk}_{i,\tilde{P}}[\rho]$. L'algorithme lance ensuite le protocole entre les participants de $\tilde{P}$, chacun prenant en entrée, sa clé long-terme $(\mathsf{ppk}_i, \mathsf{psk}_i)$, les deux instances de sa clé secrète personnelle $\mathsf{sk}_{i,\tilde{P}}[\rho+1]$ et $\mathsf{sk}_{i,\tilde{P}}[\rho]$, la clé secrète commune courante $\mathsf{sk}_{\tilde{P}}[\rho]$ et enfin la valeur publique $\mathsf{VP}_{\tilde{P}}[\rho]$. Chacun retourne secrètement la nouvelle clé commune $\mathsf{sk}_{\tilde{P}}[\rho+1]$. D'autre part, la nouvelle version des valeurs publiques $\mathsf{VP}_{\tilde{P}}[\rho+1]$ est publiée.

$$(\mathsf{sk}_{\tilde{P}}[\rho+1], \mathsf{VP}_{\tilde{P}}[\rho+1]) \leftarrow$$

$$\mathcal{RRGKA}.\text{MàJClé}(\tilde{P}^-, \tilde{P}, \mathsf{sk}_{\tilde{P}}[\rho]), \mathsf{VP}_{\tilde{P}}[\rho], p_i : (\mathsf{ppk}_i, \mathsf{psk}_i, \mathsf{sk}_{i,\tilde{P}}[\rho+1], \mathsf{sk}_{i,\tilde{P}}[\rho])$$

$\mathcal{RRGKA}$.RetrouveClé est un algorithme qui permet à un participant $p_i \in \tilde{P}$ d'obtenir la ρ-ème clé commune $\mathsf{sk}_{\tilde{P}}[\rho]$ à posteriori en prenant en entrée l'ensemble des participants $\tilde{P} \subseteq P$, la version de l'instance courante ρ, la ρ-ème version de sa clé personnelle $\mathsf{sk}_{i,\tilde{P}}[\rho]$ et les variables publiques correspondantes à la ρ-ème clé commune $\mathsf{VP}_{\tilde{P}}[\rho]$.

$$\mathsf{sk}_{\tilde{P}}[\rho] \leftarrow \mathcal{RRGKA}.\text{RetrouveClé}(\tilde{P}, \rho, \mathsf{sk}_{i,\tilde{P}}[\rho], \mathsf{VP}_{\tilde{P}}[\rho])$$

*

On notera que le protocole $\mathcal{RRGKA}$.GénClé pourra être utilisé de nombreuses fois avec la même initialisation, plusieurs instances du protocole peuvent être lancées en parallèle.

11.4.2 Propriétés de sécurité

Une propriété de sécurité très utilisée dans les schémas de mise en accord de clés [BCP01, BCPQ01, BCP02, KPT04b, KPT04a] est ce que nous appelons l'**Indépendance des clés** c'est-à-dire l'impossibilité pour un adversaire de distinguer une clé commune d'une valeur aléatoire.

Dans notre application, nous avons besoin qu'un adversaire ne puisse pas retrouver la clé d'un des participants au protocole : nous appellerons cette propriété la **Non déductibilité des clés initiales**. L'algorithme RetrouveClé permet à une entité connaissant la clé de l'instance courante d'un participant d'obtenir la clé commune courante et donc de distinguer une clé commune d'un aléa. En conséquence la propriété de **Non déductibilité des clés initiales** recouvre bien la propriété d'**Indépendance des clés**.

La définition formelle de la **Non déductibilité des clés initiales** définit un adversaire $\mathcal{A}$ qui cherche à gagner une expérience contre un challengeur $\mathcal{C}$. Dans l'expérience, l'adversaire a accès aux oracles suivants.

- $\mathcal{O}$.CréÉu génère un nouveau participant p_i avec une paire de clés long-termes $(\mathsf{psk}_i, \mathsf{ppk}_i)$ de manière similaire à l'algorithme pInit. Il retourne l'identifiant du participant i.

$$\mathcal{O}.\text{CréÉu}()$$

- L'oracle $\mathcal{O}$.GénClé permet à l'adversaire $\mathcal{A}$ de lancer la génération de la clé commune pour l'instance π. L'oracle retourne les échanges publics effectués entre les participants lors du protocole GénClé ainsi que la valeur publique initiale $\mathsf{VP}[0]$ de l'instance π que nous notons $\mathsf{VP}[\pi, 0]$. À noter que les clés secrètes des participants et la clé secrète commune $\mathsf{sk}_{\tilde{P}}[\pi, 0]$ ne sont pas divulguées à l'adversaire.

$$\mathcal{O}.\text{GénClé}(\pi)$$

– L'oracle $\mathcal{O}$.Envoie prend en entrée un identifiant de participant i, un numéro d'instance π et un message m. L'oracle retourne la réponse du participant p_i au message m dans l'instance du protocole π.

$$\mathcal{O}.\text{Envoie}(i, \mathsf{m}, \pi)$$

– $\mathcal{O}$.MàJ permet à l'adversaire $\mathcal{A}$ de mettre à jour la clé secrète d'un participant s_i de son choix. Ceci implique la mise à jour de la clé commune conformément à l'algorithme de mise à jour de clé MàJClé. Chaque sommet père change de numéro de version de clé (de ρ vers $\rho + 1$) et l'oracle retourne la nouvelle version des valeurs publiques de l'instance $\mathsf{VP}[\pi, \rho + 1]$.

$$\mathcal{O}.\text{MàJ}(s_i, \pi)$$

– L'oracle $\mathcal{O}$.Corromps retourne la version ρ de la clé secrète du participant i pour l'instance π, c'est-à-dire $\mathsf{sk}_{i,\tilde{P}}[\pi, \rho]$. La clé commune de la ρ-ème version est indirectement donnée à l'adversaire. En effet, celui-ci est capable de la retrouver grâce à l'algorithme RetrouveClé. On note $PC[\pi, \rho]$ l'ensemble des participants corrompus directement ou indirectement pour l'instance π à la version ρ.

$$\mathcal{O}.\text{Corromps}(i, \rho, \pi)$$

– L'oracle $\mathcal{O}.\mathsf{Ft} - \text{Corromps}$ prend en entrée un sommet i et retourne la clé secrète long-terme psk_i du participant p_i. Lorsqu'un participant est fortement corrompu dans une instance π, nous considérons qu'il est corrompu pour toutes les versions de l'instance π. Une fois un participant corrompu, l'adversaire peut choisir, pour toutes les versions futures, la valeur de la clé secrète du sommet $\mathsf{sk}_{i,\tilde{P}}[\pi, \rho]$ et interagir dans le protocole en lieu et place du participant.

$$\mathcal{O}.\mathsf{Ft} - \text{Corromps}(i)$$

– L'oracle $\mathcal{O}$.Révèle prend en entrée un identifiant d'instance π et un identifiant de version ρ. Il retourne la clé commune $\mathsf{sk}_{\tilde{P}}[\pi, \rho]$ avec $\tilde{P}$ les participants de l'instance π.

$$\mathcal{O}.\text{Révèle}(\pi, \rho)$$

Définition 76 (Non déductibilité des clés initiales).
Soient λ un paramètre de sécurité et Init l'algorithme d'initialisation. Soit $\mathcal{A}$ un attaquant ayant accès aux huit oracles [$\mathcal{O}$.CréÉu()r, $\mathcal{O}$.GénClé(π), $\mathcal{O}$.Envoie(i, m, π), $\mathcal{O}$.MàJ(s_i, π), $\mathcal{O}$.Corromps(i, ρ, π), $\mathcal{O}.\mathsf{Ft} - \text{Corromps}(i)$ et $\mathcal{O}$.Révèle(π, ρ), et qui retourne un quintuplet $(\pi^*, \tilde{P}^*, i^*, \rho^*, k^*)$. Nous définissons l'expérience contre la **Non déductibilité des clés initiales** comme suit.

$\text{Exp}^{\mathcal{RRGKA}}_{\textbf{NonDéd},\mathcal{A}}(\lambda)$:

$\mathcal{GKA}.\mathsf{param} \leftarrow \text{Init}(1^\lambda)$

$(\pi^*, \tilde{P}^*, i^*, \rho^*, k^*) \longleftarrow$

$\mathcal{A}^{\text{CréÉu,GénClé,Envoie,MàJ,Corromps,Ft−Corromps,Révèle}}(\mathcal{GKA}.\mathsf{param})$

Soit $PC[\pi^*, \rho^*]$ l'ensemble des participants corrompus pour la ρ^*-ème version de l'instance π^*.

Retourne 1 si

Pour l'instance π^*, il n'existe pas ρ' tel que pour la version ρ' tous les participants soient corrompus.

$i^* \notin PC[\pi^*, \rho^*]$,

$k^* = \mathsf{sk}_{\tilde{P}^*}[\rho^*]$.

Sinon retourne 0.

Le succès de l'adversaire $\mathcal{A}$ dans l'expérience $\text{Exp}^{\mathcal{RRGKA}}_{\textbf{NonDéd},\mathcal{A}}(\lambda)$ est :

$$\mathsf{Succ}^{\mathcal{RRGKA}}_{\textbf{NonDéd},\mathcal{A}}(\lambda) = Pr[1 \longleftarrow \text{Exp}^{\mathcal{RRGKA}}_{\textbf{NonDéd},\mathcal{A}}(\lambda)].$$

Un schéma de mise en accord de clés dans un groupe rejouable et renouvelable $\mathcal{RRGKA}$ respecte la propriété de **Non déductibilité des clés initiales** s'il n'existe pas d'adversaire polynomial $\mathcal{A}$ tel que pour ϵ non négligeable :

$$\mathsf{Succ}^{\mathcal{RRGKA}}_{\textbf{NonDéd},\mathcal{A}}(\lambda) > \epsilon.$$

*

Remarque. Il existe de nombreux schémas de mise en accord de clés dans la littérature qui respectent les propriétés nécessaires à un $\mathcal{RRGKA}$ que ce soit des schémas authentifiés [BCP07a, BCP07b] ou non [STW00, LKKR03, KPT04b, KPT04a].

11.5 Une nouvelle solution de dérivation de clés dans un groupe hiérarchique

Maintenant que nous avons formellement défini les schémas de mise en accord de clé rejouable et renouvelable dans un groupe, nous allons pouvoir décrire plus formellement notre solution de dérivation de clés dans un groupe hiérarchique. Il est à noter que les schémas de dérivation de clés dans un groupe hiérarchique se concentrent principalement sur le cas d'une hiérarchie statique, c'est-à-dire pour laquelle le graphe représentant la hiérarchie reste inchangée (aucun sommet n'est ajouté ou enlevé à posteriori). Néanmoins, nous verrons en section 11.5.2 que nous pouvons étendre notre schéma afin de supporter une hiérarchie dynamique.

Construction 31.

Nouvelle solution de dérivation de clés dans un graphe Notre solution utilise un schéma, rejouable et renouvelable, de gestion de clés dans un groupe que nous notons $\mathcal{RRGKA}$ (cf. Définition 75) et un schéma de codes d'authentification de message basés sur une fonction de hachage HMAC **résistant aux collisions** (voir Chapitre 2 Section 2.2.4). On note $G = (S, A)$ le graphe orienté acyclique dans lequel $S = \{s_1, s_2, ..., s_{|S|}\}$ est l'ensemble des sommets et $A = \{a_1, a_2, ..., a_a\}$ est l'ensemble des arcs. Le schéma fonctionne de la manière suivante.

$\mathcal{GKG}$.Init. Cet algorithme prend en entrée 1^λ avec λ un paramètre de sécurité et initialise le système. Il choisit aléatoirement un identifiant d'instance $\pi \in \{0,1\}^\lambda$ puis il initialise le schéma $\mathcal{RRGKA}$ en lançant la procédure $\mathcal{RRGKA}$.Init(1^λ). On note $GKA.\mathsf{param}$ les paramètres retournés. D'autre part, l'algorithme modifie le graphe en insérant les sommets virtuels nécessaires, comme expliqué en section 11.3.2 (voir exemple Figure 11.7). Nous notons $\tilde{G} = (\tilde{S}, \tilde{A})$ le nouveau graphe qui a les deux propriétés particulières suivantes. Premièrement, un ensemble de sommet pères n'a toujours qu'un seul fils : un sommet virtuel. Deuxièmement, tout sommet fils (qui n'est pas un sommet virtuel) n'a qu'un seul père : un sommet classique ou un sommet virtuel. Nous considérons dans la suite que les paramètres publics du système $\mathcal{GKG}.\mathsf{param}$ contiennent $GKA.\mathsf{param}$ et $\tilde{G}$.

$$\mathcal{GKG}.\mathsf{param} \leftarrow \mathcal{GKG}.\textsc{Init}(1^\lambda).$$

$\mathcal{GKG}$.uInit. Cet algorithme permet à chaque utilisateur de générer sa paire de clés long-termes $(\mathsf{usk}_i, \mathsf{upk}_i)$ en exécutant l'algorithme d'initialisation $\mathcal{RRGKA}.p\textsc{Init}(\mathcal{GKA}.\mathsf{param})$. Les clés publiques sont insérées dans $\mathcal{GKA}.\mathsf{param}$ et donc dans $\mathcal{GKG}.\mathsf{param}$.

$$(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathcal{GKG}.\textsc{uInit}(\lambda)$$

$\mathcal{GKG}$.GénClé. On affecte d'abord les clés des sommets racine. Pour tout sommet racine s_i du graphe la clé secrète du sommet $\mathsf{sk}_i[0]$ est choisie aléatoirement dans $\{0,1\}^\lambda$. $\mathsf{Nonce}_i^{(1)}$ et $\mathsf{Nonce}_i^{(2)}$ sont choisit aléatoirement dans $\{0,1\}^\lambda$. Nous utilisons une sous-clé différente pour chaque méthode. Chaque sommet a donc deux sous-clés définies comme suit.

$$\mathsf{sk}_i^{(1)}[0] = \mathcal{PRF}(\mathsf{sk}_i[0], \mathsf{Nonce}_i^{(1)}) \quad \text{et } \mathsf{sk}_i^{(2)}[0] = \mathcal{PRF}(\mathsf{sk}_i[0], \mathsf{Nonce}_i^{(2)})$$

La valeur publique des sommets racines sont de la forme $vp_i[0] = \mathsf{Nonce}_i^{(1)} \| \pi \| \mathsf{Nonce}_i^{(2)}$. L'algorithme procède ensuite de proche en proche. Pour chaque sommet $s_i \in \tilde{S}$ n'ayant pas encore sa clé et dont tous les pères ont obtenu la leur, on procède de la manière suivante.

- *Si i n'a qu'un seul père* $p = s_j \in S$. On applique la méthode "cas 1". Si cela n'a pas encore été fait, on initialise à 0 le compteur s_j lié au sommet père : $\mathsf{s}_j = 0$. On ajoute ensuite 1 au compteur $\mathsf{s}_j = \mathsf{s}_j + 1$ ce qui représente le fait de calculer une clé supplémentaire à partir de la clé du sommet j.

  $$\mathsf{sk}_i[0] = \mathrm{HMAC}(\mathsf{sk}_j^{(1)}[0], \pi \| \mathsf{s}_j) \quad \text{avec } \mathsf{sk}_j^{(1)}[0] = \mathcal{PRF}(\mathsf{sk}_j[0], \mathsf{Nonce}_j^{(1)}).$$

  La valeur publique liée au sommet i est la valeur du compteur s_j au moment de la génération de la clé concaténée à l'identifiant du sommet, soit $\mathsf{s}_j \| i$ ainsi que $\mathsf{Nonce}_j^{(1)}$ et $\mathsf{Nonce}_j^{(2)}$ choisis aléatoirement.
- *Si i a plus d'un père*, nous avons $P = (p_1 = s_{j_1}, \cdots, p_{|P|} = s_{j_{|P|}})$. On applique la méthode "cas 2" en exécutant le protocole $\mathcal{RRGKA}$.GénClé sur l'ensemble des pères P. Chaque père p_i utilise sa sous-clé de sommet $\mathsf{sk}_{p_i}^{(2)}[0] = \mathcal{PRF}(\mathsf{sk}_{p_i}[0], \mathsf{Nonce}_i^{(2)})$ comme clé secrète personnelle dans le protocole. La clé commune $\mathsf{sk}_P[0]$ obtenue est affectée au sommet i : $\mathsf{sk}_i[0] = \mathsf{sk}_P[0]$. La valeur publique $vp_i[0]$ du sommet fils s_i est la valeur publique $\mathsf{VP}_P[0]$ générée durant le protocole, l'identifiant du sommet i ainsi que $\mathsf{Nonce}_i^{(1)}$ et $\mathsf{Nonce}_i^{(2)}$ choisis aléatoirement.

À la fin, une clé secrète initiale $\mathsf{sk}[0]$ a été affectée à chaque sommet et l'algorithme retourne la première version de la valeur publique générale est publiée $VP[0] = \{vp_i[0]\}_{i \in S}$.

$$(\{\mathsf{sk}_i[0]\}_{i \in S}, \mathsf{VP}[0]) \leftarrow \textsc{GénClé}(\mathsf{u}_i : (\mathsf{upk}_i, \mathsf{usk}_i), \mathcal{GKG}.\mathsf{param})$$

$\mathcal{GKG}$.DérivClé. La première phase de cet algorithme est le choix du meilleur chemin du sommet dont la clé est connue s_a vers le sommet cible s_b. S Le choix du chemin le plus court (comportant le moins d'arcs) n'est pas forcément le meilleur en termes d'efficacité. En effet le coût de la méthode 1 (avec un père unique) est bien plus efficace que la méthode 2 (avec plusieurs pères). En conséquence le meilleur chemin est celui ayant le plus petit nombre d'arc dans le cas 2. Ce chemin peut être trouvé soit par recherche exhaustive, soit par un algorithme de recherche de chemin pour les graphes pondérés comme, par exemple, l'algorithme de Dijkstra [Dij59]. Nous considérons à présent que nous avons choisi une méthode pour trouver le meilleur chemin $C_{a,b}$ entre le sommet a et le sommet b et que s'il n'existe pas de chemin $C_{a,b}$ du sommet s_a vers le sommet cible s_b, l'algorithme retourne $\perp$ (s_a n'est pas un ascendant de s_b).
Pour la seconde phase, l'algorithme s'intéresse à chaque sommet s_i du chemin $C_{a,b}$ du premier au dernier de la façon suivante.

- Si s_i a un seul père $p_i = s_j$, nous recalculons la sous-clé du père correspondant au cas 1 à père unique $\mathsf{sk}_j^{(1)}[\rho] = \mathcal{PRF}(\mathsf{sk}_j[\rho], \mathsf{Nonce}_j^{(1)})$. La partie de $\mathsf{VP}[\rho]$ concernant le sommet s_i nous permet de retrouver la valeur s_j utilisée pour la clé du sommet s_i. On obtient cette clé de la façon suivante.

$$\mathsf{sk}_i[\rho] = \mathrm{HMAC}(\mathsf{sk}_j[\rho], \pi \| \mathsf{s}_j).$$

- Si s_i a plusieurs pères, nous posons $P_i = \{p_i[1], \cdots, p_i[|P|]\}$ l'ensemble de ceux-ci. La partie publique de $\mathsf{VP}[\rho]$ concernant le sommet s_i nous permet d'obtenir la valeur publique $\mathsf{VP}_P[\rho]$ générée durant le protocole de génération de la clé commune aux pères P. Soit $p_i[k] = s_j$ le père dont la clé secrète $\mathsf{sk}_j[\rho]$ est connue. La sous-clé correspondante dans le cas 2 (plusieurs pères) est $\mathsf{sk}_j^{(2)}[\rho] = \mathcal{PRF}(\mathsf{sk}_j[\rho], \mathsf{Nonce}_j^{(2)})$. L'algorithme utilise la procédure $\mathcal{RRGKA}$.RetrouveClé$(P, \rho, \mathsf{sk}_j^{(2)}[\rho], \mathsf{VP}_P[\rho])$ qui retourne la clé $\mathsf{sk}_i[\rho]$.

Nous obtenons ainsi les clés de tous les sommets sur le chemin $C_{a,b}$ du premier au dernier. Ainsi, la dernière clé obtenue est la clé $\mathsf{sk}_b[\rho]$.

$$\{\perp, \mathsf{sk}_b[\rho]\} \leftarrow \text{DérivClé}(s_b, \rho, \mathsf{sk}_a[\rho], \mathsf{VP}[\rho], \mathcal{GKG}.\mathsf{param})$$

$\mathcal{GKG}$.MàjClé. Comme nous insérons des sommets virtuels, le sommet cible s a obligatoirement un seul sommet père $p_s = s_i$. Celui-ci peut être soit un sommet virtuel soit un sommet de G. Si ρ est le numéro de version courant, alors nous avons $\mathsf{sk}_i[\rho+1] = \mathsf{sk}_i[\rho]$. Nous avons $\mathsf{sk}_i^{(1)}[\rho+1] = \mathcal{PRF}(\mathsf{sk}_i[\rho+1], \mathsf{Nonce}_i^{(1)})$ la sous-clé du père correspondant au cas à père unique pour la $\rho+1$ème version. De plus nous avons s_i le compteur courant du sommet s_i qui représente le nombre de clés déjà générées à partir de la sous clé $\mathsf{sk}_i^{(1)}[\rho]$ du sommet i. On pose $\mathsf{s}_i = \mathsf{s}_i + 1$ puis on calcule la clé du sommet s :

$$\mathsf{sk}_s[\rho+1] = \mathrm{HMAC}(\mathsf{sk}_i^{(1)}[\rho+1], \pi \| \mathsf{s}_i).$$

La valeur publique correspondante devient $vp_i = \mathsf{s}_j \| i$. Le compteur courant du sommet i est remis à zéro $\mathsf{s}_i = 0$.
Nous devons à présent mettre à jour les clés des descendants du sommet i. Pour chaque sommet s_k descendant de s_i il y a deux possibilités en fonction du nombre de pères de s_k. S'il n'en a qu'un, cela revient à ce que nous venons de décrire. Sinon nous notons P_k l'ensemble de ses pères et P_k^- le sous-ensemble des pères ayant changés de clé entre les versions ρ et $\rho+1$. Chaque utilisateur $\mathsf{u}_i \in P_k$ calcule sa sous clé pour le cas 2, version

$\rho+1 : \mathsf{sk}_i^{(2)}[\rho+1]$. Les pères participent ensuite au protocole $\mathcal{RRGKA}$.MàJClé afin de mettre à jour la clé commune en fonction des clés modifiées des pères appartenant à P_k^-. Le sommet s_k obtient en conséquence une nouvelle clé $\mathsf{sk}_k[\rho+1]$ et la valeur publique courante $\mathsf{VP}[\rho+1]$ mise à jour.

$$(\{\mathsf{sk}_i[\rho+1]\}_{i\in[i_1,i_k]}, \mathsf{VP}[O]) \leftarrow \text{MàJClé}(s, \mathsf{u}_i : (\rho, \mathsf{sk}_i[\rho], \mathsf{VP}[\rho]), \mathcal{GKG}.\mathsf{param})$$

Remarque. Au cours de la procédure GénClé, il est possible pour un utilisateur de tricher en ne donnant pas la bonne clé à un de ses descendants. Afin de détecter une telle fraude, il est possible d'ajouter une procédure de confirmation. Pour cela, à la fin de l'algorithme GénClé, chaque sommet publie un label relié à sa clé secrète de façon à ce que n'importe quel ascendant d'un sommet ayant une fausse clé puisse le détecter. Une fois la fraude détectée, il est facile de retrouver le coupable.

11.5.1 Preuve de sécurité

Théorème 13. *Sous l'hypothèse que le schéma de code d'authentification de message basée sur une fonction de hachage* HMAC *est* ***EU-CMA****, que le schéma de mise en accord de clé dans un groupe rejouable et renouvelable $\mathcal{RRGKA}$ vérifie la propriété de* ***Non déductibilité des clés initiales*** *et que le schéma de fonction pseudo-aléatoire $\mathcal{PRF}$ soit* ***pseudo-aléatoire*** *alors le nouveau schéma de gestion de clés dans un graphe orienté et acyclique est sûr dans le modèle : il respecte la propriété de sélective-****récupération de clés****.* ♢

Pour des raisons de lisibilité et pour ne pas alourdir ce chapitre déjà long, je ne donnerais ici que l'idée de la sécurité de notre schéma.

Notre construction est principalement basée sur une solution de $\mathcal{RRGKA}$, une $\mathcal{PRF}$ et un HMAC. Moralement il est possible de se convaincre de la sécurité de cette solution en considérant dans quels cas un adversaire peut déduire la clé d'un sommet donné. S'il ne connait la clé d'aucun sommet fils de celui-ci alors il ne peut l'avoir obtenue qu'au hasard. S'il connait des clés de certains de ses sommets fils n'ayant que ce sommet comme père alors l'adversaire peut être utilisé pour inverser la fonction HMAC et celle-ci n'est donc pas **EU-CMA**. Enfin, s'il connait les clés de certains des sommets fils qui ont plusieurs pères, dont le sommet cible, alors il peut être utilisé pour casser la propriété de *sélective-***récupération de clés** du schéma $\mathcal{RRGKA}$.

11.5.2 Le cas dynamique

Notre construction principale considère déjà le renouvellement d'une clé d'un sommet existant. Cependant, comme cela est le cas pour la majorité des schémas de dérivation de clés dans un graphe, nous n'avons pas abordé la possibilité d'ajouter ou d'enlever un sommet a posteriori. Nous indiquons ici brièvement comment notre schéma peut être étendu à ces cas. Pour cela nous devons ajouter deux nouvelles procédures au modèle.

- Ajouteest un algorithme qui prend en entrée deux ensembles $S_1, S_2 \subset S$ de sommets et lance un protocole entre les utilisateurs afin de générer un nouveau sommet ayant comme pères les sommets S_1 et comme fils les sommets S_2.
- Supprimeest un algorithme qui prend en entrée un sommet s_i et lance un protocole entre les utilisateurs afin de le supprimer du graphe.

Les oracles correspondants à ces deux algorithmes sont ajoutés dans l'expérience. La définition de l'expérience restant globalement inchangée nous ne la répéterons donc pas ici.

L'algorithme Ajoute. L'introduction du sommet virtuel nous permet d'insérer plus facilement un nouveau sommet noté s_n. En fait, la mise en place du nouveau sommet dépend du nombre de pères dans l'ensemble S_1. Plus précisément, la procédure fonctionne de la manière suivante.

1. La génération de la clé du nouveau sommet.
   - Si $|S_1| = 1$, alors s_n a un unique père, que nous notons p_n. Nous utilisons le cas à père unique pour générer la nouvelle clé. Notons que le compteur s_{p_n} correspond au nombre de clés générées avec la clé du sommet s_n. Nous l'augmenterons donc de 1 à chaque nouvelle clé, y compris dans ce cas.
   - Si $|S_1| > 1$, il y a deux possibilités.
     - Soit il existe déjà un sommet virtuel entre les pères S_1. Dans ce cas, nous revenons au cas précédent avec le sommet virtuel comme père unique.
     - Soit il n'existe pas encore de sommet virtuel. Dans ce cas nous l'ajoutons au graphe et nous utilisons le cas 2, à plusieurs pères, pour générer la clé du sommet virtuel. Nous nous ramenons ensuite au premier cas, le nouveau sommet virtuel étant l'unique père du nouveau sommet s_n.
2. L'étude des sommets fils du nouveau sommet.

   Pour chaque sommet fils f_n nous observons le nombre de père prévus.
   - Si le sommet f_n n'a qu'un seul père nous utilisons la méthode a père unique pour générer sa clé.
   - S'il a d'autres pères, il y a deux possibilités.
     - Soit il existe déjà un sommet virtuel entre les pères tels que tous les fils communs seront aussi des fils du nouveau sommet. Dans ce cas nous utilisons la dynamique du schéma $\mathcal{RRGKA}$ (ajout d'un participant) et nous obtenons la clé du sommet virtuel. Nous déduisons ensuite la (ou les) clés du (ou des) sommets fils grâce à la méthode à père unique.
     - Soit un tel sommet virtuel n'existe pas. Dans ce cas nous le créons en utilisant le schéma $\mathcal{RRGKA}$ puis nous en déduisons la clé du sommet fils grâce à la méthode à père unique.
3. L'étude des descendants. Les clés des sommets descendants du nouveau sommet sont renouvelés grâce à la procédure MàJClé.

L'algorithme Supprime. Nous souhaitons à présent supprimer le sommet s. Ce sommet ne peut être un sommet virtuel. Il ne peut donc pas avoir plusieurs pères. S'il n'a pas de pères, alors nous supprimons simplement le sommet et les arcs qui y sont liés. Nous remplaçons les clés des sommets fils par des aléas puis nous rafraichissons les clés des descendants en conséquence. Enfin, nous retournons le nouveau graphe avec les valeurs publiques mises à jour. Sinon, nous notons p le sommet père de s et nous nous intéressons à chacun de ses fils, un par un, de la manière suivante.

1. Nous modifions le graphe.

   Pour tous $f_i \in F_s$, nous cherchons un chemin C_{p,f_i} entre le père p et le fils f_i. Si le chemin existe nous supprimons l'arc (s, f_i). S'il n'existe pas, nous ajoutons un arc entre p et f_i puis nous supprimons l'arc (s, f_i).

   Le sommet s n'a plus de sommet fils. Nous le supprimons et retirons l'arrête entre celui-ci et son père.

2. Nous renouvelons les clés des fils.

   Pour chaque fils $f_i \in F_s$, nous regardons son (ou ses) père(s). S'il en a plusieurs, nous utilisons la dynamique du schéma $\mathcal{RRGKA}$ (suppression d'un participant) pour retirer s. De plus, s'il n'existe pas de chemin entre le fils f_i et p, nous l'ajoutons aux participants au $\mathcal{RRGKA}$. La nouvelle clé commune est la nouvelle clé secrète du fils f_i. Si f_i n'a qu'un seul père dans la nouvelle architecture, la clé du fils est déduite de celui-ci par la méthode du cas 1.

3. Enfin, nous rafraichissons les clés de tous les sommets descendants par un MàJClé.

Conclusion

J'ai présenté dans ce chapitre une solution mise au point avec S.Canard qui permet de dériver des clés dans une hiérarchie représentée par un graphe orienté. Pour la première fois nous obtenons une solution permettant de gérer des hiérarchies représentées par un graphe à plusieurs racines de manière efficace et sans autorité de confiance. De plus, la stratégie que nous proposons repose, pour les dérivations les moins efficaces, sur un schéma de mise en accord de clé rejouable et renouvelable générique et non sur une instance particulière. Ceci nous permet d'utiliser la littérature des schémas de mise en accord de clé pour cette partie et donc bénéficier, sans modification, des avancées dans ce domaine. Cette solution a été publiée à la conférence Indocrypt 2008 [CJ08].

Ce schéma a été développé dans l'objectif de gérer des contenus dans un groupe hiérarchisé tout en respectant la vie privée des membres du groupe. Cette application a donné lieu à un brevet avec S.Canard et M.Milhau [CJM07].

Conclusion

J'ai présenté au cours de ce mémoire mes contributions à la cryptographie issues de travaux en collaboration avec O.Blazy, S.Canard, I.Coisel, G.Fuchsbauer, M.Izabachène, E.Malville, M.Milhau, H.Sibert, J.Traoré et D.Vergnaud.

Dans un premier temps, j'ai exposé mes résultats sur les preuves de connaissance. J'ai d'abord développé une étude sur les preuves permettant de garantir qu'un secret appartient à un intervalle public, étude à laquelle j'ai travaillé avec S.Canard, I.Coisel et J.Traoré qui est actuellement en cours de soumission. J'ai brièvement décrit les paradigmes sous-jacents aux différentes solutions de la littérature, avant d'introduire nos nouvelles solutions. Je me suis ensuite appuyée sur ces descriptions pour comparer les différentes solutions et dégager celles à privilégier selon les en fonction de la taille de l'intervalle considéré, des capacités de calcul du prouveur et de celles du vérifieur.

J'ai ensuite exposé les travaux permettant d'optimiser la vérification des preuves de connaissance dites "Groth-Sahai" que j'ai présentés, avec O.Blazy, G.Fuchsbauer, M.Izabachène, H.Sibert et D.Vergnaud, à ACNS 2010 [BFI+10a]. Notre technique de rassemblement d'équations bilinéaires permet de diminuer de 60 à 90% le nombre d'évaluations de couplages nécessaires à la vérification de ces preuves. Cependant, ce type de preuve nécessite encore de grande capacité de calcul, il faut donc continuer à travailler sur leur optimisation.

Dans un second temps, j'ai retracé l'étude globale que j'ai menée sur les signatures caméléons avec l'aide de S.Canard.

J'ai d'abord exposé le cas simple dans lequel un signataire choisit un tiers, le modifieur, auquel il permet de modifier certaines parties de ses messages signés, de telle façon que les messages modifiés restent assortis à une signature valide du signataire. Après une description du modèle classique pour ce type de signature, j'ai développé une analyse complète de la sécurité des solutions existantes. Enfin, j'ai proposé une solution efficace et sûre dans le modèle classique.

J'ai ensuite intégré quatre types d'extensions s'intéressant aux limitations que le signataire pourrait souhaiter imposer au modifieur.

ValFixées Cette extension permet au signataire de fixer les modifications de certaines parties du message dans un ensemble de valeurs autorisées. Le délégué souhaitant modifier une telle partie doit choisir sa modification dans l'ensemble de valeurs autorisées par le signataire.

ModifEgales Avec cette extension, le signataire impose qu'un ensemble de parties modifiables restent identiques entre elles. Si le délégué souhaite modifier une des parties de l'ensemble, il doit nécessairement modifier les autres de la même manière.

LimNbModif Cette extension permet au signataire de limiter le nombre de parties que le délégué peut modifier parmi l'ensemble des parties modifiables. Si le délégué dépasse cette limite, une fraude est détectée.

LimNbVersion Cette extension permet au signataire d'imposer une limite sur le nombre de versions que le délégué peut générer. Si le délégué dépasse cette limite, alors une fraude est détectée.

J'ai décrit une extension du modèle classique prenant en compte ces limitations, puis j'ai analysé les solutions existantes en fonction de ce modèle étendu. Ce travail a débouché sur un ensemble de nouvelles solutions réparant ou remplaçant les propositions existantes pour de telles signatures caméléons étendues. Ces résultats ont été présentés à la conférence CT-RSA 2010 [CJ10a].

J'ai ensuite exposé mes travaux, effectués conjointement avec S.Canard, sur les signatures de groupes caméléons. Ces signatures caméléons s'intéressent au cas où le signataire souhaite être anonyme dans un groupe de signataires autorisés. À cette occasion, j'ai élaboré une synthèse des propriétés de sécurité souhaitables en fonction de l'objectif visé. L'anonymat du signataire au sein du groupe étant prioritaire, j'ai décrit les différentes déclinaisons possibles pour les autres propriétés et le modèle de sécurité en découlant. J'ai ensuite présenté une solution de signature de groupe non-transparente, c'est-à-dire pour laquelle il est publiquement possible de distinguer une signature originale d'une signature modifiée. Enfin, j'ai esquissé les principes permettant d'obtenir, à partir de cette première solution, une signature de groupe atteignant les autres notions de transparence étudiées. La notion de multi-signature caméléon considérant non seulement un groupe de signataires mais aussi un groupe de modifieurs reste cependant à étudier. Pour cela, il faudra, je pense, sortir de l'utilisation systématique des fonctions de hachage caméléon qui ne permettent de répondre aux questions de responsabilités des modifieurs que par la non-responsabilité des signataires. Les notions de responsabilité individuelle ou de groupe dans ce cas complexe seront d'autre part à réévaluer.

Enfin, dans un troisième temps, j'ai décrit les différentes applications auxquelles je me suis intéressée au cours de ma thèse, l'objectif étant de proposer de nouvelles solutions respectueuses de la vie privée des utilisateurs. J'ai ainsi exposé une solution permettant la facturation de services de telle sorte que le fournisseur de service n'apprenne pas l'identité du consommateur et que le fournisseur de facture ignore le détail des services souscris. Cette solution, élaborée en collaboration avec S.Canard et E.Malville, a débouché sur le dépôt d'un brevet [CJM09].

J'ai ensuite détaillé un ensemble de méthodes d'abonnement qui permettent d'obtenir plusieurs compromis entre l'intraçabilité des utilisateurs et le profilage de ceux-ci par le fournisseur de service. Ces solutions, issues de travaux communs avec S.Canard, ont été brevetées [CJ09] et publiées à la conférence Trustbus 2010 [CJ10b].

Enfin, j'ai exposé une solution de dérivation de clés dans un graphe qui a été présentée à la conférence Indocrypt 2008 [CJ08]. Celle-ci peut être utilisée pour la gestion de contenus dans un groupe hiérarchisé comme j'ai pu le décrire dans le brevet [CJM07] déposé avec S.Canard et M.Milhau.

Les questions de respect de la vie privée des utilisateurs deviennent de plus en plus présentes dans notre actualité, il est donc aujourd'hui essentiel de chercher de nouvelles solutions. Ces nouvelles thématiques proposent de nouveaux défis pour la recherche car elles donnent naissance à de nouveaux paradigmes. Du point de vue industriel, il est plus simple et moins onéreux d'anticiper ces questionnements que de devoir les intégrer a posteriori. D'autant plus que les affaires médiatisées de divulgation d'informations privées ont un effet négatif sur l'image de l'entreprise et donc sur son capital confiance auprès de ses clients. Les entreprises s'investissent ainsi de plus en plus dans la recherche de nouvelles solutions efficaces à ces questions. La recherche

en cryptologie profite de ces objectifs économiques pour soulever de nouveaux défis et s'améliorer constamment.

Mes publications

Articles

[BFI+10b] Batch groth-sahai, avec O.Blazy, G.Fuchsbauer, M.Izabachène, H.Sibert et D.Vergnaud, ACNS 2010.

[CJ10a] On extended sanitizable signature schemes, avec S.Canard, CT-RSA 2010.

[CJ10b] Untraceability and profiling are not mutually exclusive, avec S.Canard, TrustBus 2010.

[CJ08] Group key management : From a non-hierarchical to a hierarchical structure, avec S.Canard, INDOCRYPT 2008.

[CCJT] Methodologies for range proof, avec S. Canard, I. Coisel et J. Traoré, en soumission (Annexe A).

[CJ] How to Protect Customers Privacy in Billing Systems, avec S.Canard, en soumission (Annexe B).

[CJL] Sanitizable Signatures with Several Signers and Sanitizers, avec S.Canard et R.Lescuyer, en soumission.

Brevets

[CJ09] Procédé cryptographique d'abonnement anonyme à un service, avec S.Canard, brevet français, 2009.

[CJM09] Procédé cryptographique d'authentification anonyme et d'identification séparée d'un utilisateur, avec S.Canard et E.Malville, brevet français, 2009.

[CJM07] Procédé de génération d'une clé cryptographique associée à un sous-groupe dans une hiérarchie, programme d'ordinateur et dispositifs associés, avec S.Canard et M.Milhau, brevet français, 2007.

Bibliographie

[ACdT05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS 2005*, volume 3679 of *LNCS*, pages 159–177, Milan, Italy, September 12–14, 2005. Springer, Berlin, Germany.

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 255–270, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Berlin, Germany.

[Ad04] Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In Carlo Blundo and Stelvio Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 165–179, Amalfi, Italy, September 8–10, 2004. Springer, Berlin, Germany.

[ADFD04] J.-P. Andreaux, A. Durand, T. Furon, and E. Diehl. Copy protection system for digital home networks. *Signal Processing Magazine, IEEE*, 21(2) :100 – 108, mar. 2004.

[AFB05] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05*, pages 190–202, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.

[ALNR09] Christophe Arene, Tanja Lange, Michael Naehrig, and Christophe Ritzenthaler. Faster computation of the tate pairing. Cryptology ePrint Archive, Report 2009/155, 2009. `http://eprint.iacr.org/`.

[ASM08] Man Ho Au, Willy Susilo, and Yi Mu. Practical anonymous divisible e-cash from bounded accumulators. In Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 287–301, Cozumel, Mexico, January 28–31, 2008. Springer, Berlin, Germany.

[AT83] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a multilevel security problem. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 237–249, Santa Barbara, CA, USA, 1983. Plenum Press, New York, USA.

[ATSM09] Man Ho Au, Patrick P. Tsang, Willy Susilo, and Yi Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 295–308, San Francisco, CA, USA, April 20–24, 2009. Springer, Berlin, Germany.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of

LNCS, pages 56–73, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2) :149–177, April 2008.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 171–188, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Berlin, Germany.

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 04*, pages 132–145, Washington D.C., USA, October 25–29, 2004. ACM Press.

[BCDv88] Ernest F. Brickell, David Chaum, Ivan Damgård, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 156–166, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Berlin, Germany.

[BCP01] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 290–309, Gold Coast, Australia, December 9–13, 2001. Springer, Berlin, Germany.

[BCP02] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 321–336, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

[BCP07a] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably secure authenticated group diffie-hellman key exchange. *ACM Trans. Inf. Syst. Secur.*, 10(3) :10, 2007.

[BCP07b] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. A security solution for ieee 802.11's ad hoc mode : password-authentication and group diffiehellman key exchange. *IJWMC*, 2(1) :4–13, 2007.

[BCPQ01] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *ACM CCS 01*, pages 255–264, Philadelphia, PA, USA, November 5–8, 2001. ACM Press.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 26–45, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Berlin, Germany.

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 317–336, Irvine, CA, USA, March 18–20, 2009. Springer, Berlin, Germany.

[BFI+10a] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch groth-sahai. In Jianying Zhou and Moti Yung, editors, *ACNS*, volume 6123 of *Lecture Notes in Computer Science*, pages 218–235, 2010.

[BFI+10b] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch groth-sahai. Cryptology ePrint Archive, Report 2010/040, 2010. `http://eprint.iacr.org/`.

[BFP+01] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *20th ACM PODC*, pages 274–283, Newport, Rhode Island, USA, August 26–29, 2001. ACM Press.

[BFPW07] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio, and Bogdan Warinschi. A closer look at PKI : Security and efficiency. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 458–475, Beijing, China, April 16–20, 2007. Springer, Berlin, Germany.

[BG97] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In *ACM CCS 97*, pages 78–91, Zurich, Switzerland, April 1–4, 1997. ACM Press.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341, Cambridge, MA, USA, February 10–12, 2005. Springer, Berlin, Germany.

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 236–250, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[Bla08] Marina Blanton. Online subscriptions with anonymous access. In Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08*, pages 217–227, Tokyo, Japan, March 18–20, 2008. ACM Press.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures : Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629, Warsaw, Poland, May 4–8, 2003. Springer, Berlin, Germany.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 431–444, Bruges, Belgium, May 14–18, 2000. Springer, Berlin, Germany.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical : A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures : The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153, San Francisco, CA, USA, February 14–18, 2005. Springer, Berlin, Germany.

[BZNR01] Jean-camille Birget, Xukai Zou, Guevara Noubir, and Byrav Ramamurthy. Hierarchy-based access control in distributed environments. In *IEEE International Conference on Communications (ICC'01)*, pages 229–233, 2001.

[CCJT] Sébastien Canard, Iwen Coisel, Amandine Jambert, and Jacques Traoré. Methodologies for range proof. en soumission.

[CCS08] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252, Melbourne, Australia, December 7–11, 2008. Springer, Berlin, Germany.

[CDG+09] Sébastien Canard, Cécile Delerablée, Aline Gouget, Emeline Hufschmitt, Fabien Laguillaumie, Hervé Sibert, Jacques Traoré, and Damien Vergnaud. Fair e-cash : Be compact, spend faster. In *ISC 2009*, LNCS, pages 294–309. Springer, Berlin, Germany, 2009.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Berlin, Germany.

[CES+05] Liqun Chen, Matthias Enzmann, Ahmad-Reza Sadeghi, Markus Schneider II, and Michael Steiner. A privacy-protecting coupon system. In Andrew Patrick and Moti Yung, editors, *FC 2005*, volume 3570 of *LNCS*, pages 93–108, Roseau, The Commonwealth Of Dominica, February 28 – March 3, 2005. Springer, Berlin, Germany.

[CFT98] Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 561–575, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218, Dallas, Texas, USA, May 23–26, 1998. ACM Press.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 40–57, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany.

[CGH06] Sébastien Canard, Aline Gouget, and Emeline Hufschmitt. A handy multi-coupon system. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06*, volume 3989 of *LNCS*, pages 66–81, Singapore, June 6–9, 2006. Springer, Berlin, Germany.

[CGHGN01] Dario Catalano, Rosario Gennaro, Nick Howgrave-Graham, and Phong Q. Nguyen. Paillier's cryptosystem revisited. In *ACM CCS 01*, pages 206–214, Philadelphia, PA, USA, November 5–8, 2001. ACM Press.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 103–118, Konstanz, Germany, May 11–15, 1997. Springer, Berlin, Germany.

[CHKO08] Philippe Camacho, Alejandro Hevia, Marcos A. Kiwi, and Roberto Opazo. Strong accumulators from collision-resistant hashing. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC 2008*, volume 5222 of *LNCS*, pages 471–486, Taipei, Taiwan, September 15–18, 2008. Springer, Berlin, Germany.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 302–321, Aarhus, Denmark, May 22–26, 2005. Springer, Berlin, Germany.

[CJ] Sébastien Canard and Amandine Jambert. How to protect customers privacy in billing systems. en soumission.

[CJ08] Sébastien Canard and Amandine Jambert. Group key management : From a non-hierarchical to a hierarchical structure. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT 2008*, volume 5365 of *LNCS*, pages 213–225, Kharagpur, India, December 14–17, 2008. Springer, Berlin, Germany.

[CJ09] Sébastien Canard and Amandine Jambert. Procédé cryptographique d'abonnement anonyme à un service, 2009.

[CJ10a] Sébastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In *CT-RSA 2010*, LNCS, pages 179–194. Springer, Berlin, Germany, 2010.

[CJ10b] Sébastien Canard and Amandine Jambert. Untraceability and profiling are not mutually exclusive. In Sokratis K. Katsikas, Javier Lopez, and Miguel Soriano, editors, *TrustBus*, volume 6264 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2010.

[CJL] Sébastien Canard, Amandine Jambert, and Roch Lescuyer. Sanitizable signatures with several signers and sanitizers. en soumission.

[CJM07] Sébastien Canard, Amandine Jambert, and Michel Milhau. Procédé de génération d'une clé cryptographique associée à un sous-groupe dans une hiérarchie, programme d'ordinateur et dispositifs associés, 2007.

[CJM09] Sébastien Canard, Amandine Jambert, and Éric Malville. Procédé cryptographique d'authentification anonyme et d'identification séparée d'un utilisateur, 2009.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 481–500, Irvine, CA, USA, March 18–20, 2009. Springer, Berlin, Germany.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289, Amalfi, Italy, September 12–13, 2002. Springer, Berlin, Germany.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Berlin, Germany.

[CLL04] Jue-Sam Chou, Chu-Hsing Lin, and Ting-Ying Lee. A novel hierarchical key management scheme based on quadratic residues. In Jiannong Cao, Laurence Tianruo Yang, Minyi Guo, and Francis Chi-Moon Lau, editors, *ISPA*, volume 3358 of *Lecture Notes in Computer Science*, pages 858–865. Springer, 2004.

[CLM08] Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoorsanitizable signatures and their application to content protection. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS 08*, volume 5037 of *LNCS*, pages 258–276, New York, NY, USA, June 3–6, 2008. Springer, Berlin, Germany.

[CLTW04] Chin-Chen Chang, Iuon-Chang Lin, Hui-Min Tsai, and Hsiao-Hsi Wang. A key assignment scheme for controlling access in partially ordered user hierarchies. In *AINA (2)*, pages 376–379. IEEE Computer Society, 2004.

[CLX09] Ee-Chien Chang, Chee Liang Lim, and Jia Xu. Short redactable signatures using random trees. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 133–147, San Francisco, CA, USA, April 20–24, 2009. Springer, Berlin, Germany.

[CM06] Benoit Chevallier-Mames. *Cryptographie à clé publique : Constructions et preuves de sécurité*. PhD thesis, Université Paris VII - Denis Diderot, 2006.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Berlin, Germany.

[CSST06] Sébastien Canard, Berry Schoenmakers, Martijn Stam, and Jacques Traoré. List signature schemes. *Discrete Applied Mathematics*, 154(2) :189–201, 2006.

[CT90] Gerald C. Chick and Stafford E. Tavares. Flexible access control with master keys. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 316–322, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Berlin, Germany.

[CT03] Sébastien Canard and Jacques Traoré. List signature schemes and application to electronic voting. In *Workshop on Coding and Cryptography (WCC)*, pages 81–90, 2003.

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265, Brighton, UK, April 8–11, 1991. Springer, Berlin, Germany.

[CZK04] Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. Chameleon hashing without key exposure. In Kan Zhang and Yuliang Zheng, editors, *ISC 2004*, volume 3225 of *LNCS*, pages 87–98, Palo Alto, CA, USA, September 27–29, 2004. Springer, Berlin, Germany.

[DH76] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, volume 45 of *AFIPS Conference Proceedings*, pages 109–112. AFIPS Press, 1976.

[Dij59] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 :269–271, 1959.

[DLN] Digital living network alliance. www.dlna.org.

[DP06] Cécile Delerablée and David Pointcheval. Dynamic fully anonymous short group signatures. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06*, volume 4341 of *LNCS*, pages 193–210, Hanoi, Vietnam, September 25–28, 2006. Springer, Berlin, Germany.

[Dri53] Étienne Drioton. Les principes de la cryptographie égyptienne. *Comptes-rendus des séances de l'Académie des inscriptions et belles-lettres*, 97(3) :355–364, 1953.

[DSD07] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing cryptographic pairings over Barreto-Naehrig curves (invited talk). In Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, editors, *PAIRING 2007*, volume 4575 of *LNCS*, pages 197–207, Tokyo, Japan, July 2–4, 2007. Springer, Berlin, Germany.

[DSGP05] Manik Lal Das, Ashutosh Saxena, Ved Prakash Gulati, and Deepak B. Phatak. Hierarchical key management scheme using polynomial interpolation. *Operating Systems Review*, 39(1) :40–47, 2005.

[DVB] Dvb content protection & copy management. http ://www.dvb.org/technology/dvb-cpcm/.

[ECR08] ECRYPT II. eBACS : ECRYPT benchmarking of cryptographic systems. http ://bench.cr.yp.to/index.html, 2008.

[EGW09] N.P. Smart E. Ghadafi and B. Warinschi. Groth–sahai proofs revisited. Cryptology ePrint Archive, Report 2009/599, 2009. `http://eprint.iacr.org/`.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1985. Springer, Berlin, Germany.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2) :77–94, 1988.

[FGHOP09] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Ostergaard Pedersen. Practical short signature batch verification. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 309–324, San Francisco, CA, USA, April 20–24, 2009. Springer, Berlin, Germany.

[FI05] Jun Furukawa and Hideki Imai. An efficient group signature scheme from bilinear maps. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP 05*, volume 3574 of *LNCS*, pages 455–467, Brisbane, Queensland, Australia, July 4–6, 2005. Springer, Berlin, Germany.

[Fis01] Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 457–472, San Francisco, CA, USA, April 8–12, 2001. Springer, Berlin, Germany.

[FO99a] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In Hideki Imai and Yuliang Zheng, editors, *PKC'99*, volume 1560 of *LNCS*, pages 53–68, Kamakura, Japan, March 1–3, 1999. Springer, Berlin, Germany.

[FO99b] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Germany.

[FP01] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 351–368, Gold Coast, Australia, December 9–13, 2001. Springer, Berlin, Germany.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself : Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Berlin, Germany.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90 : Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 416–426, New York, NY, USA, 1990. ACM.

[GHK+08] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, Tal Rabin, Steffen Reidt, and Stephen D. Wolthusen. Strongly-resilient and non-interactive hierarchical key-

agreement in MANETs. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 49–65, Málaga, Spain, October 6–8, 2008. Springer, Berlin, Germany.

[Gir91] Marc Girault. Self-certified public keys. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 490–497, Brighton, UK, April 8–11, 1991. Springer, Berlin, Germany.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1) :169–192, 1996.

[GMR85a] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

[GMR85b] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical" solution to the signature problem (abstract) (impromptu talk). In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, page 467, Santa Barbara, CA, USA, August 19–23, 1985. Springer, Berlin, Germany.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1) :186–208, 1989.

[Gro05] Jens Groth. Non-interactive zero-knowledge arguments for voting. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05*, volume 3531 of *LNCS*, pages 467–482, New York, NY, USA, June 7–10, 2005. Springer, Berlin, Germany.

[Gro07] Jens Groth. Fully anonymous group signatures without random oracles. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 164–180, Kuching, Malaysia, December 2–6, 2007. Springer, Berlin, Germany.

[GS07] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. Cryptology ePrint Archive, Report 2007/155, 2007. `http://eprint.iacr.org/`.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[HFKY03] Mingxing He, Pingzhi Fan, Firoz Kaderali, and Ding Yuan. Access key distribution scheme for level-based hierarchy. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies PDCAT'2003*, pages 942–945, 2003.

[HZS93] Thomas Hardjono, Yuliang Zheng, and Jennifer Seberry. New solutions to the problem of access control in a hierarchy. Technical Report Preprint 93-2, 1993.

[ITW82] Ingemar Ingemarsson, Donald T. Tang, and C. K. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5) :714–719, 1982.

[Kah96] David Kahn. *The Codebreakers, The Story of Secret Writing*. Scribner, 2nd revised edition (6 octobre 1997) edition, 1996.

[KKLV05] Paul Koster, Frank Kamperman, Peter Lenoir, and Koen Vrielink. Identity based drm : Personal entertainment domain. In Jana Dittmann, Stefan Katzenbeisser, and Andreas Uhl, editors, *Communications and Multimedia Security, 9th IFIP TC-6 TC-11 International Conference, CMS 2005, Salzburg, Austria, September 19-21, 2005, Proceedings*, volume 3677 of *Lecture Notes in Computer Science*, pages 42–54, 2005.

[KL06] Marek Klonowski and Anna Lauks. Extended sanitizable signatures. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC 06*, volume 4296 of *LNCS*, pages 343–355, Busan, Korea, November 30 – December 1, 2006. Springer, Berlin, Germany.

[KPT04a] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Group key agreement efficient in communication. *IEEE Trans. Computers*, 53(7) :905–921, 2004.

[KPT04b] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Tree-based group key agreement. *ACM Trans. Inf. Syst. Secur.*, 7(1) :60–96, 2004.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS 2000*, San Diego, California, USA, February 2–4, 2000. The Internet Society.

[KSCL99] F.H. Kuo, V.R.L. Shen, T.S. Chen, and F. Lai. Cryptographic key assignment scheme for dynamic access control in a user hierarchy. In *IEE Proceedings Computers and Digital Techniques*, volume 146, pages 235–240, 1999.

[LAN02] Helger Lipmaa, N. Asokan, and Valtteri Niemi. Secure Vickrey auctions without threshold trust. In Matt Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 87–101, Southampton, Bermuda, March 11–14, 2002. Springer, Berlin, Germany.

[Lin01] Chu-Hsing Lin. Hierarchical key assignment without public-key cryptography. *Computers & Security*, 20(7) :612–619, 2001.

[Lip03] Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 398–415, Taipei, Taiwan, November 30 – December 4, 2003. Springer, Berlin, Germany.

[LKKR03] Sangwon Lee, Yongdae Kim, Kwangjo Kim, and Dae-Hyun Ryu. An efficient tree-based group key agreement using bilinear map. In Jianying Zhou, Moti Yung, and Yongfei Han, editors, *ACNS 03*, volume 2846 of *LNCS*, pages 357–371, Kunming, China, October 16–19, 2003. Springer, Berlin, Germany.

[Lyn07] Ben Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, 2007.

[Mas96] Mastercard and VIsa. Secure electronic transaction (set), 1996.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426, Santa Barbara, CA, USA, August 18–22, 1986. Springer, Berlin, Germany.

[Möl01] Bodo Möller. Algorithms for multi-exponentiation. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001*, volume 2259 of *LNCS*, pages 165–180, Toronto, Ontario, Canada, August 16–17, 2001. Springer, Berlin, Germany.

[MTMA85] Stephen J. MacKinnon, Peter D. Taylor, Henk Meijer, and Selim G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Trans. Computers*, 34(9) :797–802, 1985.

[MVO91] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC '91 : Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 80–89, New York, NY, USA, 1991. ACM.

[MvV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

[NSN04] Lan Nguyen and Reihaneh Safavi-Naini. Efficient and provably secure trapdoor-free group signature schemes from bilinear pairings. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 372–386, Jeju Island, Korea, December 5–9, 2004. Springer, Berlin, Germany.

[OMA] Open mobile alliance. www.openmobilealliance.org.

[OP01] Tatsuaki Okamoto and David Pointcheval. REACT : Rapid Enhanced-security Asymmetric Cryptosystem Transform. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 159–175, San Francisco, CA, USA, April 8–12, 2001. Springer, Berlin, Germany.

[PCTK04] Bogdan C. Popescu, Bruno Crispo, Andrew S. Tanenbaum, and Frank L.A.J. Kamperman. A drm security architecture for home networks. In *DRM '04 : Proceedings of the 4th ACM workshop on Digital rights management*, pages 1–10, New York, NY, USA, 2004. ACM.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Berlin, Germany.

[Poi00] David Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000*, volume 1751 of *LNCS*, pages 129–146, Melbourne, Victoria, Australia, January 18–20, 2000. Springer, Berlin, Germany.

[PS98] Guillaume Poupard and Jacques Stern. Security analysis of a practical "on the fly" authentication and signature generation. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 422–436, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3) :361–396, 2000.

[RRN02] Indrakshi Ray, Indrajit Ray, and Natu Narasimhamurthi. A cryptographic solution to implement access control in a hierarchy and more. In *SACMAT '02 : Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 65–73, New York, NY, USA, 2002. ACM.

[RS85] J.O. Rabin and J. Shallit. Randomized algorithms in number theory. Technical report, University of Chicago, 1985.

[SBZ01] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In Kwangjo Kim, editor, *ICISC 01*, volume 2288 of *LNCS*, pages 285–304, Seoul, Korea, December 6–7, 2001. Springer, Berlin, Germany.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Berlin, Germany.

[Sch01] Berry Schoenmakers. Some efficient zero-knowledge proof techniques. In *Workshop on Cryptographic Protocols*, 2001.

[Sch05] Berry Schoenmakers. Interval proofs revisited. In *Workshop on Frontiers in Electronic Elections*, 2005.

[SFM04] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Cryptographic key assignment schemes for any access control policy. *Inf. Process. Lett.*, 92(4) :199–205, 2004.

[Sho04] Victor Shoup. Sequences of games : a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[STW00] Michael Steiner, Gene Tsudik, and Michael Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8) :769–780, August 2000.

[Tar09] Christophe Tartary. Ensuring authentication of digital information using cryptographic accumulators. In *CANS 09*, LNCS, pages 315–333. Springer, Berlin, Germany, 2009.

[TS06] Isamu Teranishi and Kazue Sako. k-times anonymous authentication with a constant proving cost. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 525–542, New York, NY, USA, April 24–26, 2006. Springer, Berlin, Germany.

[WHA] W-ha. www.w-ha.com.

[WW05] Jiang Wu and Ruizhong Wei. An access control scheme for partially ordered set hierarchy with provable security. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 221–232. Springer, 2005.

[YHM+09] Tsz Hon Yuen, Qiong Huang, Yi Mu, Willy Susilo, Duncan S. Wong, and Guomin Yang. Efficient non-interactive range proof. In *COCOON '09 : Proceedings of the 15th Annual International Conference on Computing and Combinatorics*, pages 138–147, Berlin, Heidelberg, 2009. Springer-Verlag.

[Zho02] Sheng Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8) :750–759, 2002.

[ZHP91] Yuliang Zheng, Thomas Hardjono, and Josef Pieprzyk. Sibling intractable function families and their applications (extended abstract). In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 124–138, Fujiyoshida, Japan, November 11–14, 1991. Springer, Berlin, Germany.

[ZKB07] Xukai Zou, Yogesh Karandikar, and Elisa Bertino. A dynamic key management solution to access hierarchy. *Int. Journal of Network Management*, 17(6) :437–450, 2007.

[ZW04] Q. Zhang and Y. Wang. A centralized key management scheme for hierarchical access control. In *Global Telecommunications Conference, GLOBECOM '04. IEEE*, volume 4, pages 2067–2071, 2004.

Annexe A

Methodologies for range proof

Methodologies for Range Proof

S.Canard, I.Coisel, A.Jambert and J.Traoré

Abstract. Zero-knowledge proofs of knowledge are now used in many applications and permit to prove the knowledge of secrets with many (complex) properties. Among them, the proof that a secret lies in a given interval is very useful in the context of electronic voting, e-cash or for anonymous credentials. It currently exists several techniques to prove that one secret belongs to an interval of type $[a, b]$. It is for example possible to use the decomposition in squares from Boudot, Lipmaa or Groth. Lipmaa, Asokan and Niemi have proposed a multi-base decomposition technique while Camenisch, Chaabouni and Shelat use a signature-based one. Thus, it seems hard to know which method is the best to use, depending on the size in bits of the secret or the size of the studied interval. In this paper, we first provide several improvements to the multi-base decomposition of the secret. We also introduce a new signature-based method, which does not ask the prover to compute pairings. We finally make the first complete comparison between all existing range proofs, showing that our methods are useful in many practical cases.
Keywords. Range proof, set membership proof, binary decomposition, signature based.

1 Introduction

Zero-knowledge proofs of knowledge (ZKPK) are largely used to *e.g.* prove the possession of some secret corresponding to a discrete logarithm [45, 28] or an e-th root [30]. It moreover exists some generic constructions which are now proven to be secure [33, 16, 10], taking as a basis the work on zero-knowledge proofs for proving the knowledge of a representation [40] or the equality of secrets [19, 15]. We also find in the literature proofs of knowledge of a secret lying in a given interval, *a.k.a.* range proofs, in which we focus in this paper.

Range proofs are useful in many cryptographic applications such as e-cash of multi-coupon [12, 17], where it is necessary to prove that the counter j of already spent coins is not greater than the number J of withdrawn coins. They are also used in the context of electronic voting [32, 2, 21], where the voter needs to prove that her secret vote corresponds to a valid choice in the set of all possible candidates, numbered from 1 to n. It is also very useful for anonymous credential systems [13] where a user may have *e.g.* to prove that her age is greater than a given authorized one, without revealing her true age.

1.1 Related Work

The first proposals [9, 18] for a ZKPK of a secret lying in a specific interval were very efficient but the price to pay was that they only prove the membership to a much larger interval than expected (expansion rate greater than 1, as defined in [8]). Nowadays, there exists several exact range proof methods (with an expansion rate equal to 1).

The first family is due to Bellare and Goldwasser [3]. It uses the binary decomposition of the secret and compares bit-by-bit the secret with each bound of the interval. The drawback of this method is that the space and time efficiencies depend on the size of the secret and that the method of Bellare et al. only describes a proof for intervals of the form $[0, 2^k[$. Schoenmakers [46, 47] has proposed a solution to this latter drawback by giving a general method which permits to make a proof that some secret x lies in some interval $[a, b]$ by using twice a range proof in $[0, 2^k[$.

Lipmaa, Asokan and Niemi [36] have presented a similar range proof for an interval of the form $[0, v]$, based on the work from Damgård and Jurik [23], which is very efficient in the case of a group of prime order. This is based on what can be called the multi-base decomposition of the secret in some base v, which is a generalization of the above binary decomposition. Again, in this method, the space and time complexities depend on the size of the secret.

The second type of method is due to Boudot [8] and uses some mathematical properties of positive integers such as the decomposition of such integer in a sum of squares. This method was later refined by Lipmaa [35] and next by Groth [29] to obtain a very efficient exact method where the time and space complexities are independent of the size of the secret. However, these methods need a group of unknown order, and thus the manipulation of bigger variables, which makes them less efficient for small secrets.

More recently, Camenisch, Chaabouni and Shelat [11] have proposed a new way to treat range proofs, based on an initial work from Teranishi and Sako [48]. They first propose a set membership protocol, which permits to prove that one secret value belongs to a set Φ, and next design a new range proof. The main idea is that a designated authority produces public signatures on each element of the set Φ (resp. interval $[a, b]$). The proof of knowledge of *e.g.* a secret $x \in [a, b]$ next consists in proving the knowledge of a (public) signature on the secret x (which is possible only if x belongs to the interval) without revealing x nor the signature. Camenisch, Chaabouni and Shelat [11] also refine this method by using the u-ary representation of the secret x, and thus prove that each digit of x in base u is in the interval $[0, u[$, which implies to publish less signatures than the initial method. The prover's time complexity is very efficient, except that she has to compute some pairing evaluations. The main problem with this method is the size of the public key and the time complexity of the SETUP phase, which both depend on the number of integers in the interval $[a, b]$.

1.2 Our Contributions

In this paper, we first improve the range proof methods which belong to the first family. Considering that a prover wants to prove that her secret x belongs to the public interval $[a, b]$, we use the multi-base decomposition of a and x in base b, which gives us, following [36], that $x \leq b$. We next compare a and x by using a generalization of a characterization due to Fischlin [25]. This one says that if $a = [\![a_0, \cdots, a_\ell]\!]_b$ and $x = [\![x_0, \cdots, x_\ell]\!]_b$ are two integers where the a_i's and the x_i's are the multi-base decomposition of a and x in base b respectively, then $x > a$ if and only if $\exists i' \in [0, \ell - 1]/a_{i'} = 0$, $x_{i'} = 1$ and $\forall j > i', a_j = x_j$. We also use the fact that both the prover and the verifier know the value a and its multi-base decomposition in base b. Next, from the commitment on all the x_i's, if the most significant digits of a are equal to 1, then this is necessarily the case for the corresponding digits of the secret x. Thus, without compromising the secrecy of x, we can reveal these digits by simply opening the corresponding commitments. We finally use some boolean logic results to obtain one of the most efficient range proof method.

We next give a new signature-based set membership proof which is very relevant in the case of electronic voting [32, 2, 21] where the (set of) verifier(s) owns a secret decryption key to open ballots. Our new solution, contrary to the initial one from Camenisch, Chaabouni and Shelat [11], does not require to compute pairings, and is thus more efficient.

We finally make the first complete study on all existing methods to prove that a secret lies in a given interval. We thus compare the prover and verifier times complexities, the space complexity, the size of the public key and the time complexity of the SETUP phase. We then show that it is currently not possible to say that one solution is always the best one and prove that our new methods are very interesting in many practical cases. Such study has already been done in [11] but they do not consider the Lipmaa-Asokan-Niemi method, and they do not compare all characteristics of range proofs.

1.3 Organization of the Paper

This paper is organized as follows. Section 2 recalls the existing characterizations that a value lies into a given interval. In Sections 3 and 4, we respectively introduce our new signature-based and multi-base decomposition based range proofs. Before concluding, we make our complete efficiency analysis of all existing and new range proofs in Section 5. We also formally describe existing methods and give the details of our complexity study in the appendices.

2 Characterizations for Range Proofs

In this section, we review several characterizations that a value x belongs to an interval of the form $[a, b]$. In the following, the negation of a predicate $\mathtt{p}$ is denoted $\neg\mathtt{p}$, the "or" operator is denoted $\vee$ and the "and" one is denoted $\wedge$.

2.1 Introduction to our Study

In the following, we consider, except when explicitly mentioned, that we are working on an elliptic curve, and we use the multiplicative notation. More precisely, let p be a prime integer and let $E/\mathbb{F}_p$ be an elliptic curve over $\mathbb{F}_p$. Let q be a prime divisor of the group order and let E have embedding degree

k with respect to q. We study a secret integer $x \in \mathbb{Z}_q^*$ which belongs to a public interval $[a, b]$ where $a, b \in \mathbb{Z}_q^*$, with $a < b$. We consider that this secret is committed as $C = g^x h^r$, using the Pedersen commitment [41].

It is also possible to replace this commitment by a ciphertext on x, which is the case in some applications, such as electronic voting [2, 21]. Note that our new methods can easily be described in this setting, even if we do not always detail it.

Our aim in this paper is to construct a non-interactive zero-knowledge proof of knowledge, using the Fiat-Shamir heuristic [24], that the committed value x lies in the given public interval $[a, b]$. In this section, we describe several characterizations that an integer $x \in [a, b]$ which can be used to describe such zero-knowledge proof.

Note that, in this paper, range proofs are described as a non-interactive protocol between a prover knowing x and a verifier. As there are related to an interactive version, our new methods are also relevant in this case.

2.2 The Decomposition in the Sum of Squares

In [8], Boudot uses the fact that $a \leq x \leq b$ iff $x - a \geq 0$ and $b - x \geq 0$ and he tries to find a characterization that a number is greater or equal to 0. In fact, he proposes two solutions, depending on the chosen characterization.

Lemma 1. *Let X and L be two positive integers. Then $0 \leq X < L$ if and only if $\exists (x_1, x_2) \in \mathbb{Z}^2 : X = x_1^2 + x_2$ and $0 \leq x_2 < 2\sqrt{L}$.*

Boudot has also studied the following characterization, but the related zero-knowledge proof has been completely described by Lipmaa in [35].

Lemma 2 (Lagrange theorem). *Let X be an integer. Then $X \geq 0$ if and only if it exists $(x_1, x_2, x_3, x_4) \in \mathbb{Z}^4 : X = x_1^2 + x_2^2 + x_3^2 + x_4^2$.*

Finally, Groth has proposed in [29] a variant in a special case for the secret x.

Lemma 3. *Let X be an integer which is not of the form $4^n(8k + 7)$, where $n, k \in \mathbb{N}$. Then $X \geq 0$ if and only if $\exists (x_1, x_2, x_3) \in \mathbb{Z}^3 : X = x_1^2 + x_2^2 + x_3^2$.*

Note that the Rabin and Shallit algorithm [43] permits to find such decomposition in a sum of three or four squares. The zero-knowledge proof is next composed of (i) the proof of knowledge of the different values ((x_1, x_2), (x_1, x_2, x_3, x_4) or (x_1, x_2, x_3)) and (ii) the proof that, using these values, we can reconstruct the committed secret value (see Appendix J for details).

2.3 The u-ary Decomposition

It is a well-known result that an integer can be decomposed in any base u. We thus have the following lemma, which is quite obvious.

Lemma 4 (u-ary decomposition). *Let x and u be positive integers. Then, $x \in [0, u^\ell[$ if and only if $x = \sum_{i=0}^{\ell-1} x_i u^i$ where $\forall i \in [0, \ell[, x_i \in [0, u[$. In this case, we write $x = [x_0, \ldots, x_{\ell-1}]_u$.*

In cryptography, the most used one is the binary decomposition, that is when $u = 2$. Thus, using the above lemma with $u = 2$, we have a characterization for any integer $x \in [0, 2^\ell[$.

2.4 The Double u-ary Representation

We now describe a characterization for the predicate $x \in [a, b]$ which is due to Schoenmakers [46, 47] for the binary case, while the u-ary case has been described in [11]. More precisely, we have the following lemma.

Lemma 5. *Let a, b, x be three integers in $\mathbb{Z}_q^*$, and u a positive integer. We denote $B = b - a + 1$ and $X = x - a$. Let k be the unique integer such that $u^k \leq B \leq u^{k+1}$. We denote $B_0 = u^{k+1} - B$ and $Y = x - a + B_0$. Then $x \in [a, b]$ if and only if $X \in [0, u^{k+1}[$ and $Y \in [0, u^{k+1}[$.*

Proof. $x \in [a,b]$ can also be written $X \in [0,B[$. As $u^k \leq B \leq u^{k+1}$ and $B_0 = u^{k+1} - B$, this is equivalent to say that $X \in [0,u^{k+1}[\cap]-B_0, B[$. As $B + B_0 = u^{k+1}$, we can rewrite this equation as $(X \in [0,u^{k+1}[) \wedge (X + B_0 \in [0,u^{k+1}[)$ which is equivalent to $(X \in [0,u^{k+1}[) \wedge (Y \in [0,u^{k+1}[)$, which concludes the proof. □

In fact, it is now possible to prove that $x \in [a,b]$ by only using twice the proof that a secret lies in an interval of the form $[0,u^\ell[$ (see Appendix H for details).

2.5 Signature Based Characterization

The idea of using the signatures of all the integers of the interval is due to Teranishi and Sako [48] and next used in [11] by Camenisch, Chaabouni and Shelat. More precisely, we have the following characterization.

Lemma 6. *Let a, b, x be three integers. For all $k \in [a,b]$, let $\sigma_k = \textsc{Sign}(k)$ be a signature on the value k using the secret key of a designated authority. Let Σ be the set of all σ_k. Then, $x \in [a,b]$ if and only if $\exists \sigma \in \Sigma$ such that $\sigma = \textsc{Sign}(x)$.*

The authors of [48, 11] propose to use Boneh-Boyen short signatures [7]. These signatures permit to prove the knowledge of a message and a signature on this message without revealing the message nor the signature. Camenisch, Chaabouni and Shelat [11] also use the u-ary representation of x (in the general case) to obtain better efficiency (see Appendix K for details).

We introduce in Section 3 a new signature-based range proof, based on the work in [11], which does not ask the prover to compute pairings.

2.6 Multi-Base Decomposition

In [36], Lipmaa, Asokan and Niemi use the following characterization to prove that one secret is in a given interval $[0,v]$. We call this characterization the *multi-base decomposition*.

Lemma 7 (Multi-base decomposition). *Let v and x be integers in $\mathbb{Z}_q^*$ and let $\ell = \lfloor \log_2 v \rfloor$. Then, $x \in [0,v]$ if and only if $x = \sum_{i=0}^{\ell} v_i x_i$ where for all $i \in [0,\ell]$, $x_i \in \{0,1\}$ and $v_i = \lfloor (v + 2^{\ell-i})/2^{\ell-i+1} \rfloor$. In this case, we write $x = [\![x_0, \ldots, x_\ell]\!]_v$.*

This is a generalization of the binary decomposition given by Lemma 4. If one wants to prove that $x \in [0,2^k[$, then, using the above lemma, $v = 2^k - 1$ and thus, for all $i \in [0,k-1]$, $v_i = 2^{k-i-1}$. This is exactly the binary decomposition of x. We consequently have $x = [\![x_0, \ldots, x_\ell]\!]_{2^k-1} = [x_0, \ldots, x_{k-1}]_2$. In fact, this is possible to prove Lemma 7 for any v and x, using the binary decomposition of v. We do not detail this proof in the paper.

This lemma can next be used to design a zero-knowledge proof that $x \in [0,v]$ by committing each x_i and next using the public v_i to prove the relation $x = \sum_{i=0}^{\ell} v_i x_i$ (see Section 4.1).

In the following, we also need the following result, which is a direct consequence of lemma 7.

Lemma 8. $v = [\![1, \ldots, 1]\!]_v$

Proof. We prove this lemma by contradiction. We consider that $v \neq [\![1, \ldots, 1]\!]_v$. We denote $u = [\![1, \ldots, 1]\!]_v \in [0,v[$ (as $u \neq v$). We first remark that $v \in [0,v]$ and thus, it exists $\tilde{v}_0, \ldots, \tilde{v}_\ell$ such that $v = [\![\tilde{v}_0, \ldots, \tilde{v}_\ell]\!]_v$ and $\exists i_0 \in [0,\ell] / \tilde{v}_{i_0} = 0$. Thus, it is necessary that $u = \sum_{i=0}^{\ell} v_i > \sum_{i=0}^{\ell} v_i \tilde{v}_i = v$. This contradicts the fact that $u \in [0,v[$, which concludes the proof. □

Uniqueness of the decomposition. For some value v, the decomposition of an integer x is not unique. For example, for $v = 10$, we have $v_3 = 5, v_2 = 3, v_1 = 1$ and $v_0 = 1$. Then, the integer $x = 6$ can be written $[\![0,1,0,1]\!]_v$ or $[\![1,0,0,1]\!]_v$. In the following, some results are based on the uniqueness of the decomposition. For this reason, we define a decomposition algorithm (Algorithm 1) which outputs a unique decomposition for a given integer. As a consequence, we always consider that this algorithm is used and, given a base v and an integer x, we thus assume the uniqueness of the multi-base decomposition of x in base v.

Algorithm 1: $\text{MBDEC}(x, v_0, \ldots, v_\ell)$

```
m = x;
for i = ℓ to 0 step −1 do
    if m ≥ v_i then
        x_i := 1;
        m := m − v_i
    end
    else x_i := 0
end
return x = [[x_0, ..., x_ℓ]]_v;
```

General case. It is also possible to use the same kind of method to prove that x belongs to a given interval of the form $[a, b]$, as explained in [36]. This is done by using the following lemma. The complete ZKPK is given in Appendix I.

Lemma 9 (General case for multi-base decomposition). *Let a, b and x be integers in $\mathbb{Z}_p^*$ and let $\ell = \lfloor \log_2 b \rfloor$. Then, $x \in [a, b]$ if and only if $x = \sum_{i=-1}^{\ell} b_i x_i$ where for all $i \in [-1, \ell]$, $x_i \in \{0, 1\}$, $b_{-1} = a$ and for all $i \in [0, \ell]$ $b_i = \lfloor (b - a + 2^{\ell - i})/2^{\ell - i + 1} \rfloor$.*

2.7 The Bit by Bit Characterization

The next characterization is due to Fischlin [25]. We here present it in the case of the multi-base decomposition of two different values, while Fischlin uses the binary representation. The idea is to compare these two values by using their representations. More precisely, we have the following lemma.

Lemma 10 (Bit-by-bit Lemma). *Let v, $a = [[a_0, \cdots, a_\ell]]_v$ and $x = [[x_0, \cdots, x_\ell]]_v$ be integers in $\mathbb{Z}_q^*$. Then, $a < x$ if and only if $\exists i' \in [0, \ell] / a_{i'} = 0$, $x_{i'} = 1$ and $\forall j > i', a_j = x_j$.*

The proof of this lemma is relatively straightforward and is not detailed here.

3 Our New Signature-based Set Membership Proof

In this section, we present a new set membership protocol that bears some similarities with the protocol proposed by Camenisch *et al.* [11]. However our set membership proof does not use pairings and is thus computationally more efficient, as we will see in Section 3.4. Our protocol is particularly suited for the context of e-voting where the voter has to prove that his choice belongs to the set of valid candidates and the verification (the tallying phase) is performed by several verifiers (talliers).

3.1 Overview and Main Building Block

Let $\mathbb{G}$ be a group of prime order q and let Φ be a discrete set. Let $s \in \Phi$. The basic idea is that the verifier $\mathcal{V}$ first sends the prover a signature of every element in the set Φ. Thus, the prover $\mathcal{P}$ picks a signature σ on the particular element s to which Com is a commitment. The prover then "encrypts" this received signature and performs a proof of well-formedness. The verifiers (a threshold of them) then use their private keys to check whether the ciphertexts encrypt a valid signature (but without decrypting the corresponding ciphertexts).

Boneh-Boyen signature without pairings. Our signature-based protocol relies on a variant of the Boneh-Boyen signature scheme (see [7] and Appendix A.3) but without bilinear maps.

Let $\mathbb{G}$ be a cyclic group with prime order q where the Decision Diffie-Hellman (DDH) problem is assumed to be hard and g and g_1 two random generators of $\mathbb{G}$. The signer's private key is $y \in \mathbb{Z}_q^*$ and the corresponding public key is $Y = g^y$. The signature on a message m is $A = g_1^{\frac{1}{y+m}}$ (notice that this implies that $A^y = g_1 A^{-m}$). Since we work in a group $\mathbb{G}$ not equipped with a bilinear map, the signer S has to prove that the signature on m is valid as follows: S generates a ZKPK Π that the discrete logarithm of $(g_1 A^{-m})$ in the base A is equal to the discrete logarithm of Y in the base g using the discrete log equality test due to Chaum and Pedersen [19]. The signature on m is valid iff the proof Π is valid. This variant of the Boneh-Boyen signature scheme can be proven secure, in the random oracle model, against an existential forgery under a weak chosen message attack under the $\mathfrak{q}$-Strong Diffie-Hellman assumption, as shown in Appendix A.3.

3.2 The Protocol in Details

We now present a detailed description of our protocol, considering the building blocks presented in Appendix A.

Participants and Notation. We first introduce the participant and our notation.

- The prover is identified by $\mathcal{P}$. She picks a valid signature A on the particular element s to which $Com = g_2^s g_3^f$ is a commitment (where $f \in_R \mathbb{Z}_q$).
- The verifiers are composed of a set of verifiers denoted by $\mathcal{V}$ (they can represent for example the set of talliers in a voting scheme[1]). They share an El Gamal private key $\widehat{V}$ corresponding to a public key V. They also share the private key y associated to the Boneh-Boyen signature public key Y.

In addition to the notation above, we use the following one: $E_V[m]$ is an El Gamal encryption of a message m computed with the public key V, and $D_{\widehat{V}}[m]$ is an El Gamal decryption of m computed with the private key $\widehat{V}$.

Setup of the public parameters. This phase takes place prior the set membership protocol. In order to establish these parameters, first a cyclic group $\mathbb{G}$ with prime order q is defined. The decision Diffie-Hellman must be hard in this group. After that, the verifiers produce four generators g, $g_1, g_2, g_3 \in \mathbb{G}$. The verifiers $\mathcal{V}$ now collaborate to generate the public key V and its corresponding private key $\widehat{V}$. The resulting key $\widehat{V}$ is not known by the verifiers individually. Each verifier $\mathcal{V}_i$ knows only a share v_i of this key. Their corresponding public key is denoted by $V_i = g^{v_i}$. The verifiers $\mathcal{V}$ also cooperate to establish a public key $Y = g^y$ and the corresponding shared private key y. Each verifier $\mathcal{V}_i$ only knows a share y_i of this key. Their corresponding public key is denoted by $Y_i = g^{y_i}$.

After generating their keys, the verifiers are ready to issue the signatures of every element in the set Φ. The verifiers might generate the signatures in a threshold fashion by means of a scheme similar to [49].

The signature-based set membership protocol. The prover picks a valid signature A on the particular element s to which she is committed to: $A = g_1^{\frac{1}{y+s}}$. She first selects a random $r \in \mathbb{Z}_q$ and computes $B = A^r$. After that, she computes the tuple: $\langle B, E_V[B^{r^{-1}}], E_V[B^{sr^{-1}}], \Pi\rangle$ which is equal to $\langle B, E_V[A], E_V[A^s], \Pi\rangle = \langle B, C, D, \Pi\rangle$. The voter then sends this tuple to the verifier along with Com.

The tuple is composed of the ciphertexts $\langle E_V[B^{r^{-1}}], E_V[B^{sr^{-1}}]\rangle$ that correspond to the "encryption" of the signature $A = g_1^{\frac{1}{y+s}}$ on the message s. In addition, it has a set of non-interactive zero-knowledge proofs Π. This set contains the following elements.

- (Π_1) A proof that the prover knows the plaintext related to the ciphertext $C = E_V[B^{r^{-1}}] = (C_1, C_2)$. In particular, the prover has to prove that he knows the representation of C_2 in the bases B and V using the protocol proposed by Okamoto [40]. In other words, he has to prove that he knows a pair (t, u) such that $C_2 = B^t V^u$.
- (Π_2) A proof that the prover knows the plaintext related to the ciphertext $D = E_V[B^{sr^{-1}}] = (C_3, C_4)$. In particular, the prover has to prove that he knows the representation of C_4 in the bases B and V using the protocol proposed by Okamoto [40]. In other words, he has to prove that he knows a pair (v, w) such that $C_4 = B^v V^w$.
- (Π_3) A proof that the prover knows the representation of Com in the base g_2 and g_3.
- (Π_4) A proof that the discrete logarithm of C_4 in the base C_2 is equal to the discrete logarithm of Com in the base g_2 using a variant of the discrete log equality test owing to Chaum and Pedersen [19].

[1] Recall that in most e-voting systems, the voter has to prove that his choice belongs to the set of valid candidates [32].

Verification. The verification can next be executed as follows.

1. V checks all proofs and abort if one is invalid.
2. From the tuple $\langle B, E_V[B^{r^{-1}}], E_V[B^{sr^{-1}}], \Pi\rangle$, $\mathcal{V}$ performs the following steps to check whether this tuple *encrypts* a valid signature A on a message s (i.e; satisfying the relation $A^{y+s} = g_1$).
   (a) By means of his secret key y, $\mathcal{V}$ cooperatively computes: $E_V[B^{r^{-1}}]^y = E_V[B^{yr^{-1}}]$. Now $\mathcal{V}$ uses the El Gamal homomorphic property to compute: $E_V[B^{yr^{-1}}] \cdot E_V[B^{sr^{-1}}] = E_V[B^{yr^{-1}+sr^{-1}}] = E_V[A^{y+s}]$. From $E_V[B^{yr^{-1}+sr^{-1}}]$, the public parameter g_1 and the El Gamal homomorphic property, they can compute $E = E_V[B^{yr^{-1}+sr^{-1}} g_1^{-1}] = E_V[A^{y+s} g_1^{-1}]$.
   (b) $\mathcal{V}$ executes a Plaintext Equivalence Test [31] in order to determine whether C is an encryption of the ciphertext 1 or not. For this, $\mathcal{V}$ cooperatively selects a random number $\alpha \in \mathbb{Z}_q$ and computes C^α. $\mathcal{V}$ then cooperatively decrypts C^α. If the decryption result is equal to 1, then C is an encryption of 1. Otherwise, the result will be a random number and this indicates that the encrypted plaintext is different from 1.
   (c) In order to identify whether E encrypts a valid signature, $\mathcal{V}$ executes a Plaintext Equivalence Test [31] in order to determine whether E is an encryption of the ciphertext 1 or not. For this, $\mathcal{V}$ cooperatively selects a random number $z \in \mathbb{Z}_q$ and computes E^z. $\mathcal{V}$ then cooperatively decrypts E^z. If the decryption result is equal to 1, then E encrypts a valid signature and $\mathcal{V}$ accepts the proof. Otherwise, the result will be a random number and this indicates the encryption of an invalid signature.

3.3 Correctness

We now give some words about the correctness of our signature based method. In fact, the purpose of the proofs Π_i's is to ensure that only tuples which encrypt valid signatures will pass the test. Indeed, when the verifiers perform the test described at Step 2c of the verification phase to determine whether the tuple encrypt a valid signature or not, they compute the following ciphertext which is, using the above notation (see the remarks on the proofs Π_1 and Π_2) equal to $E_V[B^{yt}B^v g_1^{-1}]$. In other words this is an encryption of $B^{yt+v}g_1^{-1}$. If this is an encryption of 1, this means that $B^{yt+v}g_1^{-1} = 1$. Remember that at step 2b the verifiers check that C is not an encryption of 1. This therefore proves that $t \neq 0 \pmod q$. Let us denote by $A = B^t$ and $s = v/t$. So $B^{yt+v}g_1^{-1} = 1$ can be rewritten as follows: $A^{y+s}g_1^{-1} = 1$ which is equivalent to $A^{y+s} = g_1$. In other words, if the set of zero-knowledge proofs and the test are all valid, this means that the prover knows a tuple (A, s) such that $A^{y+s} = g_1$. Since $B^t = A$, where A is a valid signature on the message s and $t \neq 0 \pmod q$, $\mathcal{P}$ could not know the discrete logarithm of B in the base V. Therefore, the proof Π_4 ensures that the signed message s is the message committed in Com.

3.4 Comparison with Related Work

We now compare our new signature based set membership solution with the one described in [11], since they are very similar. As show in Table 1, our solution is definitely more efficient than Camenisch *et al.* one (choosing the number of verifiers $l_v = 2$ or 3). However, our scheme needs the restriction that the verification should be done cooperatively between at least two actors. In the case of an interactive set membership proof, we however notice that these two actors can be the verifier and the prover herself, which makes this restriction not relevant anymore. We here argue that the prover will not decrypt the ciphertexts if she does not want to, which makes the scheme secure.

Method	Prover's time complexity (mod mul in $\mathbb{G}$)	Verifier's time complexity (mod mul in $\mathbb{G}$)	Space complexity (in bits)	Size of public key (in bits)
Signature based [11]	$\frac{1817}{3}\|q\| + 3$	$\frac{14561}{24}\|q\| + 1$	$2\|\mathbb{G}\| + \|\mathbb{G}_T\| + 4q$	$(3 + \|\Phi\|)\|\mathbb{G}\| + \|\mathbb{G}_T\|$
Our signature method (basic)	$\frac{1975}{12}\|q\| + 6$	$\left(\frac{375}{2}\|q\| + 57\right)l_v$	$12\|\mathbb{G}\| + 8\|\mathbb{Z}_q\|$	$(2l_v + 3\|\Phi\| + 6)\|\mathbb{G}\| + 2\|\Phi\|q$

Table 1. Efficiency comparison of signature-based set membership proof

4 Our New Multi-base Decomposition Range Proof

In this section, we give our new method to prove that a secret value x belongs to a given public interval $[a, b]$, based on the multi-base decomposition of the secret. We describe it step by step to improve readability.

4.1 Focus on $x \leq b$

We first focus on the case $x \leq b$ using the technique presented in [23, 36], which generalized the binary case, first introduced by Bellare and Goldwasser [3], to the multi-base case one. In fact, we adapt the technique from [23, 36] to the case of a group of prime order (see Remark 1 below).

The idea is to use the multi-base decomposition (see Lemma 7 given in Section 2.6), and thus by taking the case where $v = b$ (which will now be the case in the rest of the paper). More precisely, if we denote $\ell = \lfloor \log_2 b \rfloor$, the proof that $x \leq b$ is done by using the fact that $x = [\![x_0, \ldots, x_\ell]\!]_b$ and by next committing on all the x_i's before proving that (i) each $x_i \in \{0, 1\}$ and that (ii) the relation $x = \sum_{i=0}^{\ell} b_i x_i$ holds. We here remember that each $b_i = \lfloor (b + 2^{\ell-i})/2^{\ell-i+1} \rfloor$ can be computed by both the prover and the verifier, since b is publicly known, and thus integrated in the public key. The complete proof that $x \in [a, b]$ can now be written as described in Figure 1.

1. The prover first randomly chooses $r_0, \ldots, r_\ell \in_R \mathbb{Z}_q$ and computes $C_i = g^{x_i} h^{r_i}$ for all $i \in [0, \ell]$. These C_i's are next sent to the verifier.
2. Both the prover and the verifier can compute $\tilde{C} = \prod_{i=0}^{\ell} C_i^{b_i}$, which is equal to $g^x h^{\sum_{i=0}^{\ell} b_i r_i}$ iff the x_i's correspond to the multi-base decomposition of x in base b. In the following, we denote $t = \sum_{i=0}^{\ell} b_i r_i$.
3. The prover and the verifier then play the following interactive ZKPK

$$U_1 = \textsc{Pok}(x, t, r_0, \ldots, r_\ell : (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \ldots \wedge (C_\ell = h^{r_\ell} \vee C_\ell/g = h^{r_\ell}) \wedge \tilde{C} = g^x h^t \wedge x \geq a)$$

Fig. 1. First version of our range proof

Note that the predicate $L_1 = (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \ldots \wedge (C_\ell = h^{r_\ell} \vee C_\ell/g = h^{r_\ell})$ permits to prove that each x_i is a bit (see Appendix A.2 to perform this ZKPK). The second part of U_1, related to $\tilde{C}$, proves that the relation $x = \sum_{i=0}^{\ell} b_i x_i$ holds (see above).

Remark 1. In fact, one may think that we only prove that $x = \sum_{i=0}^{\ell} b_i x_i \pmod q$. But, as the x_i's belong to $\{0, 1\}$, $\sum_{i=0}^{\ell} b_i x_i \leq \sum_{i=0}^{\ell} b_i = b < q$, since we consider, for obvious reasons, that $2^\ell < q$. Thus, $x = \sum_{i=0}^{\ell} b_i x_i$ (in $\mathbb{Z}$). This is why this method is very efficient in a group of prime order. This is not so efficient in the case of group of unknown order [23, 36], since it remains to prove that the equation holds in $\mathbb{Z}$, *e.g.* by using a proof of knowledge of three integers α, β, γ such that $\alpha = \beta\gamma$, which is done by using *e.g.* [23].

4.2 Treatment of the Most Significant Digits of $a \leq x$

In the above method, we have proved that the secret x can be represented in the multi-base b. Due to Lemma 7, this permits to prove that $x \leq b$. We now focus on the case $x \geq a$, but based on the above method. In fact, we also represent a in the multi-base b, that is, we have $a = [\![a_0, \ldots, a_\ell]\!]_b$, since $a < b$. We now compare x and a using their respective multi-base decompositions in base b, which permits us to make some simplifications.

Our first trick is to use the fact that both the prover and the verifier know the value a, and thus its representation in the multi-base b. We thus use the following result.

Lemma 11. $\exists! i_0 \in [0, \ell] / a_{i_0} = 0 \wedge \forall i > i_0, a_i = 1$

Proof. The existence is given by the fact that $a \neq b$ and the uniqueness is straightforward, assuming that the Algorithm 1 described in Section 2.6 is used[2]. □

[2] This is always the case in practice since this step is done at the generation phase, and thus by some trusted authorities.

This lemma says that some most significant digits of a in the multi-base b can be equal to 1. It also says that the first 0-bit of a is denoted i_0. Thus, using this lemma, and considering that both b_i and a_i are equals to 1 from the bit i_0+1 to the bit ℓ (see Lemma 8 for the b_i's), then it is necessary that $\forall i \in [i_0+1, \ell], x_i = 1$, since $x \in [a, b]$.

As the verifier knows a and b, the prover does not reveal any secret information if she simply opens the commitment on x_i, that is reveals the corresponding r_i. This trick permits to save the corresponding part of the proof of knowledge of L_1 (*i.e.* $\bigwedge_{j=i_0+1}^{\ell} \big((x_j = 0) \vee (x_j = 1)\big)$). Thus, instead of proving $x \geq a$, the verifier and the prover should determine i_0, and thus only focus on the proof that $\tilde{x} \geq \tilde{a}$, where $\tilde{a} = [\![a_0, \ldots, a_{i_0}]\!]_{\tilde{b}}$ and $\tilde{x} = [\![x_0, \ldots, x_{i_0}]\!]_{\tilde{b}}$, with $\tilde{b} = \sum_{i=0}^{i_0} b_i$.

For the general case, the computation of i_0 is given in Appendix B. We also give some practical cases in Appendix C with a different i_0. Note that this value, only depending on a and b, should only be computed once at the creation of the parameters a and b and can also be viewed as an additional parameter of the system.

4.3 Remaining Digits of $a \leq x$

We now focus on the comparison between $\tilde{a} = [\![a_0, \ldots, a_{i_0}]\!]_{\tilde{b}}$ and $\tilde{x} = [\![x_0, \ldots, x_{i_0}]\!]_{\tilde{b}}$. For this purpose, we use the bit-by-bit Lemma 10, which says that $a < x$ if and only if $\exists i' \in [0, \ell] / a_{i'} = 0$, $x_{i'} = 1$ and $\forall j > i', a_j = x_j$.

Study of the i-th digit. Under the fact that the verifier already knows a, we thus have the following for the i-th digit of a.

- If $a_i = 0$, there are two possible cases.
  - If $x_i = 0$, we have to prove this fact and next compare the remaining digits of $\tilde{a}$ and $\tilde{x}$, using the same method.
  - If $x_i = 1$, then we have to prove this fact, which is enough to claim that $a \leq x$.

  As we want to prove in a zero-knowledge manner that $a \leq x$, we have to consider both cases ($x_i = 0$ and $x_i = 1$), without saying which one is true. For this purpose, we use an "or" statement. If we denote by B the statement regarding the comparison between the remaining digits of a and x, this gives the proof
  $$(x_i = 1) \vee \big((x_i = 0) \wedge B\big).$$
- If $a_i = 1$, by construction of the recursion, we necessarily have $x_i = 1$. Then, we have to prove this fact and compare the remaining digits of $\tilde{a}$ and $\tilde{x}$ (from 0 to $i-1$), using the same method.

As the verifier knows the value a, she is able to decide whether the prover has to perform the first type of proof or the second one. We do not need to use again an "or" statement for both cases ($a_i = 0$ and $a_i = 1$).

The special case $i = i_0$. By construction, $a_{i_0} = 0$. We consequently necessary fall in the first case, and next have to prove that
$$(x_{i_0} = 1) \vee \big((x_{i_0} = 0) \wedge B\big),$$
where B is the statement regarding the comparison between the i_0 less significant digits of a and x. We now remark that if this predicate is true, the verifier is convinced that x_{i_0} is a bit. Consequently, the prover has not to prove that $(x_{i_0} = 0 \vee x_{i_0} = 1)$ in the L_1 part of the proof, as it was done in Figure 1, which slightly reduces the proof's size.

Simplification of the first case. Regarding the first case ($a_i = 0$), the predicate $(x_i = 1) \vee \big((x_i = 0) \wedge B\big)$ can be further simplified. If we denote by A the predicate $(x_i = 1)$, then the predicate $(x_i = 0)$ is $\neg A$ (as it is proven in L_1 that x_i is a bit). Moreover, as B is related to the remaining digits of a and b, it does not contain any predicate on x_i. Thus the predicates A and B are independent. Finally, in our main proof, the predicate B is never use again, while A is connected to the studied predicate by an "and" statement between $A \vee (\neg A \wedge B)$ and $A \vee \neg A$. As $A \vee \neg A$ is necessary true, it is possible to use the well-known result from boolean logic which says that the predicate $A \vee (\neg A \wedge B)$ is equal to $A \vee B$.

Description of the main algorithm. Using these results, we can now define Algorithm 2 only depending on the a_i's (it can thus be executed at the key generation phase) which outputs the proof one prover knowing an $x \geq a$ has to do.

Algorithm 2: FGreat($\{a_0, \cdots, a_i\}$)

```
if (a_i = ... = a_0 = 0) then L := ∅;
else
    if (a_i = 1) then
        if (i = 0) then L := (x_0 = 1);
        else L := (x_i = 1) ∧ [FGreat({a_0, ··· , a_{i−1}})];
    end
    else L := (x_i = 1) ∨ [FGreat({a_0, ··· , a_{i−1}})];
end
return L;
```

Note that the optimization for $i = i_0$ can not be done for the other digits ($i < i_0$). Indeed, if we simplify the remaining predicate, a prover which knows a secret such that $x_{i_0} = 1$ may simulates the whole remaining list. Consequently, for all $i < i_0$, the verifier is not convinced that x_i is a bit and a fraud is next possible. As a consequence, the prover still has to prove it.

As a conclusion, a prover should prove that the predicate $(x_{i_0} = 1) \vee [(x_{i_0} = 0) \wedge \text{FGreat}(\{a_0, \cdots, a_{i_0-1}\})]$ is true.

4.4 Summary of Our Method

Let b, $a = [\![a_0, \cdots, a_\ell]\!]_b$ and $x[\![x_0, \cdots, x_\ell]\!]_b$ be three integers such that $a < b$ and $x \in [a, b]$. At the key generation phase, one has to first determine i_0 such that $a_{i_0} = 0$ and $\forall i > i_0, a_i = 1$. It next executes the Algorithm 2 named FGreat on input $\{a_0, \cdots, a_{i_0-1}\}$ in order to obtain L.

Range proof protocol. During the main protocol, the prover and the verifier follow the steps described in Figure 2. A concrete example of our new method is described in Appendix C.

1. The prover first randomly chooses $r_0, \ldots, r_\ell \in_R \mathbb{Z}_q$ and computes $C_i = g^{x_i} h^{r_i}$ for all $i \in [0, \ell]$. These C_i's are next sent to the verifier.
2. Both the prover and the verifier can compute $\tilde{C} = \prod_{i=0}^{\ell} C_i^{b_i}$, which is equal to $g^x h^{\sum_{i=0}^{\ell} b_i r_i}$ iff the x_i's correspond to the multi-base decomposition of x. We now denote $t = \sum_{i=0}^{\ell} b_i r_i$.
3. For all $i \in [i_0 + 1, \ell]$, the prover reveal r_i.
4. The prover and the verifier then play the following interactive ZKPK

$$U_f = \text{Pok}\Big(x, t, r_0, \ldots, r_{i_0} : \tilde{C} = g^x h^t \wedge (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \ldots \wedge$$
$$(C_{i_0-1} = h^{r_{i_0-1}} \vee C_{i_0-1}/g = h^{r_{i_0-1}}) \wedge \big(C_{i_0}/g = h^{r_{i_0}} \vee (C_{i_0} = h^{r_{i_0}} \wedge L)\big)\Big)$$

Fig. 2. Final version of our range proof

4.5 Security Consideration

The security of our new method is quite easy to study. Steps 1 and 2 above are only composed of commitments. Moreover, it is relatively obvious that both the revelation of the r_i's in the third step and the final proof U_f do not reveal any information about prover's secret other than what can be expected by the knowledge of a and b. This is due to (i) the methodology described in [36], (ii) Lemma 11 and the remark given in Section 4.2 and (iii) Lemma 10 and the description given in Section 4.3.

5 Efficiency Comparison of Range Proof Methods

In this section, we compare the efficiency of both related work and our proposals to know which solution has to be used in which cases. Note that each proof considers that the commitment on x has already been done and thus, this step is not considered in the efficiency comparison.

We here use the results given in the appendix and we summarize the main results in Table 2. Considering a security level of 128, we take the following concrete values for our comparison (see also Appendix D for some details): $|q| = 256$, $|\mathbb{G}| = 257$, $l_{\mathbb{Z}_n} = 3248$, $l_e = l_s = 160$. For the following graphics, we use the captions given in Figure **??**.

Method	Prover's time complexity (mod mul in $\mathbb{G}$)	Verifier's time complexity (mod mul in $\mathbb{G}$)	Space complexity (in bits)	Size of public key (in bits)
Double binary [46, 47]	$\frac{293\|q\|+12}{6}\ell + \frac{359\|q\|+3}{3}$	$\frac{175\|q\|+6}{3}\ell + \frac{125\|q\|}{2} + \frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell}}\|q\|$	$(6\|\mathbb{G}\| + 8\|q\|)\ell + (6\|\mathbb{G}\| + 4\|q\|)$	$4(\|\mathbb{G}\| + \|q\|)$
Multi-base decomposition [36]	$\frac{75\|q\|+1}{2}\ell + \frac{325\|q\|+2}{4}$	$\frac{175\|q\|+6}{6}\ell + \frac{1775\|q\|+48}{24} + \frac{25}{3}\frac{2^{\ell+3}-1}{2^{\ell+2}}\|q\|$	$(3\|\mathbb{G}\| + 4\|q\|)\ell + (5\|\mathbb{G}\| + 5\|q\|)$	$2\|\mathbb{G}\| + (\ell+5)\|q\|$
Square decomposition [35]	$(30.2l_{\mathbb{Z}_n} + 16.2l_e + 34.1l_s + 2\ell)(\frac{l_{\mathbb{Z}_n}}{\|q\|})^2$	$(17.1l_{\mathbb{Z}_n} + 17.1l_e + 21.1l_s + 2\ell + 6)(\frac{l_{\mathbb{Z}_n}}{\|q\|})^2$	$11\|\mathbb{G}\| + 10l_{\mathbb{Z}_n} + 15l_e + 19l_s + \frac{5}{2}\ell$	$2\|\mathbb{G}\| + l_{\mathbb{Z}_n} + l_s + l_e$
Square decomposition [29]	$(8.75l_{\mathbb{Z}_n} + 16.7l_e + 16.7l_s + 7.9\ell + 7.9)(\frac{l_{\mathbb{Z}_n}}{\|q\|})^2$	$(9.4l_{\mathbb{Z}_n} + 16.3l_e + 13.3l_s + 4\ell + 6)(\frac{l_{\mathbb{Z}_n}}{\|q\|})^2$	$7\|\mathbb{G}\| + 6l_{\mathbb{Z}_n} + 11l_e + 10l_s + 4\ell$	$7\|\mathbb{G}\| + l_{\mathbb{Z}_n} + l_s + l_e$
Signature based [11]	$1250\|q\| + 66.7k\|q\| - 1148$	$583.8\|q\| + 12.5u\|q\| + 10u + 1181k - 1150$	$(u+k+4)\|\mathbb{G}\| + (2k+2)\|\mathbb{G}_T\| + (4k+8)q$	$4\|\mathbb{G}\| + (k+3)q$
Our multi-base method	$\left(\frac{175\|q\|+12}{96}\right)\ell^2 + \left(\frac{5275\|q\|+156}{96}\right)\ell + \left(\frac{1025\|q\|+12}{48}\right)$	$\left(\frac{525\|q\|+24}{12}\right)\ell - \left(\frac{75\|q\|+6}{12}\right) + \left(\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell+1}}\right)\|q\|$	$(4\|\mathbb{G}\| + 6\|q\|)\ell - 3\|q\|$	$2\|\mathbb{G}\| + \ell + \|q\|$

Table 2. Efficiency comparison

Verifier. The verifier's efficiency comparison between existing methods is given in Figure 4. For $\ell > 25$, it is clear that signature-based solutions are the most interesting ones. When ℓ is small, $\ell \leq 5$, then our new multi-based decomposition solution is the most interesting one, while the initial work from [36] is the ideal one when $5 < \ell \leq 25$.

Prover. The prover's efficiency comparison between all methods is given in Figure 5. The result is relatively similar to the previous one, except that the breaking points are different. This time, signature based solution are the most interesting ones when $\ell > 60$, while ours is the one to be chosen when $\ell \leq 3$. Again, the initial multi-base decomposition scheme [36] is the most interesting one when $3 < \ell \leq 60$.

Space. We finally compare the space complexities in Figure 6, with the conclusion that, again, our method is very interesting when $\ell \leq 5$. Next, for $5 < \ell \leq 14$, the double binary method [46, 47] is the most interesting one, while for $14 < \ell \leq 24$, one has to choose a signature-based method. Finally, for $\ell > 24$ the square decomposition methods [35, 29] are the most efficient ones.

6 Conclusion

It seems impossible to determine which range proof is the best one. It seems that for $\ell \leq 3$, our new multi-base decomposition method is the one to be chosen, while the Lipmaa-Asokan-Niemi solution [36] is the one for $5 < \ell \leq 25$ and signature-base methods are more interesting when $\ell > 60$. Next, for $3 < \ell \leq 5$ and $\ell < 60$, the choice should be done in accordance to the specificities of the system, regarding the computational strength of both the prover and the verifier, and the flow between both of them.

Regarding signature-based solutions, our proposal is the most efficient one, under the condition that the set of verifiers own a secret decryption key, which is *e.g.* the case for e-voting purpose.

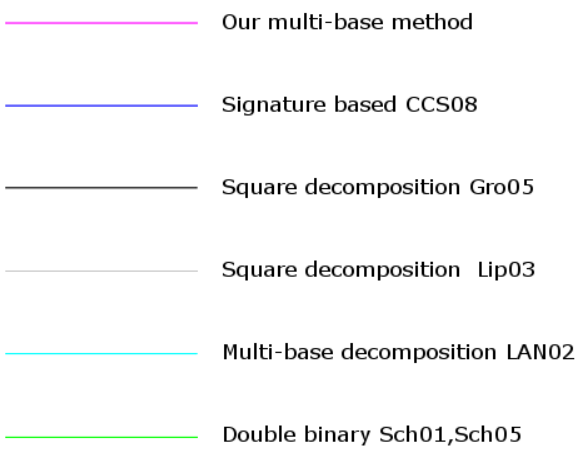

Fig. 3. Caption for graphics

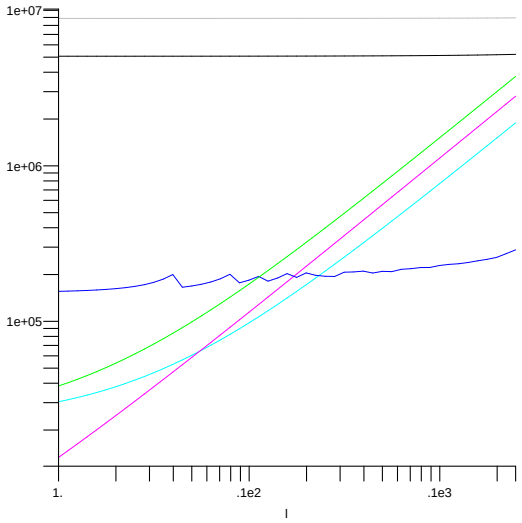

Fig. 4. Verifier's efficiency comparison for different values of ℓ

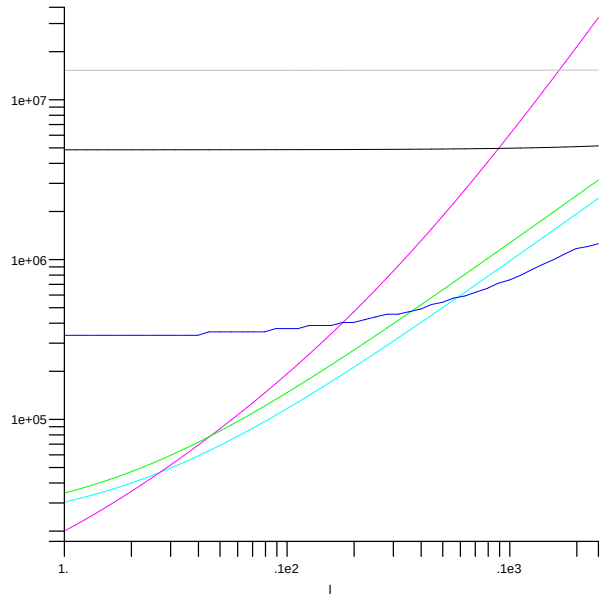

Fig. 5. Prover's efficiency comparison for different values of ℓ

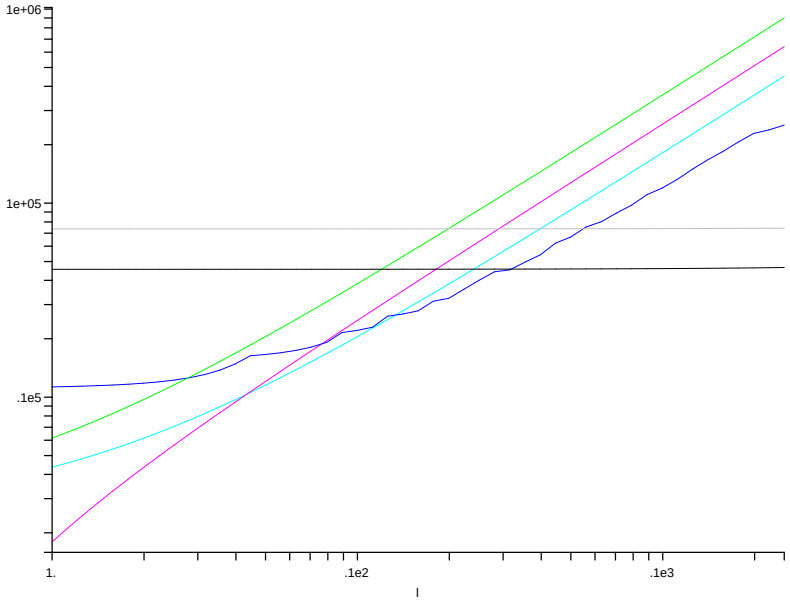

Fig. 6. Space's efficiency comparison for different values of ℓ

References

1. Christophe Arene, Tanja Lange, Michael Naehrig, and Christophe Ritzenthaler. Faster computation of the tate pairing. Cryptology ePrint Archive, Report 2009/155, 2009. http://eprint.iacr.org/.
2. Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC*, pages 274–283, 2001.
3. Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In *ACM Conference on Computer and Communications Security*, pages 78–91, 1997.
4. Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. EFD, 2009. http://www.hyperelliptic.org/EFD/.
5. Ian Blake, Gadiel Seroussi, and Nigel Smart. Elliptic curves in cryptography, 1999.
6. Dan Boneh. The decision diffie-hellman problem. In *ANTS'98*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 1998.
7. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.
8. F. Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer-Verlag, 2000.
9. Ernest F. Brickell, David Chaum, Ivan Damgård, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 156–166, 1987.
10. J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. In *EUROCRYPT'09*, volume 5479 of *LNCS*, pages 425–442. Springer, 2009.
11. Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer, 2008.
12. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology - Eurocrypt '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer-Verlag, 2005.
13. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. K. Franklin, editor, *Advances in Cryptology - Crypto 04*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer-Verlag, 2004.
14. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.
15. Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In J. Stern, editor, *Advances in Cryptology - Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer-Verlag, 1999.
16. Sébastien Canard, Iwen Coisel, and Jacques Traoré. Complex zero-knowledge proofs of knowledge are easy to use. In *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 2007.
17. Sébastien Canard, Aline Gouget, and Emeline Hufschmitt. A handy multi-coupon system. In *ACNS 2006*, volume 3989 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 2006.
18. Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In K. Nyberg, editor, *Advances in Cryptology - Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 561–575. Springer-Verlag, 1998.
19. D. Chaum and T. Pedersen. Wallet Databases with Observers. In E. F. Brickell, editor, *Advances in Cryptology - Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.
20. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *Advances in Cryptology - Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.
21. Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.
22. Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002.
23. Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.
24. A. Fiat and A. Shamir. How to Prove Yourself : Practical Solutions to Identification and Signature Problems. In A. M. Odlyzko, editor, *Advances in Cryptology - Crypto '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, 1987.
25. Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 457–472. Springer, 2001.
26. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.
27. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 1999.
28. M. Girault, G. Poupard, and J. Stern. On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order. *J. Cryptology*, 19(4):463–487, 2006.
29. Jens Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 467–482. Springer, 2005.
30. Louis C. Guillou, Michel Ugon, and Jean-Jacques Quisquater. Cryptographic authentication protocols for smart cards. *Computer Networks*, 36(4):437–451, 2001.
31. Markus Jakobsson and Ari Juels. Addition of elgamal plaintexts. In *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 346–358. Springer, 2000.

32. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES 2005*, pages 61–70. ACM, 2005.
33. Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable Signatures. In *Eurocypt'04*, pages 571–589, 2004.
34. Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 249–263. Springer, 1997.
35. Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 398–415. Springer, 2003.
36. Helger Lipmaa, N. Asokan, and Valtteri Niemi. Secure vickrey auctions without threshold trust. In *Financial Cryptography 2002*, volume 2357 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2002.
37. Ben Lynn. *On the Implementation of Pairing-nased Cryptosystems*. PhD thesis, Stanford University, 2007.
38. Bodo Möller. Algorithms for multi-exponentiation. In *SAC '01: Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, pages 165–180, London, UK, 2001. Springer-Verlag.
39. European Network of Excellence in Cryptology II. Ecrypt2 yearly report on algorithms and keysizes (2008-2009), 2009.
40. Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*. Springer, 1992.
41. T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1992.
42. D. Pointcheval and J Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13:361–396, 2000.
43. J. O. Rabin and Jeffrey Shallit. Randomized algorithms in number theory. Technical report, University of Chicago, Chicago, IL, USA, 1985.
44. A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung. On Monotone Formula Closure of SZK. *FOCS 1994*, pages 454–465, 1994.
45. C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In G. Brassard, editor, *Advances in Cryptology - Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer-Verlag, 1990.
46. Berry Schoenmakers. Some efficient zero-knowledge proof techniques. In *Workshop on Cryptographic Protocols*, 2001.
47. Berry Schoenmakers. Interval proofs revisited. In *Workshop on Frontiers in Electronic Elections*, 2005.
48. Isamu Teranishi and Kazue Sako. -times anonymous authentication with a constant proving cost. In *Public Key Cryptography 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 525–542. Springer, 2006.
49. Hong Wang, Yuqing Zhang, and Dengguo Feng. Short threshold signature schemes without random oracles. In *INDOCRYPT*, volume 3797 of *Lecture Notes in Computer Science*, pages 297–310. Springer, 2005.

A Notations and Useful Tools

A.1 Complexity Assumptions Needed in this Paper

All along our paper, we need the following complexity assumptions.

$\mathfrak{q}$-Strong Diffie-Hellman assumption ($\mathfrak{q}$-SDH) [7]: In [7], Boneh and Boyen introduced a new computational problem in *bilinear* context. However, for our purpose, we will consider this problem in the classical discrete log setting, i.e. without *bilinear map*.

Let k denotes a security parameter. Let $\mathbb{G}$ be a group of prime order q with $2^k < q < 2^{k+1}$. We say that the $\mathfrak{q}$-SDH assumption holds in $\mathbb{G}$ if for all polynomial-time adversaries A the advantage

$$\mathrm{Adv}^{\mathfrak{q}\text{-SDH}}_{\mathbb{G},A}(k) = \Pr[A(g, g^x, ..., g^{x^{\mathfrak{q}}}) = (c, g^{1/(x+c)})]$$

is a negligible function in k, where $g \leftarrow \mathbb{G}^*$ and $x, c \leftarrow \mathbb{Z}_q$.

Decision Diffie-Hellman assumption (DDH) [6]: Let k denotes a security parameter. Let $\mathbb{G}$ be a group of prime order q with $2^k < q < 2^{k+1}$. We let $\mathrm{D}_{\mathrm{DDH}}$ be the distribution $(g,\ g^x,\ g^y,\ g^{xy})$ in $\mathbb{G}^4$ where g is a random generator in $\mathbb{G}$ and x, y are uniform in $\mathbb{Z}_q$. We let $\mathrm{R}_{\mathrm{DDH}}$ be the distribution $(g,\ g^x,\ g^y,\ g^z)$ where g is a random generator in $\mathbb{G}$ and x, y, z are uniform in $\mathbb{Z}_q$ subject to $z \neq xy$. We say that the DDH assumption holds in $\mathbb{G}$ if for all polynomial-time adversaries A the advantage

$$\mathrm{Adv}^{\mathrm{DDH}}_{\mathbb{G},A}(k) = |\Pr[x \leftarrow \mathrm{D}_{\mathrm{DDH}} \ : A(x) = 1] - \Pr[x \leftarrow \mathrm{R}_{\mathrm{DDH}} \ : A(x) = 1]|$$

is a negligible function in k.

A.2 Some Zero-Knowledge Proofs of Knowledge

In this paper, the zero-knowledge proofs of knowledge are constructed over a cyclic group $\mathbb{G} =< g >$ either of prime order q (in our case an elliptic curve such as described in Section 2) or of unknown order[3](but where the bit-length of the order is $l_{\mathbb{G}}$). The base of each construction is either the Schnorr authentication scheme [45] for a prime order group or the GPS authentication scheme [28] for an unknown order group. Note that it may be necessary to check that the components of some elements are of order q to prevent the attacks described in [34] (see also [10]).

These constructions should verify the soundness and zero-knowledge properties [33, 16, 10]. Relations proved in zero-knowledge are typically the following ones:

- proof of knowledge of a discrete logarithm [45, 28]: $\text{POK}(\alpha : y = g^{\alpha})$;
- proof of knowledge of a representation [40]: $\text{POK}(\alpha_1, \ldots, \alpha_q : y = g_1^{\alpha_1} \ldots g_q^{\alpha_q})$;
- proof of equality of discrete logarithms [19]: $\text{POK}(\alpha : y = g^{\alpha} \wedge z = h^{\alpha})$;
- proof of the "or" statement [20, 44] (proof of knowledge of one representation among k ones): $\text{POK}(\{\alpha_{ij}; j \in J_i\} : \bigvee_{i=1}^{k} C_i = \prod_{j \in J_i} g_j^{\alpha_{ij}})$;
- plaintext equivalence test [31].

Note that from $C = g^x h^r$, the predicate $(x = 1)$ (resp. $(x = 0)$) can be proven, in a zero-knowledge manner, by $\text{POK}(r : C/g = h^r)$ (resp. $\text{POK}(r : C = h^r)$). As a consequence, the predicate $(x = 1 \vee x = 0)$ is related to the proof of knowledge $\text{POK}(r : C/g = h^r \vee C = h^r)$.

In this paper, we independently use both notation, one in case we speak about the predicate, and the other one when we speak about the ZKPK. We consider as obvious the way to go from one notation to the other and do not detail it each time.

Remark 2. Note that $\text{POK}(r : C/g = h^r)$ prove that $C = gh^r$ and thus that $x = 1$. Consequently, this proof of knowledge is not zero-knowledge regarding x. Nevertheless, $\text{POK}(r : C/g = h^r \vee C = h^r)$ ensures this property as anybody is able to learn if x equals 0 or 1. The only information given by this proof is that x is a bit.

A.3 Boneh-Boyen Signatures

Boneh and Boyen have proposed a short signature scheme [7], secure under the $\mathfrak{q}$-SDH assumption (see above), for which it is possible to prove the knowledge of a signature on a message, without revealing the signature nor the message. On input a secret γ and a generator g of a group $\mathbb{G}$ of prime order q, the signer can sign a message $m \in \mathbb{Z}_q$ by computing $\sigma = g^{1/(\gamma+m)}$. The verification is done using g, the public key $w = g^{\gamma}$ and a bilinear map e by checking that $e(\sigma, wg^m) = e(g, g)$.

As sketch in [14] and used in [11], the zero-knowledge proof of a message m and a corresponding Boneh-Boyen signature is done by first computing $C = g^m h^s$ and $T = \sigma^r$ where $r, s \in \mathbb{Z}_q$ and next making the zero-knowledge proof of knowledge $\text{POK}(m, r, s : C = g^m h^s \wedge e(T, w) = e(T, g)^x e(g, g)^r)$.

Boneh-Boyen signature without pairings. In Section 3.1, we have described a variant of this Boneh-Boyen signature scheme without pairings. We here give the proof that the proposed scheme is EU-CMA.

Theorem 1. *The BB without pairing scheme is existential unforgeable under a weak chosen message attack under the* $\mathfrak{q}$*-Strong Diffie-Hellman assumption, in the random oracle model.*

Proof (sketch). Under the $\mathfrak{q}$-Strong Diffie-Hellman assumption, this is not possible to find a A such that $A = g_1^{\frac{1}{y+m}}$ for a chosen m which is not given to the signing oracle, as proved in [7]. Moreover, in the random oracle model, the additional signature of knowledge Π is unforgeable [24, 42], which concludes the proof. □

[3] Damgård and Fujisaki [22] showed that, under the Flexible RSA Assumption, the standard proofs of knowledge that work for a group of known order are also proofs of knowledge in this setting.

A.4 A Threshold Cryptosystem

Our signature-based scheme relies on a threshold version of a semantically secure cryptosystem with homomorphic property, such as El Gamal [26] under secure groups. Let $\mathbb{G}$ be a cyclic group of order q where the DDH problem is hard. The public key is composed of the elements $(g, h = g^x)$ with $h, g \in \mathbb{G}$ and the corresponding private key is formed by $x \in \mathbb{Z}_q$. The El Gamal ciphertext of a message $m \in \mathbb{G}$ is $(C_1 = g^r, C_2 = mh^r)$, where $r \in \mathbb{Z}_q$ is a random number. The message m is obtained from the ciphertext (C_1, C_2) by $C_2/(C_1^x)$. The El Gamal cryptosystem is *semantically secure* under the DDH assumption. In the threshold version, the El Gamal public key and its corresponding private key are cooperatively generated by n parties; though, the private key is shared among the parties. In order to decrypt a ciphertext, a minimal number of t out of n parties is necessary.

A complete description of an El Gamal threshold cryptosystem is given in [21] while Gennaro *et al.* [27] describe a secure key generation protocol.

B A Result on Probability Theory

Let $a = [\![a_0, \cdots, a_\ell]\!]_v$ be a uniformly and randomly chosen integer in the range $[0, v]$ with its corresponding multi-base decomposition in base v. We want to know how many successive most significant digits of a are equals to 1, in average. More precisely, we are searching the value i_0 such that $a_{i_0} = 0$ and $\forall i \in [i_0 + 1, \ell], a_i = 1$.

For this purpose, we introduce the random variable N representing the number of successive most significant digits of a being equal to 1. The value we want to compute is next the expected value of N, that is $E(N) = \sum_{n \in N} n \cdot \Pr(N = n)$ where $\Pr(N = n)$ is the probability that $N = n$. As a is an ℓ-digit integer, we can simplify by writing $E(N) = \sum_{n=0}^{\ell} n \cdot \Pr(N = n)$. We have now to compute $\Pr(N = n)$ for all $n \in [0, \ell]$.

For example, $\Pr(N = 1) = \Pr(a_\ell = 1 \cap a_{\ell-1} = 0) = \Pr(v_\ell \leq a < v_\ell + v_{\ell-1})$. We can now generalize this result to obtain the following:

$$\Pr(N = n) = \Pr\Big(\sum_{i=\ell-n+1}^{\ell} v_i \leq a < \sum_{\ell-n}^{\ell} v_i\Big) = \frac{v_{\ell-n}}{v+1}$$

For the completeness of this formula, we define $v_{-1} = 1$. Let us now focus on the expected value of N, that is

$$\begin{aligned} E(N) &= \sum_{n=0}^{\ell+1} n \cdot \Pr(N = n) = \sum_{n=1}^{\ell+1} n \frac{v_{\ell-n}}{v+1} \\ &= \frac{1}{v+1} \sum_{n=1}^{\ell+1} n \left\lfloor \frac{v + 2^n}{2^{n+1}} \right\rfloor \end{aligned}$$

The expected value $E(N)$ can now be bounded as follows

$$\begin{gathered} \frac{1}{v+1} \sum_{n=1}^{\ell+1} n\Big(\frac{v+2^n}{2^{n+1}} - 1\Big) \leq E(N) \leq \frac{1}{v+1} \sum_{n=1}^{\ell+1} n\Big(\frac{v+2^n}{2^{n+1}}\Big) \\ \frac{1}{v+1}\Big(\Big(\sum_{n=1}^{\ell+1} \frac{nv}{2^{n+1}}\Big) + \Big(\sum_{n=1}^{\ell+1} \frac{n}{2}\Big) - \Big(\sum_{n=1}^{\ell+1} n\Big) \leq E(N) \leq \frac{1}{v+1}\Big(\Big(\sum_{n=1}^{\ell+1} \frac{nv}{2^{n+1}}\Big) + \Big(\sum_{n=1}^{\ell+1} \frac{n}{2}\Big)\Big) \\ \frac{1}{v+1}\Big(\Big(v \sum_{n=1}^{\ell+1} \frac{n}{2^{n+1}}\Big) - \Big(\frac{1}{2} \sum_{n=1}^{\ell+1} \frac{n}{2}\Big)\Big) \leq E(N) \leq \frac{1}{v+1}\Big(\Big(v \sum_{n=1}^{\ell+1} \frac{n}{2^{n+1}}\Big) + \Big(\frac{1}{2} \sum_{n=1}^{\ell+1} n\Big)\Big) \\ \frac{1}{v+1}\left(v\theta_\ell - \frac{(\ell+1)(\ell+2)}{4}\right) \leq E(N) \leq \frac{1}{v+1}\left(v\theta_\ell + \frac{(\ell+1)(\ell+2)}{4}\right) \end{gathered}$$

where

$$\theta_\ell = \sum_{n=1}^{\ell+1} \frac{n}{2^{n+1}} = \frac{1}{4} \sum_{n=1}^{\ell+1} \frac{n}{2^{n-1}}.$$

Let $f_n : x \longmapsto x^n$ and $F_\ell : x \longmapsto \sum_{n=1}^{\ell+1} f_n(x)$. It is obvious that for all $x \in \mathbb{R}$, $f_n'(x) = nx^{n-1}$, where f_n' denotes the derivative of f_n. Thus, for all $x \in \mathbb{R}$,

$$F_\ell'(x) = \sum_{n=1}^{\ell+1} f_n'(x) = \sum_{n=1}^{\ell+1} nx^{n-1} \text{ and } \theta_\ell = \frac{1}{4}F_\ell'(1/2),$$

where F_ℓ' denotes the derivative of F_ℓ. Moreover, for all $x \in \mathbb{R}$,

$$F_\ell(x) = \sum_{n=1}^{\ell+1} x^n = \frac{x - x^{\ell+2}}{1-x}.$$

and thus for all $x \in \mathbb{R}$,

$$\begin{aligned} F_\ell'(x) &= \frac{(1-(\ell+2)x^{\ell+1})(1-x) + (x - x^{\ell+2})}{(1-x)^2} \\ &= \frac{1-(\ell+2)x^{\ell+1} - x + (\ell+2)x^{\ell+2} + x - x^{\ell+2}}{(1-x)^2} \\ &= \frac{1-(\ell+2)x^{\ell+1} + (\ell+1)x^{\ell+2}}{(1-x)^2} \end{aligned}$$

and consequently

$$\begin{aligned} F_\ell'(1/2) &= \frac{1-(\ell+2)(1/2)^{\ell+1} + (\ell+1)(1/2)^{\ell+2}}{(1-1/2)^2} \\ &= 4\Big(1 - (2\ell + 4 - \ell - 1)\Big(\frac{1}{2}\Big)^{\ell+2}\Big) \\ &= 4\Big(1 - (\ell+3)\Big(\frac{1}{2}\Big)^{\ell+2}\Big) \end{aligned}$$

and thus

$$\theta_\ell = 1 - \frac{\ell+3}{2^{\ell+2}}.$$

Finally, we obtain the following

$$\frac{1}{v+1}\left(v\Big(1 - \frac{\ell+3}{2^{\ell+2}}\Big) - \frac{(\ell+1)(\ell+2)}{4}\right) \leq E(N) \leq \frac{1}{v+1}\left(v\Big(1 - \frac{\ell+3}{2^{\ell+2}}\Big) + \frac{(\ell+1)(\ell+2)}{4}\right)$$

As $\ell = \lfloor \log_2 v \rfloor$, we obtain Table 3 which gives us that, in average, there is **one** successive most significant digits of a which is equal to 1.

v	1	2	5	10	10^2	10^3	10^4	10^5
underestimating of i_0	0.375	0.5	0.695	0.811	0.971	0.996	0.999	0.999
increasing of i_0	0.625	0.833	0.972	1.007	1.009	1.001	1.000	1.000

Table 3. Number of most significant digits equal to 1 in average

However, there are some special cases where this value is higher, as we will see in the next Appendix C.

C Concrete Examples for Our Multi-base Decomposition Method

We now introduce a practical case and use our method on them. We assume that we want to prove that a secret x belongs to the interval $[65, 130]$. Thus, $\lfloor \log_2 130 \rfloor = 7$ $i_0 = 6$, and $b_0 = 1$, $b_1 = 1$, $b_2 = 2$, $b_3 = 4$, $b_4 = 8$, $b_5 = 16$, $b_6 = 33$ and $b_7 = 65$. We now have $a = 65 = [\![0, 0, 0, 0, 0, 0, 0, 1]\!]_{130}$, $b = [\![1, 1, 1, 1, 1, 1, 1, 1]\!]_{130}$ and $x = [\![x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]\!]_{130} \in [65, 130]$. By executing the algorithm described above, we obtain the following results:

– the prover has to reveal r_7;
– $\tilde{L} := \text{FGREAT}(\{0,0,0,0,0,0\}) = \emptyset$;
– the final proof U_f if the following:

$$U_f = \text{POK}\big(x, t, r_0, r_1, r_2 : \tilde{C} = g^x h^t \wedge (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \cdots \wedge (C_6 = h^{r_6} \vee C_6/g = h^{r_6})\big).$$

Interval [81, 90]. We want to prove that a secret x belongs to the interval $[81, 90]$. Thus, $\lfloor \log_2 90 \rfloor = 6$, $i_0 = 3$, and $b_0 = 1$, $b_1 = 1$, $b_2 = 3$, $b_3 = 6$, $b_4 = 11$, $b_5 = 23$, and $b_6 = 45$. We now have $a = 81 = [\![1,1,0,0,1,1,1]\!]_{90}$, $b = [\![1,1,1,1,1,1,1]\!]_{90}$ and $x = [\![x_0, x_1, x_2, x_3, x_4, x_5, x_6]\!]_{90} \in [81, 90]$. By executing the algorithm described above, we obtain the following results:

– the prover has to reveal r_4, r_5, r_6;
– $\tilde{L} := \text{FGREAT}(\{1,1,0,0\}) = C_3/g = h^{r_3} \vee C_2/g = h^{r_2} \vee (C_1/g = h^{r_1} \wedge C_0/g = h^{r_0})$;
– the final proof U_f is the following:

$$U_f = \text{POK}\big(x, t, r_0, r_1, r_2, r_3 : \tilde{C} = g^x h^t \wedge (C_0 = h^{r_0} \vee C_0/g = h^{r_0}) \wedge \cdots \wedge (C_2 = h^{r_2} \vee C_2/g = h^{r_2}) \wedge (C_3/g = h^{r_3} \vee C_2/g = h^{r_2} \vee (C_1/g = h^{r_1} \wedge C_0/g = h^{r_0}))\big).$$

D Complexity Tools

As we are considering that we work over an elliptic curve, the basic operation we will use is the addition of points in the curve. Using ECRYPT II Recommendations [39] with a security level of 128, the elements of $\mathbb{F}_p$ are typically 256-bits integers. Thus, as it is necessary to store a point using the x-coordinate plus additional one bit to know which y-coordinate is to be chosen [5], a point needs 257 bits to be stored. The integer q should also be chosen as a 256-bit integer.

One aim of our paper is to make a complete comparison between all existing range proof methods. The problem here is that some of them need pairings [11], some others (square decomposition based [8, 35, 29]) need to work over $\mathbb{Z}_n$ where n is an RSA modulus. We thus need to know what is the ratio between for example a pairing and a modular multiplication in $\mathbb{Z}_n$ or a point addition in an elliptic curve, which is not an easy task. We here give our ratio, based on the addition of points in the above elliptic curve, using the existing literature on the subject.

D.1 Modular multiplication in $\mathbb{Z}_n$

As we are considering a security level of 128, we need to use a 3248-bits RSA modulus, according to [39]. Using the bouncycastle Java implementation of modular multiplication and the point addition in the secp-256r1 elliptic curve, we obtain that $M_{3248} = 5.2A_{256}$ where M_{3248} is the cost for the modular multiplication and A_{256} is the cost for the points addition.

D.2 Use of Shamir's trick

We also recall that the computation of a representation $c = \prod_{i=1}^{l} g_i^{e_i}$ can be improved by the use of the well-known Shamir's trick, presented in [38]. Note that this technique is generic and consequently also work for the elliptic curve case. In a nutshell, it is not necessary to compute each modular exponentiation and multiply the results since c can be computed globally. In most cases, this permits to save lots of computation. We thus consider that the computation of c necessitates approximatively $\frac{2^{l+1}-1}{3\times 2^{l-1}}$ times the cost of a modular exponentiation modulo a q-bits integer, with an exponent of size b the greatest bit length of the e_i's.

Note moreover that this is a well-known result in the modular arithmetic theory that one modular exponentiation modulo a n-bits number and with an exponent of size e corresponds to $\frac{3}{2}e$ modular multiplications modulo a n-bits number. We can thus conclude that one multi-exponentiation with n terms necessitates $b\frac{2^{l+1}-1}{2^l}$ modular multiplications, where b is the greatest bit length of the e_i's.

Regarding elliptic curves, if we consider one scalar multiplication with a scalar of size e, then we have $S_{256} = \frac{25e}{2}\mathbf{m}$, using the results from [4] for Edwards curves, where S_{256} is the cost for the scalar multiplication and $\mathbf{m}$ is the cost of multiplication in the base field $\mathbb{F}_p$. This gives $b\frac{25}{3}\frac{2^{l+1}-1}{2^l}\mathbf{m}$ for a multi-exponentiation with n terms.

D.3 The case of pairings

It is today not an easy task to obtain the efficiency of a pairing evaluation which can be used in our purpose. We have made the choice of using the recent work from Arène, Lange, Naehrig and Ritzenthaler [1] on the Tate pairing for Edwards curves, which is, to the best of our knowledge, the most efficient pairing implementation.

It is written in this paper that the evaluation of the reduced Tate pairing, given by $e : E(\mathbb{F}_p)[q] \times E(\mathbb{F}_{p^k})/qE(\mathbb{F}_{p^k}) \to \mu_q$ where $\mu_q \subset \mathbb{F}^*_{p^k}$ denotes the group of q-th roots of unity, needs $(|q| - 1)$ iterations of one point doubling and one point addition. If $\mathbf{s}$ denotes the cost of squaring in the base field $\mathbb{F}_p$ and if $\mathbf{M}$ and $\mathbf{S}$ denote the costs of multiplication and squaring in the extension field of degree $k = 6$, then the doubling step takes $1\mathbf{M} + 1\mathbf{S} + (k + 6)\mathbf{m} + 5\mathbf{s}$ while the mixed addition step needs $1\mathbf{M} + (k+12)\mathbf{m}$. Assuming that $\mathbf{M} = \mathbf{S}$ and $\mathbf{m} = \mathbf{s}$, and, using [37], that $\mathbf{M} = (k-1)k^2\mathbf{m}$, we obtain that one pairing evaluation P_{256} necessitates $(|q| - 1)(3(k - 1)k^2 + 2k + 23) = 575(|q| - 1)$, for $k = 6$, multiplications in the base field.

In the following, we consider the pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_1$ necessitates 256-bits elements, with an order q of size 256 and $\mathbb{G}_T$ needs to manipulate elements of size 1542 (with $k = 6$). Note that using the above results, we obtain that $P_{256} \approx 46S_{256}$

E General Result for Complexities

We here study the complexity of the main steps of our range proof described in Figure 2 (see Section 4.4).

E.1 Computation of the C_i's (prover)

The computation of $C_i = g^{x_i}h^{r_i}$, $\forall i \in [0, \ell]$, where each $x_i \in \{0, 1\}$, implies $\ell + 1$ modular exponentiations in the group $\mathbb{G}$ with a q-bits exponent, plus one more modular multiplication in $\mathbb{G}$ each time $x_i = 1$, which is, on average, one out of two times. This finally gives, using the results from Section D, $\frac{3q+1}{2}(\ell + 1)$ modular multiplications in $\mathbb{G}$ in total for this step.

Regarding the space complexity, the prover sends all the C_i's, and thus $(\ell + 1)|\mathbb{G}|$ bits in total.

E.2 Computation of $\tilde{C}$ (prover)

The prover and the verifier has to compute $\tilde{C} = \prod_{i=0}^{\ell} C_i^{b_i} = g^x h^{\sum_{i=0}^{\ell} b_i r_i} = g^x h^t$ iff $x = \sum_{i=0}^{\ell} x_i b_i$. As the prover knows x and can compute $t = \sum_{i=0}^{\ell} b_i r_i$, this step only needs one double modular exponentiation in $\mathbb{G}$ with a q-bits exponent. This gives in total $\frac{175q}{12}$ modular multiplications in $\mathbb{G}$.

The value $\tilde{C}$ needs not to be sent to the verifier and is consequently not considered in the space complexity.

E.3 Computation of $\tilde{C}$ (verifier)

For the computation of $\tilde{C} = \prod_{i=0}^{\ell} C_i^{b_i}$, the verifier can not use x since she does not know the secret. Thus, the computation of $\tilde{C}$ needs $q\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell+1}}$ modular multiplications in $\mathbb{G}$, using Shamir's trick (see Section D).

E.4 Computation of a proof Pok$(x_i, r_i : C_i = h^{r_i} \vee C_i/g = h^{r_i})$ (prover)

This proof necessitates 1 modular exponentiation in $\mathbb{G}$ with a q-bits exponent (for the correct predicate), 1 double modular exponentiation in $\mathbb{G}$ with q-bits exponents (for the wrong predicate), and, one time out of two, an additional modular multiplication for the case $x_i = 0$ (computation of C_i/g). This gives in total $\frac{275q+6}{12}$ modular multiplications in $\mathbb{G}$.

Regarding the space complexity, this proof necessitates 2 elements of $\mathbb{G}$ for the commitments and 4 elements in $\mathbb{Z}_q^*$ for the challenge (c_1 and c_2 such that the received challenge $c = c_i \oplus c_2$, see [20, 44] for details) and the response. In total, we have $2|\mathbb{G}| + 4q$ bits.

E.5 Verification of a proof $\mathbf{Pok}(x_i, r_i : C_i = h^{r_i} \vee C_i/g = h^{r_i})$ (verifier)

The verification of this proof necessitates 2 double exponentiations in $\mathbb{G}$ with q-bits exponents, and thus $\frac{175q}{6}$ modular multiplication in $\mathbb{G}$ and one additional modular multiplications in $\mathbb{G}$ (computation of C_i/g), which gives in total $\frac{175q}{6} + 1$ modular multiplications in $\mathbb{G}$.

E.6 Computation of the proof $\mathbf{Pok}(x, t : \tilde{C} = g^x h^t)$ (prover)

This proof necessitates 1 double modular exponentiation, and thus $\frac{175q}{12}$ modular multiplications in $\mathbb{G}$.

Regarding space complexity, this proof needs 1 element in $\mathbb{G}$ and 1 element in $\mathbb{Z}_q^*$, and thus $|\mathbb{G}| + q$ bits in total.

E.7 Verification of the proof $\mathbf{Pok}(x, t : \tilde{C} = g^x h^t)$ (verifier)

This proof necessitates 1 triple modular exponentiation, and thus $\frac{125q}{8}$ modular multiplications in $\mathbb{G}$.

F Complexity Considerations for our Signature-Based Solution

We finally study the complexity of our new method. This can be sum up as follows.

We denote by $\mathbf{m}$ the cost of the multiplication of two elements of $\mathbb{F}_q$.

- **Prover's time complexity**: the prover has to compute the following values: Com, B, C and D. She will thus have to compute 3 exponentiations in $\mathbb{G}$ and 3 interleaved exponentiations in $\mathbb{G}$ (with $n = 2$). Therefore she will have to perform $\frac{325}{4}|q|$ multiplications in $\mathbb{F}_q$. She will then have to compute the values involved in the following interactive ZKPK:

$$\begin{aligned} \Pi = \textsc{Pok}\big(&\alpha, \beta, \gamma, t, u, v, w : \\ &Com = g_2^{\alpha} g_3^{\beta} \wedge C = (C_1, C_2) = (g^u, B^t V^u) \wedge \\ &D = (D_1, D_2) = (g^w, B^v V^w) \wedge D_2 = C_2^{\alpha} V^{\gamma}\big) \end{aligned}$$

  More precisely, she will have to compute the witnesses and the answers corresponding to this ZKPK. $\mathcal{P}$ will therefore have to compute 6 witnesses which correspond to 2 exponentiations in $\mathbb{G}$ and 4 interleaved exponentiations in $\mathbb{G}$ (with $n = 2$). This implies $\frac{500}{6}|q|$ multiplications in $\mathbb{F}_q$. Finally, the computation of the answers implies 6 multiplications in $\mathbb{F}_q$. In conclusion, we obtain a total number of multiplications in $\mathbb{F}_q$ of:

$$\frac{1975}{12}|q| + 6$$

- **Verifier's time complexity**: We assume that there are l_v verifiers (in practice, l_v can be chosen equal to 3). Each verifier $\mathcal{V}_i$ will first have to verify the interactive ZKPK Π. These verifications will imply the computation of 2 exponentiations in $\mathbb{G}$ and 4 interleaved exponentiations in $\mathbb{G}$ (with $n = 3$). Thus $\frac{175}{2}|q|$ multiplications in $\mathbb{F}_q$. In Step (a) of the verification phase, $\mathcal{V}_i$ will compute 2 exponentiations in $\mathbb{G}$ and 3 additions in $\mathbb{G}$. Thus $25|q|$ +33 multiplications in $\mathbb{F}_q$. In Step (b) of the verification phase, $\mathcal{V}_i$ will have to compute 2 exponentiations in $\mathbb{G}$ (she will have to raise the ciphertext C to a random number $\alpha_i \in \mathbb{Z}_p$) and decrypt a ciphertext. $\mathcal{V}_i$ will thus have to compute 3 exponentiations in $\mathbb{G}$ and 1 subtraction in $\mathbb{G}$. Thus $\frac{75}{2}|q| + 12$ multiplications in $\mathbb{F}_q$. In Step (c) of the verification phase, V_i will perform similar computations than in Step (b). This implies $\frac{75}{2}|q| + 12$ multiplications in $\mathbb{F}_q$. In conclusion, we thus obtain a total number of multiplications in $\mathbb{F}_q$ of:

$$\big(\frac{375}{2}|q| + 57\big) l_v$$

- **Space complexity**: We tacitly assume that $\mathcal{V}$ publishes the signatures on the set elements once and for all. So we do not count them in the communication complexity. The prover sends 6 elements of $\mathbb{G}$ (namely Com, B, C and D) and 6 witnesses corresponding to the ZKPK Π. That is 12 elements of $\mathbb{G}$. Then the verifier sends the challenge of size $|\mathbb{Z}_q|$. Finally, the prover sends 7 elements (the answers) of size $|\mathbb{Z}_q|$. We thus obtain a space complexity of

$$12|\mathbb{G}| + 8|\mathbb{Z}_q|$$

– **Public key size**: The public key is composed of the elements g, g_1, g_2, g_3, V, Y, the V_i's and the Y_i's. There is also one BB signature without pairing for each element in the set Φ. Thus the size of the public key is

$$(2l_v + 3|\Phi| + 6)|\mathbb{G}| + 2|\Phi|q$$

G Complexity of Our Bit-by-Bit Method

In this appendix, we give the complete complexity study of our bit-by-bit method. We first focus on the FGreat algorithm, and next in the general method.

G.1 Complexity of the L part of the proof

We detail here the expected number of modular multiplications and exponentiations the prover (resp. the verifier) needs to perform during the proof of knowledge of the predicate L:

$$(x_{i_0} = 1) \vee [(x_{i_0} = 0) \wedge \text{FGreat}(\{a_0, \cdots, a_{i_0-1}\})] .$$

Prover's side. Two cases are possible, depending of the prover's secret:

– if $x_{i_0} = 1$, the prover proves this fact (which requires $25q/2$ modular multiplications) and simulates the rest of the proof (which requires $(175q + 12)i_0/12$ modular multiplications[4])
– if $x_{i_0} = 0$, the prover simulates the predicate $x_{i_0} = 1$ (which requires $(175q + 12)/12$ modular multiplications) and then proves that $x_{i_0} = 0$ (which requires $25q/2$ modular multiplications) and that $\text{FGreat}(\{a_0, \cdots, a_{i_0-1}\})$ is true.

Let us denote by m_i the number of modular multiplications in $\mathbb{G}$ implied by the $\text{FGreat}(\{a_0, \cdots, a_i\})$ algorithm, in average. We have:

$$\begin{aligned}
\mathsf{m}_i &= \begin{cases} \frac{25q}{2} + \mathsf{m}_{i-1} & \text{if } a_i = 1 \\ \frac{25q}{2} + \frac{175q+12}{12}(i-1) & \text{if } a_i = 0 \wedge x_i = 1 \wedge (\exists j < i \mid a_j = 1) \\ \frac{175q+12}{12} + \mathsf{m}_{i-1} & \text{if } a_i = 0 \wedge x_i = 0 \wedge (\exists j < i \mid a_j = 1) \\ 0 & \text{if } \forall j \leq i : \ a_i = 0 \end{cases} \\
&= \frac{1}{2}(\frac{25q}{2} + \mathsf{m}_{i-1}) + \left(1 - \frac{1}{2^i}\right)\left(\frac{1}{2}(\frac{25q}{2} + \frac{175q+12}{12}(i-1)) + \frac{1}{2}(\frac{175q+12}{12} + \mathsf{m}_{i-1})\right) \\
&\approx \frac{25q}{2} + \mathsf{m}_{i-1} + \frac{175q+12}{24}i \text{ by neglecting the } \frac{1}{2^i} \text{ factor} \\
&\approx \mathsf{m}_0 + \frac{25qi}{2} + \frac{175q+12}{24}\sum_{j=1}^{i} j
\end{aligned}$$

The last equation is due to the fact that $\mathsf{m}_1 = \frac{25q}{2} + \frac{175q+12}{24} + \mathsf{m}_0$, $\mathsf{m}_2 = \frac{25q}{2} + \frac{175q+12}{24} + \mathsf{m}_1 = \frac{25q}{2}2 + \frac{175q+12}{24}(1+2) + \mathsf{m}_0$, etc. We have moreover that $\mathsf{m}_0 = \frac{1}{2} \times 0 + \frac{1}{2} \times \frac{25q}{2} = \frac{25q}{4}$. This gives the following result:

$$\mathsf{m}_i \approx \left(\frac{175q+12}{48}\right)i^2 + \left(\frac{775q+12}{48}\right)i + \left(\frac{25q}{4}\right).$$

Recall that in average we have $i_0 = \ell - 1$, we obtain for the whole complexity of the L part of the proof:

$$\begin{aligned}
Lm &\approx \frac{1}{2}(\frac{25q}{2} + \frac{(175q+12)(\ell-1)}{12}) + \frac{1}{2}(\frac{25q}{2} + \frac{175q+12}{12} + m_{\ell-2}) \\
&\approx \frac{25q}{2} + \frac{(175q+12)\ell}{12} + \frac{m_{\ell-2}}{2} \\
&\approx \frac{175q+12}{96}\ell^2 + \frac{1475q+60}{96}\ell + \frac{325q+12}{48}
\end{aligned}$$

[4] Note that similarly to the result presented in Appendix B, we here consider that in average $a_0 = 0$ and $a_1 = 1$. Consequently the predicate outputted by $\text{FGreat}(\{a_0, \cdots, a_{i_0-1}\})$ contains in average $i_0 - 1$ clauses.

Verifier's side. Let us denote by m_i the number of modular multiplications in $\mathbb{G}$ the prover has to do implied by the FGreat algorithm, in average. We have:

$$\begin{aligned}\mathsf{m}_i &= \frac{175q+12}{12} + \mathsf{m}_{i-1} \\ &= \Big(\frac{175q+12}{12}\Big)i + \Big(\frac{175q+12}{24}\Big)\end{aligned}$$

since this is an arithmetic progression with $\mathsf{m}_0 = \frac{1}{2} \times 0 + \frac{1}{2} \times \frac{175q+12}{12} = \frac{175q+12}{24}$. Thus, to verify the L part of the proof, the verifier will performs the following number of modular multiplications:

$$\begin{aligned}\mathsf{m}_i &= \Big(\frac{175q+12}{12}\Big)\ell + \Big(\frac{175q+12}{24}\Big) \\ &= \frac{175q+12}{24}(2\ell+1)\end{aligned}$$

Space Complexity Concerning the space complexity, the prover has to send one element of $\mathbb{G}$ and two elements of q bits (a sub challenge and an answer) for each elements of the predicate. The list $\tilde{L}$ contains in average $i_0 - 1$, thus the global list L contains in average $i_0 + 1 = \ell$ elements. Consequently the space complexity required by this part of the proof is:

$$(|\mathbb{G}| + 2q)\ell$$

G.2 General Complexity Study

We are now ready to study in details the complexity of our method, using the results given in Appendix B, Section G.1 and above.

- **Prover's time complexity**: the first step includes the computation of the commitments C_i for all $i \in [0, \ell]$, and thus it total $\frac{25q+1}{2}(\ell+1)$ modular multiplications in $\mathbb{G}$. Next, the computation of $\tilde{C}$ necessitates $\frac{175q}{12}$ modular multiplications in $\mathbb{G}$. After that, the proof concerning the predicate $x_i = 1 \vee x_i = 0$ is done i_0 times and each occurrence necessitates $\frac{325q+6}{12}$ modular multiplications in $\mathbb{G}$. Using the result given in Appendix B, we have $i_0 = \ell - 1$. Next, the proof concerning $\tilde{C}$ necessitates $\frac{175q}{12}$ modular multiplications in $\mathbb{G}$. Finally, the proof concerning L is given by the result explained in Section G.1. This gives in total

$$\Big(\frac{175q+12}{96}\Big)\ell^2 + \Big(\frac{5275q+156}{96}\Big)\ell + \Big(\frac{1025q+12}{48}\Big) \text{ modular multiplications in } \mathbb{G}.$$

- **Verifier's time complexity**: from the verifier's point of view, the computation of $\tilde{C}$ necessitates $\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell+1}}q$ modular multiplications in $\mathbb{G}$. After that, the proof concerning the predicate $x_i = 1 \vee x_i = 0$ is done i_0 times and each occurrence necessitates $\frac{175q+6}{6}$ modular multiplications in $\mathbb{G}$, with, again, $i_0 = \ell - 1$. Next, the proof concerning $\tilde{C}$ necessitates $\frac{125q}{8}$ modular multiplications in $\mathbb{G}$. Finally, the proof concerning L is given by the result explained in Section G.1, that is $\frac{175q+12}{24}(2\ell+1)$ modular multiplications in $\mathbb{G}$. This gives in total

$$\Big(\frac{525q+24}{12}\Big)\ell - \Big(\frac{75q+6}{12}\Big) + \Big(\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell+1}}\Big)q \text{ modular multiplications in } \mathbb{G}.$$

- **Space complexity**: regarding space complexity, the prover has to send the C_i's, and thus $(\ell+1)|\mathbb{G}|$ bits. Next, the proof concerning the predicate $x_i = 1 \vee x_i = 0$ necessitates $(2|\mathbb{G}| + 4q)i_0$ bits with $i_0 = \ell - 1$. After that, the proof concerning $\tilde{C}$ necessitates $|\mathbb{G}| + q$ bits. Finally, the proof concerning L is given by the result explained in Section G.1, that is $(|\mathbb{G}| + 2q)\ell$ bits. This gives in total

$$(4|\mathbb{G}| + 6q)\ell - 3q \text{ bits.}$$

- **Setup time complexity**:
  The setup corresponds to the computation of h, thus $25q/2$ modular multiplications, to the determination of i_0 and to the composition of the List L. The two last procedures can be neglected compared to the computation of h. This gives in total $\frac{25q}{2}$ modular multiplications in $\mathbb{G}$.

– **Public key size**:
The public key is composed of g, h, i_0 and the order of the group. This gives in total $2|\mathbb{G}| + \ell + q$ bits.

H The Double Binary Representation Method

We here describe the double binary representation method which is due to Schoenmakers [46, 47]. The idea behind is to use Lemma 5 and thus twice a proof that some secret belongs to an interval of the form $[0, 2^k[$ from Bellare and Goldwasser [3].

H.1 Focus on the proof that x belongs to $[0, 2^k[$

In fact, this proof is similar to the one described in Section 4.1 (see Figure 1, except the case $x \geq a$ in the proof U_1) with $v = 2^k - 1$. We thus have $x = [x_0, \cdots, x_{k-1}]_2$. Next, the prover makes k commitments $C_i = g^{x_i} h^{r_i}$, compute $\tilde{C} = \prod_{i=0}^{k-1} C_i^{2^i} = g^x h^{\sum_{i=0}^{k-1} r_i 2^i}$ and next makes the proof U_1 as described in Figure 1, except the case $x \geq a$.

H.2 Description of the Method

Using the same notation as Lemma 5, we have $B = b - a + 1$, $X = x - a$ and k the unique integer such that $2^k \leq B \leq 2^{k+1}$. We also denote $B_0 = 2^{k+1} - B$ and $Y = x - a + B_0$. Then $x \in [a, b]$ if and only if $X \in [0, 2^{k+1}[$ and $Y \in [0, 2^{k+1}[$. The range proof can next be described as follows.

– The prover first produces a commitment C_X (resp. C_Y) on X (resp. Y). Note that a, b, k, B and B_0 are known or computable by both the prover and the verifier. The next result is easy, remarking that $C_X g^a = g^x h^{r_X}$ (resp. $C_Y g^{a-B_0} = g^x h^{r_Y}$).
– Using twice the proof of binary decomposition of a secret described above, the prover can now prove that X and Y belong to $[0, 2^{k+1}[$.

The global proof that $x \in [a, b]$ is next

$$\begin{array}{c}\textsc{Pok}(x, X, Y, r_X, r_Y : C_X g^a = g^x h^{r_X} \wedge \\ C_Y g^{a-B_0} = g^x h^{r_Y} \wedge X \in [0, 2^{k+1}[\wedge Y \in [0, 2^{k+1}[)\end{array}$$

H.3 Complexity study

We now study the complexity of this solution, using the results given in Section E twice (once for each range proofs in $[0, 2^k[$).

– **Prover's time complexity**: the first step includes the computation of the two commitments C_X and C_Y, and thus 2 double modular exponentiations in $\mathbb{G}$ with a q-bit exponent. In the second step, the zero-knowledge proof of knowledge necessitates 2 double modular exponentiations in $\mathbb{G}$ with a q-bit exponent for the proof regarding $C_X g^a$ and $C_Y g^{a-B_0}$, and next twice $\frac{3q+1}{2}(k+1) + \frac{175q}{12} + \frac{275q+6}{12}\ell + \frac{175q}{12}$ for the range proofs. We here remember that the integer k is such that $2^k \leq b - a \leq 2^{k+1}$ and we can thus take $k = \ell$. This gives in total

$$\frac{293q + 12}{6}\ell + \frac{359q + 3}{3} \text{ modular multiplications in } \mathbb{G}.$$

– **Verifier's time complexity**: the verifier has to verify the whole proof of knowledge, that is the one corresponding to $C_X g^a$ and $C_Y g^{a-B_0}$ (corresponding to 2 triple modular exponentiations in $\mathbb{G}$ with a q-bits exponent) and next twice the range proof ($q\frac{25}{3}\frac{2^{\ell+2}-1}{2^{\ell+1}} + \frac{175q+6}{6}\ell + \frac{125q}{8}$). This gives in total

$$\frac{175q + 6}{3}\ell + \frac{125q}{2} + \frac{25}{3}\frac{2^{\ell+2} - 1}{2^\ell}q \text{ modular multiplications in } \mathbb{G}.$$

– **Space complexity**: regarding space complexity, the prover has to send the C_i's corresponding to both X and Y and the whole proof of knowledge, that is,

$$\begin{array}{l}2(\ell+1)|\mathbb{G}| + 2(|\mathbb{G}| + q) + 2\ell(2|\mathbb{G}| + 4q) + 2(|\mathbb{G}| + q) \\ = (6|\mathbb{G}| + 8q)\ell + (6|\mathbb{G}| + 4q) \text{ bits}\end{array}$$

– **Setup time complexity**: the setup procedure is mostly the computation of g^a and g^{a-B_0} and thus

$$3q \text{ modular multiplications in } \mathbb{G}.$$

– **Public key size**: the public key is composed of the elements $(a, b, q, k, g, h, g^a, g^{a-B_0})$ and thus

$$4(|\mathbb{G}| + q) \text{ bits.}$$

I The Multi-base Decomposition Method

This method is the one due to Lipmaa, Asokan and Niemi [36], based on the work from Damgård and Jurik [23]. This method is based on a variant of the multi-base decomposition Lemma 7, which says that $x \in [a, b]$ if and only if $x = \sum_{i=-1}^{\ell} b_i x_i$ where for all $i \in [0, \ell]$, $x_i \in \{0, 1\}$ and $b_i = \lfloor (b - a + 2^{\ell-i})/2^{\ell-i+1} \rfloor$ and $b_{-1} = a$.

I.1 Description of the Method

Using the above result, the range proof is the following one.

1. The prover first randomly chooses $r_{-1}, \ldots, r_\ell \in_R \mathbb{Z}_q$ and computes $C_i = g^{x_i} h^{r_i}$ for all $i \in [-1, \ell]$. These C_i's are next sent to the verifier.
2. Both the prover and the verifier can compute $\tilde{C} = \prod_{i=-1}^{\ell} C_i^{b_i}$, which is equal to $g^x h^{\sum_{i=-1}^{\ell} b_i r_i}$. In the following, we denote $t = \sum_{i=-1}^{\ell} b_i r_i$.
3. The prover and the verifier then play the following interactive ZKPK

$$\begin{aligned} U = \textsc{Pok}(x, t, r_{-1}, \ldots, r_\ell : (C_{-1} = h^{r_{-1}} \vee C_{-1}/g = h^{r_{-1}}) \wedge \ldots \wedge \\ (C_\ell = h^{r_\ell} \vee C_\ell/g = h^{r_\ell}) \wedge \tilde{C} = g^x h^t) \end{aligned}$$

I.2 Complexity study

We now study the complexity of this solution, using the results given in Section E.

– **Prover's time complexity**: the first step includes the computation of the $\ell+2$ C_i's ($\frac{175q}{12}(\ell+2)$), the computation of $\tilde{C}$ ($\frac{125q}{12}$) and the whole proof of knowledge ($\frac{275q+6}{12}(\ell+1)$ for the C_i's and $\frac{175q}{12}$ for $\tilde{C}$). This gives in total

$$\frac{75q+1}{2}\ell + \frac{325q+2}{4} \text{ modular multiplications in } \mathbb{G}.$$

– **Verifier's time complexity**: the verifier has to compute $\tilde{C}$ ($\frac{25}{3}\frac{2^{\ell+3}-1}{2^{\ell+2}}q$) and verify the whole proof of knowledge ($\frac{175q+6}{6}(\ell+2)$ for the C_i's and $\frac{125q}{8}$ for $\tilde{C}$). This gives in total

$$\frac{175q+6}{6}\ell + \frac{1775q+48}{24} + \frac{25}{3}\frac{2^{\ell+3}-1}{2^{\ell+2}}q \text{ modular multiplications in } \mathbb{G}.$$

– **Space complexity**: regarding space complexity, the prover has to send the C_i's ($(\ell+2)|\mathbb{G}|$) and the whole proof of knowledge ($(2|\mathbb{G}| + 4q)(\ell+1)$ for the C_i's and $|\mathbb{G}| + q$ for $\tilde{C}$), that is in total

$$(3|\mathbb{G}| + 4q)\ell + (5|\mathbb{G}| + 5q) \text{ bits}$$

– **Setup time complexity**: the setup procedure is mostly the computation of the b_i's, which is quite immediate. We thus consider that there is no computation for this phase.
– **Public key size**: the public key is composed of the elements $(a, b, q, g, h, b_{-1}, \cdots, b_\ell)$ and thus

$$2|\mathbb{G}| + (\ell+5)q \text{ bits.}$$

J Square Decomposition Method

We now describe the method based on the decomposition in the sum of squares to prove that a secret is positive, as it is described in Section 2.2.

J.1 Description of the Method

In minute details, to ensure the zero-knowledge property, the prover computes two commitments $C_X = g^X h^{r_1}$ and $C_Y = g^Y h^{r_2}$ and proves that both committed values are positive integers. Finally, the prover also had to prove that both values are well-formed from x. The global proof of knowledge can be written as follows, where $C = g^x h^r$ is a commitment on the secret.

$$\text{PoK}\big(x, X, Y, r, r_1, r_2 : C = g^x h^r \wedge C_X = g^X h^{r_1} \wedge C_Y = g^Y h^{r_2}$$
$$\wedge C_X g^a = g^x h^{r_1} \wedge C_Y g^{-b} = g^{-x} h^{r_2} \wedge X > 0 \wedge Y > 0\big)$$

In the following, we only describe some existing manner to prove that an integer is positive. Note that l_c and l_s are security parameters and ℓ the bit-size of the secret.

Lipmaa's Method The solution proposed by Lipmaa in [35] relies on the Lagrange theorem (see lemma 2) stating that any positive integers can be represented by the sum of four squares. To prove that an integer X is positive, the prover first have to define four integers x_1, x_2, x_3, x_4 such that $X = \sum x_i^2$ (*e.g.* by using the Rabin-Shallit algorithm [43]). Then, the prover realizes the following interactive proof of knowledge.

1. The prover chooses at random $r_1, \cdots, r_4 \in \mathbb{Z}_{2^{l_{\mathbb{G}}+l_s}}$ such that $\sum r_i = r$, $m_1, \cdots, m_4 \in \mathbb{Z}_{2^{l_s+l_e+\ell/2}}$, $\bar{r}_1, \cdots, \bar{r}_4 \in \mathbb{Z}_{2^{l_{\mathbb{G}}+2l_s+l_e}}$ and $\tilde{r} \in \mathbb{Z}_{2^{l_{\mathbb{G}}+2l_s+l_e+\ell/2}}$.
2. The prover computes $C_i = g^{x_i} h^{r_i}$ for all $i \in \{1, \ldots, 4\}$, $\bar{C} = g^{\sum_i m_i} h^{\sum_i \bar{r}_i}$ and $\tilde{C} = \left(\prod_i C_i^{m_i}\right) h^{\tilde{r}}$.
3. The prover sends $(C_1, C_2, C_3, C_4, \bar{C}, \tilde{C})$ to the verifier.
4. The verifier chooses at random $e \in \{0,1\}^{l_e}$ and sends it to the prover.
5. The prover computes for all $i \in \{1, \ldots, 4\}$ and $\bar{m}_i = m_i + e x_i$, $\hat{r}_i = \bar{r}_i + e.r_i$. It also computes $R = \tilde{r} + e\sum(1 - x_i) r_i$ and sends to the verifier $(\bar{m}_1, \bar{m}_2, \bar{m}_3, \bar{m}_4, \hat{r}_1, \hat{r}_2, \hat{r}_3, \hat{r}_4, R)$.
6. The verifier checks that $\prod_i g^{\bar{m}_i} h^{\hat{r}_i} C_i^{-e} = \bar{C}$ and $h^R C_X^{-e} \prod_i C_i^{\bar{m}_i} = \tilde{C}$.

Groth's Method In [29], Groth has proposed a variant based on Lemma 3, which states that some numbers can be written as the sum of three squares. In fact, it does not exist any X such that $4X + 1$ is of the form $4^n(8k + 7)$. Thus it does not exist X such that it can not be written as a sum of three squares. Then Groth proves that X is a positive integer by proving that $4X + 1$ can be written as the sum of three squares. Again the prover has first to compute these integers, denoted x_1, x_2, x_3, such that $4X + 1 = \sum x_i^2$. Then, the prover realizes the following interactive proof of knowledge.

1. The prover chooses at random $r_X, r_1, r_2, r_3 \in \{0,1\}^{\ell+l_e+l_s}$ and computes $\Delta = 4r_X - 2x_1 r_1 - 2x_2 r_2 - 2x_3 r_3$.
2. Then the prover chooses at random $r \in \{0,1\}^{l_{\mathbb{Z}_n}}$ and $r_r \in \{0,1\}^{l_{\mathbb{Z}_n}+l_e+l_s}$. The prover computes and sends $C = g_1^X g_2^{x_1} g_3^{x_2} g_4^{x_3} g_5^{\Delta} h^r$ and $C_r = g_1^{r_X} g_2^{r_1} g_3^{r_2} g_4^{r_3} g_5^{-r_1^2 - r_2^2 - r_3^2} h^{r_r}$.
3. The verifier chooses at random $e \in \{0,1\}^{l_e}$ and sends it to the prover.
4. The prover computes $s_X = r_X + eX$, $s_r = r_r + er$ and $s_i = r_i + e x_i$ for $i = 1, 2, 3$. He sends $(s_X, s_r, s_1, s_2, s_3)$ to the verifier.
5. The verifier first computes $\tilde{\Delta} = e(4s_X + e) - s_1^2 - s_2^2 - s_3^2$ and then checks if $C^e C_r = g_1^{s_X} g_2^{s_1} g_3^{s_2} g_4^{s_3} g_5^{\tilde{\Delta}} h^{s_r}$.

J.2 Complexity Considerations

Lipmaa's Method

– **Prover's time complexity**: concerning the proof that values X and Y are well-formed from x, the prover has to compute 5 interleaved modular exponentiations (with $n = 2$) of $l_{\mathbb{Z}_n} + l_e + l_s$-bits exponent, thus $7/4(l_{\mathbb{Z}_n} + l_e + l_s)$ modular multiplications. Concerning one "positive proof", the prover has to compute $4.C_i$ (each implying $7/4(l_{\mathbb{Z}_n} + l_s)$ modular multiplications), $\bar{C}$ (implying $7/4(l_{\mathbb{Z}_n} + l_s + l_e)$ modular multiplications) and $\tilde{C}$ (implying $63/32(l_{\mathbb{G}} + 2l_s + l_e + \ell/2)$ modular multiplications). In conclusion, considering that the "positive proof" as to be done twice, we thus obtain a total number of modular multiplications of:

$$\begin{aligned}
&\frac{35}{4}(l_{\mathbb{Z}_n} + l_e + l_s) + 14(l_{\mathbb{Z}_n} + l_s) + \frac{7}{2}(l_{\mathbb{Z}_n} + l_s + l_e) + \frac{63}{16}(l_{\mathbb{G}} + 2l_s + l_e + \ell/2) \\
&= \frac{483}{16} l_{\mathbb{Z}_n} + \frac{259}{16} l_e + \frac{273}{8} l_s + \frac{63}{32}\ell \\
&\approx 30.2 l_{\mathbb{Z}_n} + 16.2 l_e + 34.1 l_s + 2\ell
\end{aligned}$$

– **Verifier's time complexity**: concerning the proof that values X and Y are well-formed from x, the verifier has to compute 5 interleaved modular exponentiations (with $n = 3$) of $l_{\mathbb{Z}_n} + l_e + l_s$-bits exponent, thus $15/8(l_{\mathbb{Z}_n} + l_e + l_s)$ modular multiplications. Concerning one "positive proof", the verifier has to compute $\prod_i g^{\bar{m}_i} h^{\hat{r}_i} C_i^{-e} (= g^{\sum \bar{m}_i} h^{\sum \hat{r}_i} (\prod_i C_i)^{-e})$ and $h^R C_X^{-e} \prod_i C_i^{\bar{m}_i}$. Thus, it implies $15/8(l_{\mathbb{Z}_n} + l_e + l_s) + 3$ modular multiplications for the former computation, and $127/64(l_{\mathbb{Z}_n} + 2l_s + l_e + \ell/2)$ for the latter. In conclusion, considering that the "positive proof" as to be done twice, we thus obtain a total number of modular multiplications of:

$$\begin{aligned} &\frac{75}{8}(l_{\mathbb{Z}_n} + l_e + l_s) + \frac{15}{4}(l_{\mathbb{Z}_n} + l_e + l_s) + \frac{127}{32}(l_{\mathbb{Z}_n} + 2l_s + l_e + \ell/2) + 6 \\ =\ &\frac{547}{32} l_{\mathbb{Z}_n} + \frac{547}{32} l_e + \frac{337}{16} l_s + \frac{127}{64}\ell + 6 \\ \approx\ &17.1 l_{\mathbb{Z}_n} + 17.1 l_e + 21.1 l_s + 2\ell + 6 \end{aligned}$$

– **Space complexity**: in the first step of the interactive protocol, the prover sends 11 elements of $\mathbb{G}$. The verifier then sends the challenge of size l_e. Finally, the prover sends 5 elements of $l_{\mathbb{Z}_n} + l_e + l_s$ bits, 4 of $l_s + l_e + \ell/2$ bits, 4 of $l_{\mathbb{G}} + 2l_s + l_e$ bits and one of $l_{\mathbb{G}} + 2l_s + l_e + \ell/2$ bits. We thus obtain a space complexity of

$$\begin{aligned} &11|\mathbb{G}| + l_e + 5(l_{\mathbb{Z}_n} + l_e + l_s) + 4(l_s + l_e + \ell/2) + 4(l_{\mathbb{G}} + 2l_s + l_e) + l_{\mathbb{G}} + 2l_s + l_e + \ell/2 \\ =\ &11|\mathbb{G}| + 10 l_{\mathbb{Z}_n} + 15 l_e + 19 l_s + \frac{5}{2}\ell \ \text{ bits} \end{aligned}$$

– **Setup time complexity**: the setup complexity of the system only concerns the computation of h, implying one modular exponentiation of a $l_{\mathbb{Z}_n}$-bits exponent.
The setup procedure of a prover is more complex, as he has to decompose her secrets in sums of squares. Using the Rabin-Shallit algorithm, the decomposition of X (and Y) in a sum of four squares has a complexity of $O(l^4)$. The other operations (computations of C, C_X, C_Y) can be neglected in regard of this complexity.
– **Public key size**: the public key is composed of the base elements (g, h), the bit-length $l_{\mathbb{Z}_n}$ of the order and the security parameters l_s, l_e. Thus, the size of the public key is $2|\mathbb{G}| + l_{\mathbb{Z}_n} + l_s + l_e$ bits.

Groth's Method

– **Prover's time complexity**: concerning one "positive proof", the prover has to compute C (implying $127/64(\ell + l_e + l_s + 1)$ modular multiplications) and C_r (implying $127/64(\ell + l_e + l_s + 1)$ modular multiplications). In conclusion, considering that the "positive proof" as to be done twice and that the previous part of the proof is the same as in Lipmaa's method, we thus obtain a total number of modular multiplications of:

$$\begin{aligned} &\frac{35}{4}(l_{\mathbb{Z}_n} + l_e + l_s) + \frac{127}{16}(\ell + l_e + l_s + 1) \\ =\ &\frac{35}{4} l_{\mathbb{Z}_n} + \frac{267}{16} l_e + \frac{267}{16} l_s + \frac{127}{16}\ell + \frac{127}{16} \\ \approx\ &8.75 l_{\mathbb{Z}_n} + 16.7 l_e + 16.7 l_s + 7.9\ell + 7.9 \end{aligned}$$

– **Verifier's time complexity**: concerning one "positive proof", the verifier has to compute $C^e C_r$ (implying $3/2 l_e + 1$) modular multiplications) and $g_1^{s_X} g_2^{s_1} g_3^{s_2} g_4^{s_3} g_5^{\tilde{\Delta}} h^{s_r}$ (implying $127/64(\ell + l_e + l_s + 1)$ modular multiplications). In conclusion, considering that the "positive proof" as to be done twice and that the previous part of the proof is the same as in Lipmaa's method, we thus obtain a total number of modular multiplications of:

$$\begin{aligned} &\frac{75}{8}(l_{\mathbb{Z}_n} + l_e + l_s) + 3l_e + 2 + \frac{127}{32}(\ell + l_e + l_s + 1) \\ =\ &\frac{75}{8} l_{\mathbb{Z}_n} + \frac{523}{32} l_e + \frac{427}{32} l_s + \frac{127}{32}\ell + \frac{191}{32} \\ \approx\ &9.4 l_{\mathbb{Z}_n} + 16.3 l_e + 13.3 l_s + 4\ell + 6 \end{aligned}$$

– **Space complexity**: in the first step of the interactive protocol, the prover sends 7 elements of $\mathbb{G}$. The verifier then sends the challenge of size l_e. Finally, the prover sends 5 elements of $l_{\mathbb{Z}_n} + l_e + l_s$ bits, 4 of $\ell + l_e + l_s$ bits and one of $l_{\mathbb{G}} + l_s + l_e$ bits. We thus obtain a space complexity of

$$\begin{aligned} &7|\mathbb{G}| + l_e + 5(l_{\mathbb{Z}_n} + l_e + l_s) + 4(\ell + l_e + l_s) + l_{\mathbb{G}} + l_s + l_e \\ =\ &7|\mathbb{G}| + 6 l_{\mathbb{Z}_n} + 11 l_e + 10 l_s + 4\ell \ \text{ bits} \end{aligned}$$

– **Setup time complexity**: the setup complexity of the system concerns the computation of $h, g_1, g_2, g_3, g_4, g_5$ implying 6 modular exponentiations of $l_{\mathbb{Z}_n}$-bits exponents.
Again, the setup procedure for a prover is mainly the decomposition of her secrets in sums of squares. Using the Rabin-Shallit algorithm, the decomposition of $4X+1$ (and $4Y+1$) in a sum of three squares has a complexity of $O(\ell^4)$. The other operations (computations of C, C_X, C_Y and the public keys) can be neglected in regard of this complexity.
– **Public key size**: the public key is composed of the base elements $(g, h, g_1, g_2, g_3, g_4, g_5)$, the bit-length $l_{\mathbb{Z}_n}$ of the order and the security parameters ℓ_s, ℓ_e. Thus, the size of the public key is $7|\mathbb{G}| + \ell_{\mathbb{Z}_n} + \ell_s + \ell_e$ bits.

K Signature-based Method

In [11], Camenisch, Chaabouni and Shelat first describe a set membership proof, based on the work of Teranishi and Sako [48]. Camenisch *et al.* next propose a new range proof, based on this set membership protocol.

K.1 The Set Membership Proof and its Complexity

The set membership proof described in [48, 11] is based on the publication of a BB signature (with pairing) of each element of the set Φ. The proof next consists in producing a proof of knowledge of a secret element and a signature on this element, without revealing which signature is used.

We consider a designated authority owning a BB signature secret key γ, while the corresponding public key is $w = g^\gamma$ (see Section A.3 for details). We consider a set Φ containing several elements. Each element $i \in \Phi$ is next signed by the given authority, as $A_i = g^{1/(\gamma+i)}$ and each couple (i, A_i) is published.

Let us consider a prover having a secret $i \in \Phi$. She first commits on i, as $C = g^i h^r$, and next proves that $i \in \Phi$ by executing the following steps.

1. she picks at random $v \in \mathbb{Z}_q^*$ and computes $V = A_i^v$
2. she generates the proof of knowledge

$$\textsc{Pok}(i, r, v : C = g^i h^r \wedge e(V, w) = e(V, g)^{-i} e(g, g)^v).$$

The verification of this proof is simply the verification of the above proof of knowledge.

Next, the complexity is quite simple to study.

– **Prover's time complexity**: the prover first computes V and next the proof of knowledge, which consists in two double modular exponentiations (computation of both $g^s h^m$ and $V^{-s} g^t$, where s, m and t are random numbers) and one pairing evaluation (that is $e(V^{-s} g^t, g)$), plus 3 modular multiplications. In conclusion, we obtain a total number of multiplications in $\mathbb{F}_q$ of:

$$\frac{1817}{3}|q| + 3.$$

– **Verifier's time complexity**: the verifier has simply to check the proof of knowledge, and consequently computes a triple modular exponentiation (concerning the proof verification on C), one pairing, a double modular exponentiation and one modular exponentiation (since $e(g, g)$ can be a public parameter). In conclusion, we thus obtain a total number of multiplications in $\mathbb{F}_q$ of:

$$\frac{14561}{24}|q| + 1.$$

– **Space complexity**: The proof consists in the element $V \in \mathbb{G}$ and the proof of knowledge, and thus, a space complexity of

$$2|\mathbb{G}| + |\mathbb{G}_T| + 4q.$$

– **Public key size**: The public key is composed of the elements g, h, w, the $|\Phi|$ signatures and the element $e(g, g)$. Thus the size of the public key is

$$(3 + |\Phi|)|\mathbb{G}| + |\mathbb{G}_T|$$

K.2 Description of the Range Proof Method

The range proof described in [11] next relies on the Schoenmakers's double u-ary representation Lemma 4 to replace the range proof that $x \in [a, b]$ by two range proofs : $x_b = x - b - u^{k+1} \in [0, u^{k+1}[$ and $x_a = x - a \in [0, u^{k+1}[$ with k such that $u^k < b < u^{k+1}$.

Thus they prove whether a secret σ belongs to a known interval of the form $[0, u^{k+1}[$ or not. The idea is to decompose σ in base u to obtain $\sigma = [\sigma_0, \cdots, \sigma_k]_u$ and then to use a signature Based Characterization of each digit σ_j in the interval $[0, u]$ (cf Lemma 6). Finally the prover proves that each digit is well composed and that they are the digits of the committed value.
This protocol uses two multiplicative groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order q such that it exists an admissible bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and in which the $\mathfrak{q}$-SDH problem is hard. The prover commits σ in $C = g^\sigma h^r$. From now, g, h, u, k and C are considered common input. The interactive proof of knowledge corresponds to the following steps:

1. The verifier chooses at random ρ in $\mathbb{Z}_q$ and sends to the prover $y = g^\rho$ and $\{A_i = g^{\frac{1}{\rho+i}}\}_{i\in[0,u[}$.
2. The prover randomly chooses $v_j \in [0, k]$, for all $j \in [0, k]$ and sends to the prover the set $\{V_j = A_{\sigma_j}^{v_j}\}_{j\in[0,k]}$.
3. The prover picks $s_j, t_j, m_j \in \mathbb{Z}_q$ at random for all $j \in [0, k]$ and sends to the verifier the set $\{a_j = e(V_j^{-s_j} \cdot g^{t_j}, g)\}_{j\in[0,k]}$ and $D = g^{\sum_j u^j s_j} \cdot h^{\sum_j m_j}$.
4. The verifier sends a random challenge $c \in \mathbb{Z}_q$.
5. The prover computes $z_{\sigma_j} = s_j - \sigma_j \cdot c$ and $z_{v_j} = t_j - v_j \cdot c$ for all $j \in [0, k]$ and $z_\rho = \sum_{j=0}^k m_j - \rho \cdot c$. Then he sends both sets $\{z_{\sigma_j}\}_{j\in[0,k]}$, $\{z_{v_j}\}_{j\in[0,k]}$ and z_ρ to the prover.
6. The verifier checks if $D = C^c \cdot h^{z_\rho} \cdot g^{\sum_{j=0}^k u^j \cdot z_{\sigma_j}}$ and if $\forall j \in [0, k]$, $a_j = e(V_j, y^c \cdot g^{-z_{\sigma_j}}) \cdot e(g, g)^{z_{v_j}}$.

Note that the two first Steps are independent of the secret σ, thus they will be done only once for the two proofs. The common verification equation sketched in [11] to simplify the last step in the case of two proofs has problems, thus it will not be used in our study.

K.3 Complexity Considerations for the Range Proof

– **Prover's time complexity**: the prover first commits both secrets and thus computes 2 interleaved modular exponentiations in $\mathbb{G}$ (with $n = 2$) (thus $\frac{175}{6}|q|$ multiplications in $\mathbb{F}_q$). Concerning the proof, the verifier has to compute the V_i only once which implies $k + 1$ modular exponentiations in $\mathbb{G}$ (thus $\frac{25(k+1)}{2}|q|$ multiplications in $\mathbb{F}_q$). From this point, the proof is done twice. The prover computes the a_j thus $2(k+1)$ interleaved modular exponentiations in $\mathbb{G}$ (with $n = 2$) and $2(k+1)$ pairings (thus $\frac{175(k+1)}{6}|q| + 2(k + 1)575(|q| - 1)$ multiplications in $\mathbb{F}_q$), then the two D which imply $2k$ multiplications in $\mathbb{G}$ and 2 interleaved modular exponentiation in $\mathbb{G}$ (with $n = 2$) (thus $25k|q|+\frac{175|q|}{6}$ multiplications in $\mathbb{F}_q$), finally the z_{σ_j}, z_{v_j} and z_ρ implying $2(2(k+1)+1)$ multiplication in $\mathbb{Z}_q$ (thus $4k+6$ multiplications in $\mathbb{F}_q$). In conclusion, we obtain a total number of multiplications in $\mathbb{F}_q$ of:

$$\begin{aligned} &1250|q| + \frac{200}{3}k|q| - 1148 \\ \approx\ &1250|q| + 66.7k \cdot |q| - 1148 \end{aligned}$$

– **Verifier's time complexity**: concerning the proof the verifier has to compute y (thus $\frac{25}{2}q$ multiplications in $\mathbb{F}_q$), then he computes the A_i which imply u modular exponentiations in $\mathbb{G}$ and u inversion in $\mathbb{Z}_q$ (thus $\frac{25u}{2}q + 10u$ multiplications in $\mathbb{F}_q$). Finally the verifications equations are computed twice (once for x_a and once for x_b), they imply thus two interleaved modular exponentiations in $\mathbb{G}$ (with $n = 3$) and $2k$ multiplications in $\mathbb{Z}_q$ for D (thus $\frac{125q}{4} + 2k$ multiplications in $\mathbb{F}_q$) and $2(k + 1)$ interleaved modular exponentiations (with $n = 2$) in $\mathbb{G}$, $2(k + 1)$ modular exponentiations in $\mathbb{G}_T$ and $2(k + 1)$ pairings for the a_j (thus $\frac{7075}{6}k + 540|q| - 1150$ multiplications in $\mathbb{F}_q$). In conclusion, we thus obtain a total number of multiplications in $\mathbb{F}_q$ of:

$$\begin{aligned} &\frac{2335}{4}|q| + \frac{25}{2}u|q| + 10u + \frac{7087}{6}k - 1150 \\ \approx\ &583.8|q| + 12.5u|q| + 10u + 1181k - 1150 \end{aligned}$$

– **Space complexity**: in the first step of the interactive protocol, the verifier sends $u+1$ elements of $\mathbb{G}$. The prover answers by $k+1$ elements of $\mathbb{G}$. From that point the protocol is done twice. The prover sends $k+1$ elements of $\mathbb{G}_T$ and one element of $\mathbb{G}$. Then the verifier sends the challenge of size $|\mathbb{Z}_q|$. Finally, the prover sends $2k+3$ elements of size $|\mathbb{Z}_q|$. We thus obtain a space complexity of

$$(u+k+5)|\mathbb{G}| + (2k+2)|\mathbb{G}_T| + (4k+8)|\mathbb{Z}_q|$$

– **Setup time complexity**: The setup implies the computation of the u^j in $\mathbb{Z}_q$ (thus $(k+1)$ multiplications in $\mathbb{F}_q$) and two u-ary decomposition of secret in $\mathbb{Z}_q$ (once for x_a and once for x_b) thus $2k$ multiplications in $\mathbb{F}_q$. We obtain a setup time complexity of

$$(3k+1)$$

– **Public key size**: The public key is composed of the elements g, h, the two C from $\mathbb{G}$, u, k and the $\{u^j\}$ from $\mathbb{Z}_q$. Thus the size of the public key is

$$4|\mathbb{G}| + (k+3)|\mathbb{Z}_q|$$

Annexe B

How to Protect Customers' Privacy in Billing Systems

How to Protect Customers' Privacy in Billing Systems

S.Canard and A.Jambert

Abstract. In this paper, we study the privacy of customers in billing systems. In such systems, customers can use a service provided by some provider of several services but need to pay the bill afterwards. We focus on the privacy property in such systems, that is the infeasibility for any actor of the system to know which customer is paying for which service. More precisely, we give a new approach based on the use of a new cryptographic building block we introduce: sanitizable group signature scheme. Such scheme, of independent interest, permits group members to anonymously sign a message. Moreover, one designated entity is able to open one given signature to retrieve the identity of the signer. Finally, another authorized entity can modify some designated parts of the initial message in such a way that the opening of the new signature is due to the initial group member.

1 Introduction

As more and more users buy services on the internet, a lot of work as been done in order to construct systems for such transactions, considering customer privacy. Especially, it is possible to make the user anonymous by authenticating her using a pseudonym to access a service, while being truly authenticated to pay for the service. A lot of research has also been done on secure anonymous payment systems such as e-cash using group (blind) signature schemes.

In this paper, we consider a set of users, a service provider and a billing provider. Our aim is to permit users to buy services from the service provider in such a way that the latter is able neither to know the identity of the user nor to link two purchases from the same user. Moreover, the billing provider can use a strong authenticated payment scheme with users while being unable to find out the services bought. Finally, in case of a claim by a user, the service provider or the billing provider, a designated judge should be able to settle the problem.

In this paper, we study a new approach with a stronger non-repudiation property and a true untraceability of customers. This is achieved thanks to a new cryptographic primitive, which is of independent interest: *Sanitizable Group Signature*. This family of signatures possesses properties from group [1, 4, 7, 19, 5, 17, 8, 15, 9] and sanitizable signature [2, 12, 22, 10]. This means that signers obtain a similar anonymity to the group signatures: group users can anonymously sign any message and only the Opener of the group is able to know who in the group signed a given signature. At the same time, a designated sanitizer is able to modify some parts (fixed by the signer during the signature phase) of a signed

message while keeping a valid group signature of the signer on the modified message. We will show that in our practical case, a weaker but faster form of sanitizable group signature (Non-Transparent) are sufficient.

In this paper, we define the notion of (Non-Transparent) sanitizable group signature scheme and propose a corresponding model. Then we give a practical construction based on the literature on sanitizable and group signatures, respectively secure (but not private) in the Brzuska *et al.* model [10] for sanitizable signatures and the BSZ one [5] for group signatures.

Next, the idea behind our privacy-preserving billing system is that customers construct a NT-sanitizable group signature on a message which includes, at a modifiable place, the chosen service. Then the service provider sanitizes the signature by replacing the service identifier by the total price to pay. As a consequence, the billing provider does not learn the chosen service but, acting as an opener in the group signature scheme, can obtain the public key of the customer and establish a strong authenticated payment protocol with her.

The paper is organized as follows. After the present introduction, we study in Section 2 the security properties required for our billing solution and look at naive solutions. In Section 3 we introduce the notion of Non-Transparent *sanitizable group signature* and propose a model for such schemes. In Section 4, we propose a construction of such schemes and the way to use it in our secured billing system. Then in Appendix A, we present some extensions for sanitizable group signature and conclude.

2 Privacy-Preserving Billing Systems

In this section, we introduce the general principle we want to study. We next study related work and finally study several possibilities to solve our problem.

2.1 General Principle

We consider a user $\mathcal{U}$ and a service provider $\mathcal{SP}$. The objective is that the user can obtain a service from the service provider in such a way that the service provider ignores the identity of the user. Next, the service provider wants to charge the user for this service, using a post-paid system. Thus, the service provider uses a billing provider $\mathcal{BP}$ to send a bill to the user. Moreover, we are in the case where the service provider proposes several different services with possibly different billing codes. In the following, a service is denoted s and the corresponding billing code is denoted α_s. We assume that the billing code does not reveal anything about the requested service, being in the case where a billing code refers several possible services. We next require that the billing provider ignore which service the user has ordered.

Such system is divided into several phases: `GeneralSetup` is a protocol between the service provider and the billing provider to initialize our protocol. `UserSetup` is a protocol used by each user to initialize their part of the protocol. Finally, the main protocol is divided into three parts: the protocol itself,

denoted `ServiceRequest`, allows users to construct a valid request for the service s, `BillingRequest` next permits the service provider to generate a billing request and `IdentityRequest` with the billing code α_s related to s and permits the billing provider to construct a valid payment request to the correct user.

More precisely, we want to design a system with the following properties:

- **Unlinkability.** From the user's point of view, the service provider should not know which user is asking for a given service.
  In the corresponding experiment, the adversary $\mathcal{A}$ can fully corrupt the Service Provider and can create a coalition of users to obtain the signing keys of any user in the system. At any time, $\mathcal{A}$ chooses two different uncorrupted users $\mathcal{U}_0$ and $\mathcal{U}_1$ and a service s. The challenger next outputs either a request for service s from the user $\mathcal{U}_0$ if $b = 0$ (a random bit from the challenger) or from the user $\mathcal{U}_1$ if $b = 1$.
  The adversary wins if she outputs a bit b' such that $b = b'$ with a probability non-negligibly better than $1/2$.
- **Service Secrecy.** From the user's point of view, the billing provider should not know which service is asked by a given user.
  In the corresponding experiment, the adversary $\mathcal{A}$ can fully corrupt the Billing Provider and can create a coalition of users to obtain the signing keys of any user in the system. At any time, the adversary chooses two different services s_0 and s_1 (such that $\alpha_{s_0} = \alpha_{s_1}$) and a not corrupted user $\mathcal{U}$. The challenger next outputs either a request by user $\mathcal{U}$ for the service s_0 if $b = 0$ (a random bit from the challenger) or for the service s_1 if $b = 1$.
  The adversary wins if she outputs a bit b' such that $b = b'$ with a probability non-negligibly better than $1/2$.
- **Non-frameability.** No coalition of users, possibly with the $\mathcal{SP}$, is able to construct a valid request such that either the $\mathcal{BP}$ considers (i.e. can prove) that the request comes from an honest user who did not participate or the $\mathcal{BP}$ considers that the request comes from nobody.
  In the corresponding experiment, the adversary can fully corrupt the service provider $\mathcal{SP}$, she can create a coalition of users to obtain the signing keys of all users except the target one, she can also corrupt users. However, she is not allowed to query the request oracle for the target message with the target user or to obtain the signing key of the target user. Finally, the Adversary outputs a request from a user and a proof τ which proves that user $\mathcal{U}_i$ has produced this request. She wins if the request is valid, $\mathcal{U}_i$ an honest user, and that τ is a publicly verifiable proof that $\mathcal{U}_i$ produced the request.
- **Non Repudiation.** No coalition of users, possibly with the billing provider $\mathcal{BP}$, is able to construct a valid billing request pointing to the service provider $\mathcal{SP}$, if $\mathcal{SP}$ did not participate to its construction.
  In the corresponding experiment, $\mathcal{A}$ can fully corrupt the billing provider $\mathcal{BP}$ and can interact with the service provider $\mathcal{SP}$. $\mathcal{A}$ outputs a request to the billing provider, different from the answers outputted from $\mathcal{SP}$. She wins if the $\mathcal{SP}$ is unable to produce a proof π that she did not produce the request.

2.2 Related Work

Today in many existing systems using, for example, PayPal, the service provider authenticate customers using a pseudonym with no direct link with the true identity of the customer. And the billing provider use a strong authentication system so as to bill the correct customer. Such system is however inefficient when we need untraceability of users. Moreover, the non-repudiation by customers is hard to obtain and may be stronger than what is proposed.

Electronic cash systems [14] allow users to withdraw electronic coins from a bank, and then to pay a merchant using electronic coins preferably in the off-line manner, i.e., there is no need to communicate with the bank or a trusted party during the payment. Finally, the merchant deposits the spent coins to the bank. Electronic cash provides user anonymity against both the bank and the merchant during a purchase to emulate the perceived anonymity of regular cash transaction. It must be impossible to link two spending protocols or a spending protocol to a withdrawal protocol. However, this is not a post-paid system and does not permit to obtain bills. Thus it does not fall in our study.

The Secure Electronic Transaction (SET) system [18] ensure the security of financial transactions on the Internet. A transaction is verified using a combination of electronic certificates (based on a PKI) among the user, the merchant, and the user's bank. The SET system ensures the confidentiality of the user in such a way that the merchant does not learn anything about the user's bank account. The privacy of users is also somewhat protected since the requested service is not known by the bank, since encrypted for the merchant. Moreover, as the user uses digital signatures and a certificate on a pseudonym not related to her true identity, she is anonymous but linkable *w.r.t.* the merchant : The merchant does not know the identity of the user but can link transactions.

2.3 Several Ideas

First of all, the user has to be anonymous and unlinkable *w.r.t.* the service provider. For this purpose, it may be possible to use anonymous authentication systems such as blind signatures [14], ring signatures [20] or group signatures [13]. But, as a user may interact several times with one or several service providers, blind signatures may not be appropriate in this case. Next, a ring signature permits a user to be anonymous but with no control on the group of possible users.

Using a group signature. A group signature scheme allows users to sign documents on behalf of the group in such a way that signatures remain anonymous and unlinkable for everybody but a designated authority, sometimes called the opening manager, who can recover the identity of the signer whenever needed (the latter procedure is called "signature opening"). This is very useful in our context since the user, being a user of the group, can be anonymous **and unlinkable** *w.r.t.* the service provider, acting as a verifier of a group signature. Moreover, the opening manager can be played by the billing provider so as to

retrieve the identity of the user to send her the corresponding bill. To obtain the non-repudiation, this group signature can be given to the billing provider but, in this case, this latter also obtains the provided service. Thus, this approach does not verify the service secrecy property. Moreover, the service provider should send the corresponding billing code to the billing provider.

$$\mathcal{U} \longrightarrow \mathcal{SP} \longrightarrow \mathcal{BP} : \text{GSIGN}(s \| \alpha_s)$$

Encrypting the service to obtain service secrecy. One solution, based on the SET system [18], is for the user to encrypt the requested service for the service provider, in such a way that the billing provider can not decrypt it. The user has next to include the encrypted value in the message to be signed using the group signature scheme.

$$\mathcal{U} \longrightarrow \mathcal{SP} \longrightarrow \mathcal{BP} : \text{GSIGN}(\text{ENC}_{\mathcal{SP}}(s) \| \alpha_s)$$

However, in practice, the service provider's billing codes should not necessarily be known by users. Thus, the service provider needs to add these codes to the message sent to the billing provider.

Signature by $\mathcal{SP}$ on the billing code. One solution to the above problem is for the service provider to add this information and signs both the initial group signature and the billing code before sending all to the billing provider.

$$\begin{aligned} \mathcal{U} &\longrightarrow \mathcal{SP} : \sigma = \text{GSIGN}(\text{ENC}_{\mathcal{SP}}(s)) \\ \mathcal{SP} &\longrightarrow \mathcal{BP} : \text{SIGN}_{\mathcal{SP}}(\sigma \| \alpha_s) \end{aligned}$$

In fact, this is possible to do something more efficient. Our solution does not require the service provider to make a signature but only a modification of the initial group signature, as described below.

Discussion on the use of sanitizable signatures. A sanitizable signature scheme permits a signer to produce a signature on a document, which document can be further modified, in a limited and controlled fashion, by a designated semi-trusted "sanitizer", with no interaction with the original signer. Moreover, the signature on the resulting message should be verifiable as a signature from the original signer. Finally, the sanitizer should be able to modify only the sanitizable parts of the message, that is, the parts that have been stated as modifiable/admissible by the signer.

It may thus be possible to plug a sanitizable signature to a group signature, so as to obtain all the desired properties. According to [10], a sanitizable signature scheme should be immutable (the sanitizer can only modify a message according to what has been stated by the signer), accountable (this is not possible to falsely accuse the signer/sanitizer from having produced a particular signature) and transparent (this is not possible to know if a message has been signed or not). However, in our context, we do not really need the transparency property. And, if services are encrypted, we can allow that the original message is re-computable from sanitized ones. As the existing schemes are designed to obtain transparency, it may be possible to simplify these schemes.

In the following, we consequently formally define the new concept of Non-Transparent sanitizable group signature, proposed constructions in both random oracle and standard model, and next apply it in the context of privacy-preserving billing systems. We also sketch the notion of transparent sanitizable group signature scheme, which may be of independent interest.

3 Non-Transparent Sanitizable Group Signature

A Non-Transparent sanitizable group signature scheme (noted NT-SGS) is composed of users of the group user, a group manager $\mathcal{GM}$, an opening manager $\mathcal{OM}$, a sanitizer $\mathcal{S}$ and a verifier $\mathcal{V}$. It permits users to sign messages as group member in such a way that

- signatures are anonymous and unlinkable except for the opening manager;
- this latter can revoke the anonymity of any group signature, at any time;
- the signed message is divided into several parts, and the sanitizer is allowed to modify some parts of the message in such a way that the resulting signature is still attributed to the initial group signer (note that the sanitizer is not necessarily a user of the group).

Note that it is possible to use all extensions proposed in [11] to our model, in order to obtain extended $(NT) - SGS$ constructions but we do not describe it here as we do not need them in our Billing system.

In the following, ADM is defined by the signer, given a message m of length ℓ and divided into t blocks, as : the length ℓ_i of each block m_i (such that $\ell = \sum_{i=1}^{t} \ell_i$) and the index of the block which will be modifiable by the sanitizer, i.e. the subset $\mathcal{T}$ of $[1, t]$ such that for all $j \in \mathcal{T}$, m_j is modifiable. By misuse of notation, we say that $i \in \mathsf{ADM}$ if $i \in \mathcal{T}$.

Moreover, on input a message m and the variable ADM, the sanitizer is able to define the modifications MOD as the set of all the (i, m'_i) such that $\mathcal{S}$ wants to replace the i-th block of m by m'_i. Note that if $(i, \mathsf{m}'_i) \in \mathsf{MOD}$, then $i \in \mathsf{ADM}$.

In the following, we formally describe procedures and security requirements.

3.1 Formal Definition of Non-Transparent Sanitizable Group Signature

A *Non-Transparent Sanitizable Group Signature scheme* $NT - \mathcal{SGS}$ is composed of the following algorithms, where λ is a security parameter:

- $\textsc{Setup}(1^\lambda)$ outputs the public key of the group gpk, the secret key msk of $\mathcal{GM}$, the secret key osk of $\mathcal{OM}$, and the parameters param of the system. In the following, we consider that λ is included into param.
- $\textsc{SanSetup}(1^\lambda)$ outputs a key pair $(\mathsf{spk}, \mathsf{ssk})$ for a sanitizer $\mathcal{S}$.
- $\textsc{Join}$ is an interactive protocol between the Group Manager $\mathcal{GM}$ and the i-th group user $\mathcal{U}_i$. At the end of the protocol, $\mathcal{U}_i$ obtains a user secret key usk_i and a certificate cer_i to prove her membership of the group while $\mathcal{GM}$ makes an entry $\mathsf{reg}[i]$ in a registration table corresponding to this registration. In the following, we consider that the table reg is included into gpk.
- $\textsc{Sign}(\mathsf{m}, (\mathsf{usk}_i, \mathsf{cer}_i), \mathsf{spk}, \mathsf{gpk})$ permits the group user $\mathcal{U}_i$ to sign a message as a user of the group. It outputs a sanitizable group signature σ on the message m, which signature contains the variable ADM as defined above.
- $\textsc{Sanitize}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{MOD}, \mathsf{ssk})$ allows the sanitizer to sanitize a given signed message. The modifications MOD describes the new message m' as described above. The algorithm outputs a new signature σ' and the modified message m' or $\perp$ in case of error.
- $\textsc{Verify}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk})$ permits to verify the signature σ (sanitized or not) on the message m. It outputs 1 if the signature is correct and 0 if it is not.
- $\textsc{Open}(\mathsf{m}, \sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{spk})$ permits the Opener $\mathcal{O}$ to open a signature i.e. to find who in the group signed the original message. It outputs a proof τ which contains the identity of the group user $\mathcal{U}_i$ and a proof of this claim $\tilde{\tau}$. In case the output is $\perp$, it is claiming that no group user produced σ.
- $\textsc{JudgeOpen}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}, \tau)$ is a public algorithm which aims at deciding the origin of a given message-signature pair (m, σ). It outputs 1 if the identity guessed in τ is exact and 0 otherwise.
- $\textsc{JudgeSanit}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk})$ is a public algorithm which aims at deciding the origin of a given message-signature pair (m, σ). It outputs either signer if she produced the given pair $(\mathsf{m}\sigma)$ or $\mathsf{sanitizer}$ if she did not.

3.2 Interactions for the Adversary

We now define the adversary $\mathcal{A}$ against a sanitizable group signature scheme. As usual, such adversary plays experiments with some kind of challenger to break security properties, using some oracles. An experiment for each security property will be given below.

We here consider that the $\textsc{Setup}$ and the $\textsc{SanSetup}$ both correspond to a unique interactive procedure between the challenger and the adversary. In fact, the adversary playing the role of the group manager $\mathcal{GM}$ (resp. opening manager $\mathcal{OM}$, sanitizer $\mathcal{S}$) secretly generates msk (resp. osk, ssk) while the challenger secretly generates other secret keys. The corresponding public values gpk, ssk, and param are also generated during these steps. We sum up this steps by first executing the procedure $\textsc{ParamKeyGen}$ which on input 1^λ outputs param and some public pk and secret sk values, such that $\mathsf{pk} \subset \{\mathsf{gpk}, \mathsf{spk}\}$ and $\mathsf{sk} \subset \{\mathsf{msk}, \mathsf{osk}, \mathsf{ssk}\}$. The adversary is next executed on input param and pk to compute her own public keys in $\{\mathsf{gpk}, \mathsf{spk}\}$ and finally outputs both gpk and spk.

We here consider that if these values are not coherent with the ones from the first step, the experiment is aborted.

The adversary can also play the role of any actor (group user, group manager, opening manager, sanitizer) in the above procedures if she knows the corresponding secret key. Next, the adversary can ask any oracle to execute, as described above, one of the following algorithm: SIGN, SANITIZE, OPEN, PROOF. Moreover, $\mathcal{A}$ can play the role of either the group manager (using the JOIN oracle) or the group user (using the ISSUE oracle) in the JOIN protocol. The adversary can also corrupt users by using the CORRUPT(i) oracle to corrupt group user $\mathcal{U}_i$. In the following, the execution of $\mathcal{A}$ with access to the oracle XXXX and with input e is denoted by $\mathcal{A}^{\text{Xxxx}}(\mathsf{e})$.

3.3 Computational Experiments

We first describe the following generic computational experiment where the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ aims at outputting a message and a corresponding sanitizable group signature with some conditions specific to the security property, depending on its power (keys and oracles). This experiment works for each computational property, that is for all $P \in \{\text{opener}, \text{signer}, \text{sanitizer}\}$, each of them being detailed further.

Experiment $Exp^{\mathsf{comp}\text{-}P}_{\mathcal{SGS},\mathcal{A}}(1^\lambda)$:

1. $(\mathsf{param}, \mathsf{pk}, \mathsf{sk}) \longleftarrow$ PARAMKEYGEN(1^λ),
2. $(\mathsf{gpk}, \mathsf{spk}, \mathsf{data}) \longleftarrow \mathcal{A}_1(\mathsf{param}, \mathsf{pk})$,
3. $(\mathsf{m}, \sigma) \longleftarrow \mathcal{A}_2^{\mathcal{O}}(\mathsf{param}, \mathsf{gpk}, \mathsf{spk}, \mathsf{data})$,
4. if $\mathsf{cond}(\mathsf{m}, \sigma) = 1$ return 1
5. return 0

The value data corresponds to the information the adversary $\mathcal{A}_1$ wants to give to the adversary $\mathcal{A}_2$. It remains to clearly define, for each security property, what is behind the public values param and pk, what are the keys sk unknown from the adversary, the secret key apk computed (and known) by the adversary, the oracles $\mathcal{O}$ which are accessible to the adversary and the conditions cond of acceptance of the adversary's response.

The success probability of $\mathcal{A}$ in breaking the P's protection, with $P \in \{\text{opener}, \text{signer}, \text{sanitizer}\}$, is defined by

$$Succ^{\mathsf{comp}\text{-}P}_{\mathcal{SGS},\mathcal{A}}(1^\lambda) = \Pr\left[Exp^{\mathsf{comp}\text{-}P}_{\mathcal{SGS},\mathcal{A}}(1^\lambda) = 1\right].$$

We finally have the following definition.

Definition 1 (P Property). *For all $P \in \{\text{opener}, \text{signer}, \text{sanitizer}\}$, a sanitizable signature scheme $\mathcal{SGS}$ verifies the P's protection if for any polynomial-time adversary $\mathcal{A}$, the success probability $Succ^{\mathsf{comp}\text{-}P}_{\mathcal{S},\mathcal{A}}(\cdot)$ is negligible.*

We now detail each protection by giving the details on the above experiment.

Opener's Protection. This property comes from the traceability property for group signature schemes. It asks that the adversary is unable to produce a valid sanitizable group signature which can not be open correctly. That is, either the opening algorithm gives no group user (that is output $\perp$), or the opening manager fails to give a correct proof of opening. Here, the adversary plays the role of the group manager and the sanitizer, and can corrupt group users. The opening manager is assumed to be honest in this experiment. The resulting experiment is very close to the initial one from group signature schemes. The details are defined in Figure 1.

Signer's Protection. This property initially comes from the immutability of sanitizable signatures. More precisely, the aim of the adversary is to output a message and a signature such that he has either modified a non admissible part of the message, or output a message for which he has not received the right to sanitize it, or (see sanitizer's accountability in sanitizable signature schemes) output a message he has sanitized while the judge says the output signature comes from a signer. In the corresponding experiment (see details in Figure 1), the adversary plays the role of the sanitizer, the group manager and the opening manager. $\mathcal{A}$ can moreover corrupt group users.

Remark 1. For the last possibility for the adversary's output, we need to differ from the initial immutability property for sanitizable signature schemes. In fact, the adversary can here corrupt a group user, ask her to make a sanitizable group signature using his sanitizable key, and finally, sanitize this message and output the new couple. In the immutability experiment for sanitizable group signature scheme, we consequently need to check that the initial signature comes from an honest group user.

This introduces a new observation. A secure group signature scheme needs to verify the non-frameability property, which asks that the adversary be unable to create a judge-accepted proof that an honest user produced a certain valid signature unless this user really did produce this signature. In fact, this property needs not to be added in our model, since it is already included in the signer's protection property. This is due to the fact that if the sanitizer can modify a previously obtained group signature from an honest user, he is able to give a new message and a valid signature which is attributed to the honest user. Thus, the adversary should not output a message and a signature such that he has previously obtained the right to sanitize it by the honest user. This case is thus totally included in the signer's protection experiment in Figure 1.

Sanitizer's Protection. This property asks that group users are not able to falsely accuse an honest sanitizer to have produced a given message and signature. For this purpose, the adversary plays the role of the group manager, the opening manager and all group users. This property is very close to the signer's accountability for sanitizable signature schemes and the experiment is given in Figure 1.

Opener's Protection	
$\mathcal{A}$:	$\mathcal{U} + \mathcal{GM} + \mathcal{S}$
$\mathcal{O}$:	CORRUPT, JOIN, SIGN, OPEN
cond:	• VERIFY$(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}) = 1$ and • OPEN$(\mathsf{m}, \sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{spk}) = (i, \tau)$ and • $\big($JUDGEOPEN$(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}, \tau) =$ 0 or $(i, \tau) = \perp \big)$

Sanitizer's Protection	
$\mathcal{A}$:	$\mathcal{U} + \mathcal{GM} + \mathcal{OM}$
$\mathcal{O}$:	SANITIZE
cond:	• VERIFY$(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}) = 1$ and • (m, σ) does not come from SANITIZE and • JUDGESANIT$(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}) =$ sanitizer

Signer's Protection	
$\mathcal{A}$:	$\mathcal{U} + \mathcal{GM} + \mathcal{OM} + \mathcal{S}$
$\mathcal{O}$:	CORRUPT, JOIN, SIGN
cond:	• VERIFY$(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}) = 1$ and • $\big(\exists j \in [1, t] : \mathsf{m}_j \in \mathsf{MOD}$ and $j \notin \mathsf{ADM}\big)$ or • OPEN$(\mathsf{m}, \sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{spk}) = (i, \tau)$ and $i \in \mathcal{HU}$ and • $(\forall j :$ OPEN$(\mathsf{m}_j, \sigma_j, \mathsf{osk}, \mathsf{gpk}, \mathsf{spk}_j) = (i, \tau_j)$, $\mathsf{spk}_j \neq \mathsf{spk})$ or • JUDGESANIT$(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}) =$ signer

Fig. 1. Description of computational experiments

3.4 Decisional Experiments: the Case of the Unlinkability

As there are only one decisional experiment, corresponding to the unlinkability property, we only describe this case in the following, where $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$. Informally speaking, the adversary, playing the role of group manager and the sanitizer (but not the one of the opening manager, for obvious reasons), is not able to distinguish, between two honest group users of her choice, which one has produced a given santizable group signature.

Experiment $Exp^{\mathsf{unlink}-b}_{\mathcal{SGS},\mathcal{A}}(1^\lambda)$:

1. $(\mathsf{param}, \mathsf{pk}, \mathsf{osk}) \longleftarrow$ PARAMKEYGEN(1^λ),
2. $(\mathsf{gpk}, \mathsf{spk}, \mathsf{data}_1) \longleftarrow \mathcal{A}_1(\mathsf{param}, \mathsf{pk})$,
3. $(\mathcal{U}_0, \mathcal{U}_1, \mathsf{m}, \mathsf{data}_2) \longleftarrow \mathcal{A}_2^{\mathcal{O}}(\mathsf{param}, \mathsf{gpk}, \mathsf{spk}, \mathsf{data}_1)$,
4. $\sigma_b =$ SIGN$(\mathsf{m}, (\mathsf{usk}_b, \mathsf{cer}_b), \mathsf{spk})$,
5. $(b') \longleftarrow \mathcal{A}_3^{\mathcal{O}}(\mathsf{param}, \mathsf{gpk}, \mathsf{spk}, \mathsf{data}_2, \sigma_b)$

The advantage of $\mathcal{A}$ for the unlinkability experiment is defined by:

$$Adv^{\mathsf{unlink}}_{\mathcal{SGS},\mathcal{A}}(\lambda) = \Pr\left[Exp^{\mathsf{unlink}-1}_{\mathcal{SGS},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[Exp^{\mathsf{unlink}-0}_{\mathcal{SGS},\mathcal{A}}(1^\lambda) = 1\right].$$

Definition 2 (Unlinkability). *A system $\mathcal{SGS}$ is* unlinkable *if for any polynomial--time adversary $\mathcal{A}$, the adversary advantage $Adv^{\mathsf{unlink}}_{\mathcal{SGS},\mathcal{A}}(\cdot)$ is negligible.*

4 Our Construction for a Non-Transparent Sanitizable Group Signature

In this section, we provide two Non-Transparent sanitizable group signature schemes (NT-SGS scheme for short). The first one is more efficient but secure in the random oracle model. The second one is secure in the standard model but less efficient. As we study a very practical application (namely privacy-preserving billing system), we give more details on our most efficient construction, such that we can study its implementation in more details. Ideas of proof are given in Appendix B.1.

4.1 Construction in the Random Oracle Model

Formally, our construction use the work of [15] on group signature and a classical chameleon hash function proved in the random oracle model.
We rely on the XDH problem, thus we consider three isomorphic cyclic groups of prime order p: $\mathcal{G}_1, \mathcal{G}_2$ and $\mathcal{G}_T$ and a bilinear map $\mathsf{e} : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$ (which can be evaluated efficiently), the DDH problem is easy in $\mathcal{G}_2$ and hard in $\mathcal{G}_1$. Finally, we denote ϕ the isomorphism from $\mathcal{G}_2$ to $\mathcal{G}_1$, which we consider one-way (easy to compute, but hard to invert). Then, in our description, we note $NIZKPK(c, C, H)$ a non-interactive zero knowledge proof of equality of the discrete logarithm C in base H with the committed value c, $PK(a, b)$ a non-interactive zero knowledge proof of the discrete logarithm of a in base b and $ZKPK(x : y(x))$ a non interactive zero knowledge proof of x such that $y(x)$ holds. Finally we note $Ext-Commit(x)$ an extractable commitment (perfectly binding, computationally hiding and with a trapdoor to open it).

- SETUP(1^λ):
  A generator g_2 of $\mathcal{G}_2$ is randomly chosen such that $\phi(\mathsf{g}_2) = \mathsf{g}_1$ is a generator of $\mathcal{G}_1$. And z is randomly chosen in $\mathcal{G}_1$.
  Then, the secret key of the manager $\mathsf{msk} = \gamma$ is randomly chosen in $\mathbb{Z}_p$, we note $w = g_2^\gamma$. The secret key of the opener $\mathsf{osk} = (\xi_1, \xi_2)$ is chosen in $\mathbb{Z}_p \times \mathbb{Z}_p$ such that $h = z^{\xi_1}, g = z^{\xi_2}$. Finally, the group public key of the group is $\mathsf{gpk} = (\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, e, \psi, g_1, g_2, h, z, g, w)$.
- JOIN(user i : gpk, upk_i, usk_i; manager : gpk, msk):

  user → manager picks randomly y in $\mathbb{Z}_p$, $\mathcal{C} \leftarrow h^{y'}$,
  $\mathsf{U} \leftarrow (c = \text{Ext-Commit}(y), NIZKPK(c, C, H))$ in $\mathbb{Z}_p^*$
  and sends C, U

  user ← manager verifies $C \in \mathcal{G}_1$, checks U
  picks randomly $x, y'' \in \mathbb{Z}_p$,
  $A \leftarrow (g_1 \cdot C \cdot h^{y''})^{\frac{1}{\gamma + x}}$
  $B \leftarrow e(g_1 \cdot C \cdot h^{y''}, g_2)/e(A, w)$
  $D \leftarrow e(A, g_2)$
  $V \leftarrow PK(B, D)$
  sends A, y'', V

$\mathsf{user} \rightarrow \mathsf{manager}$	$B \leftarrow e(g_1 \cdot C \cdot h^{y''}, g_2)/e(A, w)$
	$D \leftarrow e(A, g_2)$
	$y \leftarrow y' + y''$
	Verifies V and that $A \in \mathcal{G}_1$
	Sends $S \leftarrow Sign_{usk}(A)$
$\mathsf{user} \leftarrow \mathsf{manager}$	Checks S with upk and A
	$reg[i] = (upk, A, x, S)$
	Sends x
user :	Checks if $A^{\gamma+x} == g_1 \cdot h^y$
	i.e. $e(A, g_2)^x \cdot e(A, w) \cdot e(h, g_2)^{-y} == e(g_1, g_2)$
	$\mathsf{usk}_i = (x, y)$ with $\mathsf{cer}_i = A$

– SanSetup(1^λ):
Let $H : \{0,1\}^* \rightarrow \{0,1\}^k$ be a collision resistant hash function and $n = pq$ with p and q prime numbers of λ-bit. A random integer f is chosen such that it is prime to $\phi(n) = (p-1)(q-1)$. The corresponding secret key is d such that $fd = 1 \pmod{\phi(n)}$. Finally, the public key of the sanitizer is $\mathsf{spk} = (f, n)$ and the secret key is $\mathsf{ssk} = (p, q, d)$.

– Sign($\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_t, \mathsf{usk}_i, \mathsf{cer}_i, \mathsf{spk} = (f, n), \mathsf{gpk}$).
First, the signer generates the variable $\mathsf{ADM} = (\{\ell_i\}_{1 \leq i \leq t}, \{i | \mathsf{m}_i \text{ is modifiable }\})$ with ℓ_i the size of the i-th block. Let u be the number of modifiable parts in m. In order to compute the chameleon hash function, she randomly chooses r in $\{0,1\}^\lambda$. Then she computes $\tilde{\mathsf{m}} \leftarrow H(\mathsf{m}) \cdot \mathsf{r}^f \pmod n$. After that, the signer randomly chooses $\alpha, \beta \in \mathbb{Z}_p$ and computes $T_1 \leftarrow z^\alpha$, $T_2 \leftarrow A \cdot h^\alpha$, $T_3 \leftarrow z^\beta$, $T_4 \leftarrow A \cdot g^\beta$. Then he computes $\delta \leftarrow x\alpha + y$.
Then, the signer randomly chooses $r_\alpha, r_\beta, r_\gamma, r_\delta$ in $\mathbb{Z}_p$ and computes:
- $R_1 \leftarrow z^{r_\alpha}$, $R_2 \leftarrow e(T_2, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha} \cdot e(h, g_2)^{-r_\delta}$, $R_3 \leftarrow z^{r_\beta}$, $R_4 \leftarrow h^{r_\alpha} \cdot g^{-r_\beta}$
- $c \leftarrow \mathcal{H}(\mathsf{m} \| \mathsf{ADM} \| \tilde{\mathsf{m}}, T_1, T_2, T_3, T_4, R_1, R_2, R_3, R_4)$ with $\mathcal{H}$ an hash function.
- $s_\alpha \leftarrow r_\alpha + c\alpha \pmod p$, $s_\beta \leftarrow r_\beta + c\beta \pmod p$, $s_x \leftarrow r_x + cx \pmod p$, $s_\delta \leftarrow r_\delta + c\delta \pmod p$.

She finally obtains the group sanitizable signature $\sigma = (T_1, T_2, T_3, T_4, c, s_\alpha, s_\beta, s_x, s_\delta, \mathsf{m}, \mathsf{r}, \mathsf{ADM})$.

– Sanitize($\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{MOD}, \mathsf{ssk}$).
She recomputes $\tilde{\mathsf{m}} \leftarrow H(\mathsf{m}) \cdot \mathsf{r}^f \pmod n$, then $\forall i \in \mathsf{MOD}$ she replaces m_i by m'_i to obtain m' and computes :

$$\mathsf{r}' \leftarrow \left(H(\mathsf{m}) \cdot r^f \cdot (H(\mathsf{m}'))^{-1}\right)^d \pmod n.$$

The sanitized signature is $\sigma' = (T_1, T_2, T_3, T_4, c, s_\alpha, s_\beta, s_x, s_\delta, \mathsf{m}, \mathsf{r}, \mathsf{r}', \mathsf{ADM})$.

– Verify($\sigma, \mathsf{m}, \mathsf{gpk}, \mathsf{spk}$).
The verifier parses the signature to obtain $\sigma = (T_1, T_2, T_3, T_4, c, s_\alpha, s_\beta, s_x, s_\delta, \mathsf{m}^*, (\mathsf{r}^*,) \mathsf{r}, \mathsf{ADM})$. If she founds i such that $m_i \neq m_i^*$ and $m_i \notin \mathsf{ADM}$ then she outputs 0. Else, if $m = m^*$ she recomputes $\tilde{\mathsf{m}} \leftarrow H(\mathsf{m}) \cdot (\mathsf{r}^*)^f \pmod n$, else she verifies if $H(\mathsf{m}^*) \cdot (\mathsf{r}^*)^f \pmod n = H(\mathsf{m}) \cdot (\mathsf{r})^f \pmod n$ (if not, she outputs 0). We note $\tilde{\mathsf{m}} \leftarrow H(\mathsf{m}) \cdot (\mathsf{r})^f \pmod n$.

Then she computes $c \leftarrow \mathcal{H}(\mathsf{m}^* \| \mathsf{ADM} \| \tilde{\mathsf{m}}, T_1, T_2, T_3, T_4, R_1, R_2, R_3, R_4)$ and returns 1 if

$$z^{s_\alpha} == R_1 \cdot T_1^c,\ z^{s_\beta} == R_3 \cdot T_3^c,\ h^{s_\alpha} \cdot g^{-s_\beta} == R_4 \cdot (T_2 \cdot T_4^{-1})^c,$$
$$e(T_2, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha} \cdot e(h, w)^{-s_\delta} == R_2 \cdot (\frac{e(g_1, g_2)}{e(T_2, w)})^c.$$

else it returns 0.

- $\textsc{Open}(\mathsf{m}, \sigma, \mathsf{osk}, \mathsf{gpk}, \mathsf{spk})$.
  First, she checks if $\textsc{Verify}(\sigma, \mathsf{m}, \mathsf{gpk}, \mathsf{spk}) == 1$, if so she continues, else she outputs $\tau = \perp$. She computes $A \leftarrow (T_2 / T_1^{\xi_1})$ and computes the proof $\tau \leftarrow ZKPK(\xi_1 : A = \frac{T_2}{T_1^{\xi_1}} \wedge h = z^{\xi_1})$. Then she parses reg to find i such that $A \in reg[i]$ and returns $\tau = (i, \tilde{\tau})$.
- $\textsc{JudgeOpen}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}, \tau = \{\perp, (\mathcal{U}_i, \tilde{\tau})\})$. If $\tau = \perp$, then it returns 0. Else, she verifies the proof τ if she holds with $A = \mathsf{reg}[i]$ it outputs 1. If not, it outputs 0.
- $\textsc{JudgeSanit}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk})$.
  The Judge first verifies the pair (m, σ). If the signatures is valid, she parses the signature to obtain $\sigma = (T_1, T_2, T_3, T_4, c, s_\alpha, s_\beta, s_x, s_\delta, \mathsf{m}^*, (\mathsf{r}^*,)\ \mathsf{r}, \mathsf{ADM})$. If $\mathsf{m} = \mathsf{m}^*$, it outputs signer else, it outputs $\mathsf{sanitizer}$.

4.2 Construction in the Standard Model

A similar construction can be proposed in the standard model using a group signature secure in the BSZ model without the random oracle and a chameleon hash function secure in the standard model. To sum up, the idea is to compute a chameleon hash of the message and to sign the concatenation of the original message, the ADM description and the chameleon hash value.

In fact, we could use the Groth group signature proposed in [16] with one of the chameleon hash function scheme proved in the standard model in[3].

5 Our Privacy-Preserving Billing System

In our construction, we need three different building blocks: a classical *signature scheme* $\mathcal{S} = (\textsc{KeyGen}, \textsc{Sign}, \textsc{Verify})$ using a security parameter λ and a pair of keys $(\mathsf{bsk}, \mathsf{bpk})$; a classical $IND - CPA$ public-key encryption scheme $\mathcal{E} = (\textsc{KeyGen}, \textsc{Enc}, \textsc{Dec})$ using a security parameter λ and a pair of keys $(\mathsf{esk}, \mathsf{epk})$ and a non-transparent sanitizable group signature scheme $NT - \mathcal{SGS}$ as the one described in Sections 3 and 4. Finally in our description, we note $\mathcal{M}$ the role of manager which be played, in practice, by the Billing Provider.

Moreover we assume that all communications between the user and the service provider are anonymous (from the user or the service provider). This is easily done using standard techniques, per example using an anonymizer such as TOR [21].

Formally, our protocol uses the following algorithms and protocols:

– `GeneralSetup`
  $\mathcal{M}$ and $\mathcal{BP}$ run $NT-\mathcal{SGS}.\textsc{Setup}(1^\lambda)$. gpk and param are public, osk is added to the keys of $\mathcal{BP}$ and msk is known by $\mathcal{M}$.
  $\mathcal{BP}$ runs $\mathcal{S}.\textsc{KeyGen}(1^\lambda)$ to obtain $(\mathsf{bsk}, \mathsf{bpk})$.
  $\mathcal{SP}$ runs $NT-\mathcal{SGS}.\textsc{SanSetup}(1^\lambda)$ to obtain $(\mathsf{spk}, \mathsf{ssk})$.
  $\mathcal{SP}$ runs $\mathcal{E}.\textsc{KeyGen}(1^\lambda)$ to obtain $(\mathsf{epk}, \mathsf{esk})$.
– `UserSetup`
  $\mathcal{U}$ runs the protocol $NT-\mathcal{SGS}.\textsc{Join}$ with the manager $\mathcal{M}$. $\mathcal{U}$ obtains $(\mathsf{usk}_i, \mathsf{cer}_i)$ and $\mathcal{M}$ obtains a new entry in reg.
– `ServiceRequest`$(\mathsf{usk}_i, \mathsf{pk}_{san}, \mathsf{epk}, \mathsf{cer}_i, s)$
  $\tilde{s} \leftarrow \mathcal{E}.Enc(\mathsf{epk}, s)$
  $\mathsf{m} \leftarrow \mathsf{m}_l \| \tilde{s} \| m_r$ with m_l and m_r eventually empty.
  ADM is set such that, at least, s is admissible.
  $\sigma \leftarrow NT-\mathcal{SGS}.\textsc{Sign}(\mathsf{m}, \mathsf{pk}_{san}, \mathsf{cer}_i, \mathsf{ADM})$
  return $\mathsf{SR} \leftarrow (\mathsf{m}, \sigma)$
– `BillingRequest`$(\mathsf{SR} = (\mathsf{m}, \sigma), \mathsf{gpk}, \mathsf{pk}_{san})$
  if $NT-\mathcal{SGS}.\textsc{Verify}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{pk}_{san}) == 0$ outputs $\perp$.
  $s \leftarrow \mathcal{E}.\textsc{Dec}(\mathsf{esk}, \tilde{s})$.
  $\mathsf{m}' \leftarrow m_l \| \alpha \| m_r$ with α the billing code corresponding to the service s.
  $\mathsf{MOD} \leftarrow (2, \mathsf{m}')$
  $\sigma' \leftarrow NT-\mathcal{SGS}.\textsc{Sanitize}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{sk}_{san}, \mathsf{pk}_{san}, \mathsf{MOD})$
  return $\mathsf{BR} \leftarrow (\mathsf{m}', \sigma')$
– `IdentityRequest`$(\mathsf{BR} = (\mathsf{m}', \sigma'), \mathsf{osk}, \mathsf{gpk}, \mathsf{pk}_{san}, \mathsf{reg})$
  $\tau = (\mathcal{U}_i, \tilde{tau}) \leftarrow \textsc{Open}(\mathsf{m}', \sigma', \mathsf{osk}, \mathsf{gpk}, \mathsf{pk}_{san}, \mathsf{reg})$
  return $\mathcal{U}_i$

According to our definition of security properties for a Non-Transparent group signature scheme $NT-\mathcal{SGS}$, our construction is secure. Ideas of proof are given in Appendix B.2

6 Extension: Removing the Encryption of the Service

It is possible to remove the encryption of the service in the above proposed billing system, by slightly modifying the used non-transparent sanitizable group signature. In fact, as explained in Section 2.3, this encryption is mostly used to prevent the billing provider from obtaining any information about the user's service during the protocol between the user and the service provider. But, in any case, the billing provider can revoke the anonymity of this request from the user to obtain the identity of this user.

Regarding this request, one other solution consists in making the requested service readable for anyone, including the billing provider, but the identity of the requester unreadable, including the billing provider. As the latter can revoke the anonymity of any group signature, we need to modify the sanitizable group signature in such a way that the billing provider should be able to open a group signature used in the protocol between the service provider and himself but should not be able to open a group signature used in the protocol between the

user and the service provider. The service provider, acting as the sanitizer, needs thus to modify the group signature to obtain this property.

In the used group signature, the identity of the group member is hidden by using an encryption scheme for which the opening manager (the billing provider) knows the decryption secret key. We now differentiate the two group signatures in the whole system, that is the one in the protocol between the user and the service provider, and the other one in sent by the service provider to the billing provider. According to what we said above, the first group signature should not be openable by the opening manager, while the second one should.

Remark 2. Note that the first group signature may be openable by another authority such as a judge, or by nobody.

The sanitizer thus needs to transform the first group signature into the second one, and thus transform a ciphertext on the user's identity for *e.g.* the judge to a ciphertext on the user's identity for the billing provider, without obtaining the corresponding plaintext since the user is anonymous *w.r.t.* the sanitizer/service provider. We thud need a proxy re-encryption scheme [6].

More precisely, if we reconsider our construction in Section 4, we uses as for the encryption scheme the double El Gamal. This needs here to be replaced by one standard El Gamal encryption and one proxy re-encryption version of the El Gamal encryption scheme [6]. Due to lack of space, we do not detail this construction anymore.

7 Conclusion

In conclusion, in this paper, we define a new tool: Non-Transparent Sanitizable Group Signatures Schemes and describe a model to verify its security. Then we propose a secure construction of such a scheme in our model. Then, we use this solution to construct a new solution of payment with a fully anonymous purchase and a strong authenticated payment. Finally, we propose an extensions and conclude.

References

1. Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 255–270, London, UK, 2000. Springer-Verlag.
2. Giuseppe Ateniese, Daniel H. Chou, Breno De Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS: Proceedings of the 10th European Symposium on Research in Computer Security*, pages 159–177. Springer-Verlag, 2005.
3. Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In *SCN: Security in Communication Networks, 4th International Conference*, volume 3352, pages 165–179. Springer-Verlag, 2004.

4. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: formal definition, simplified requirements and a construction based on trapdoor permutations. In Eli Biham, editor, *Advances in cryptology - EUROCRYPT 2003, proceedings of the internarional conference on the theory and application of cryptographic techniques*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629, Warsaw, Poland, May 2003. Springer-Verlag.
5. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.
6. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT'98*, pages 127–144, 1998.
7. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology—CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Berlin: Springer-Verlag, 2004. Available at `http://www.cs.stanford.edu/~xb/crypto04a/`.
8. Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 427–444. Springer, 2006.
9. Xavier Boyen and Brent Waters. Full-domain subgroup hiding and constant-size group signatures. In *In proceedings of PKC 2007*, pages 1–15. Springer, 2007.
10. Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *Irvine: Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography*, pages 317–336, Berlin, Heidelberg, 2009. Springer-Verlag.
11. Sébastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In *CT-RSA 2010*, Lecture Notes in Computer Science. Springer, 2010.
12. Sebastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS'08: Proceedings of Applied Cryptography and Network Security 2008*, volume 5037, pages 258–276. Springer Berlin / Heidelberg, 2008.
13. D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.
14. David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.
15. Cécile Delerablée and David Pointcheval. Dynamic fully anonymous short group signatures. In *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 193–210. Springer, 2006.
16. Jens Groth. Fully anonymous group signatures without random oracles. In *ASIACRYPT*, pages 164–180, 2007.
17. Aggelos Kiayias and Moti Yung. Group signatures with efficient concurrent join. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 198–214. Springer, 2005.
18. Mastercard and VIsa. Secure electronic transaction (set), 1996.
19. Lan Nguyen and Rei Safavi-Naini. Efficient and provably secure trapdoor-free group signature schemes from bilinear pairings. In *ASIACRYPT 2004*, pages 236–347. Springer-Verlag, 2004.
20. Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret: Theory and applications of ring signatures. In *Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 164–186. Springer, 2006.

21. Trusted Computing Group. The TOR Project. http://www.torproject.org/index.html.fr.
22. Tsz Hon Yuen, Willy Susilo, Joseph K. Liu, and Yi Mu. Sanitizable signatures revisited. In *CANS '08: Proceedings of the 7th International Conference on Cryptology and Network Security*, pages 80–97, Berlin, Heidelberg, 2008. Springer-Verlag.

A Transparent SGS

A *Sanitizable Group Signature scheme* $\mathcal{SGS}$ can be constructed using a secure (in the [5] model) sanitizable signature scheme $\mathcal{SS}$ and a secure (in the BSZ model) group signature scheme $\mathcal{GS}$ as follows, where λ is a security parameter: The idea is to replace the classical signature scheme in the sanitizable signature scheme by our chosen group signature.
To describe the construction, we use a pseudorandom function PRF and a pseudorandom generator PRG.

- $\textsc{Setup}(1^\lambda)$:
  We initialize the group signature in order to obtain the group public key gpk, the public parameters param, the secret key of the opener osk and the secret key of the manager msk.
- $\textsc{Join}(\mathsf{user}_i; \mathsf{manager})$
  The algorithms correspond to the interactive Join algorithm of the group signature, in order to decide the secret key of the user usk_{sig}, its certificate of membership to the group cer_i and to register the user in the manager registry reg. Finally, the user chooses at random κ in $\{0,1\}^\lambda$ to obtain hers complete secret key $\mathsf{usk} = (\mathsf{usk}_{sig}, \kappa)$.
- $\textsc{SanJoin}(1^\lambda)$:
  This algorithm corresponds to the setup of the chameleon hash function, and to the choice of the pair of key of the sanitizer $(\mathsf{spk}, \mathsf{ssk})$.
- $\textsc{Sign}(\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_t, \mathsf{usk}, \mathsf{spk}, \mathsf{gpk}, \mathsf{ADM} = (\{\ell_i\}_{1 \le i \le t}, \{i \| \mathsf{m}_i \text{ is modifiable }\}))$.
  First, the signer generates the variable ADM as defined in the model. Let u be the number of modifiable parts in m. During this step, the signer generates the tag $\mathcal{TAG}$ by computing $x = \mathrm{PRF}(\kappa, \mathsf{Nonce})$ where $\mathsf{Nonce} \in \{0,1\}^\lambda$, and $\mathcal{TAG} = \mathrm{PRG}(x)$. In order to compute the chameleon hash function, she randomly chooses $r_1, \cdots, \mathsf{r}_u, \mathsf{r}_c$ in $\{0,1\}^\lambda$. Then, for each admissible block, she executes what is called the (public) "reconstruction procedure", which takes as input the message m, $\mathcal{TAG}$, the r_i's and the public key spk and corresponds to :
  1. Compute the values $\tilde{\mathsf{m}}_i$ for each block:
  $$\forall i, \tilde{\mathsf{m}}_i \leftarrow \begin{cases} \mathsf{h}_i \leftarrow CH.\textsc{Proceed}(\mathsf{spk}, \mathsf{m}_i, \mathsf{r}_i) & \text{if } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i || i & \text{else} \end{cases}$$
  2. Compute the final block : $\mathsf{h}_c \leftarrow CH.\textsc{Proceed}(\mathsf{spk}, \mathcal{TAG} \| \mathsf{m}, \mathsf{r}_c)$.
  3. $\tilde{\mathsf{m}} \leftarrow \tilde{\mathsf{m}}_1 || \cdots || \tilde{\mathsf{m}}_t || \mathsf{h}_c || \mathsf{spk}$

After that, the signer group signs the message $\tilde{\mathsf{m}}$ to obtain a signature s. The group sanitizable signature is $\sigma = (s, \mathcal{TAG}, \mathsf{Nonce}, \mathcal{R}, \mathsf{ADM})$ with $\mathcal{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_u, \mathsf{r}_c\}$. The signature and the message are added to the signer's database $\mathcal{DB}$.

- $\textsc{Sanitize}(\sigma, \mathsf{m}, \mathsf{m}', \mathsf{ssk}, \mathsf{MOD})$.
  The sanitizer executes the reconstruction procedure to recompute the h_i's and h_c. Then, she uses ssk to find a collision on the chameleon hash for each admissible blocks she modifies using the algorithm $\textsc{Forge}$ of the chameleon hash scheme. She also chooses at random Nonce' and $\mathcal{TAG}'$ and computes a collision on h_c such that $\mathsf{h}_c = \textsc{Proceed}_{\mathsf{spk}, \mathcal{TAG}'}(\mathsf{m}', \mathsf{r}'_c)$. Finally, the sanitized signature is $\sigma' = (s, \mathcal{TAG}', \mathsf{Nonce}', \mathcal{R}', \mathsf{ADM})$ where the set $\mathcal{R}' = \{\mathsf{r}'_1, \cdots, \mathsf{r}'_u, \mathsf{r}'_c\}$ (with $\mathsf{r}'_j = \mathsf{r}_j$ if $j \notin \mathsf{MOD}$).
- $\textsc{Verify}(\sigma, \mathsf{m}, \mathsf{gpk}, \mathsf{spk})$.
  The verifier executes the reconstruction procedure as described above to obtain $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 || \cdots || \tilde{\mathsf{m}}_t || \mathsf{h}_c || \mathsf{spk}$. Then she verifies the signature s on the message $\tilde{\mathsf{m}}$ if the verification holds, it outputs 1 else 0.
- $\textsc{Open}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk})$.
  First, she verifies the validity of the signature, if so she uses the $\textsc{Open}$ algorithm of the group signature scheme to find out who in the group signed the message (i) and to prove it $(\tilde{\tau})$. Then she returns (i, τ).
- $\textsc{Proof}(\sigma, \mathsf{m}, \mathsf{usk}_i, \mathsf{reg}, \mathsf{spk}, (\mathsf{m}_i, \sigma_i)_{i=1,2,\cdots,q})$.
  The signer parses σ to obtain $\mathcal{TAG}$ and r_c. Then she searches in $\mathcal{DB}$ an integer $j \in [1, q]$ such that

$$CH.\textsc{Proceed}(\mathsf{spk}, \mathcal{TAG} \| \mathsf{m}, r_c) = CH.\textsc{Proceed}(\mathsf{spk}, \mathcal{TAG}_j \| \mathsf{m}_j, r_{c_j}). \quad (1)$$

  with $\mathcal{TAG}_j = \mathrm{PRG}(x_i)$ for $x_j = \mathrm{PRF}(\kappa, \mathsf{Nonce}_j)$ and $\mathsf{m} \neq \mathsf{m}_i$. If it exists, it outputs $\pi = (\mathcal{TAG}_j, \mathsf{m}_j, \mathsf{r}_{c_j}, x_j)$ else, it outputs $\perp$.
- $\textsc{Judge}(\mathsf{m}, \sigma, \mathsf{gpk}, \mathsf{spk}, \tau = \{\perp, (i, \tilde{\tau})\}, \pi)$. If $\pi = \perp$, then it returns signer. Else she verifies if the Equation (1) holds in π $\mathsf{m} \neq \mathsf{m}_i$ and $\mathcal{TAG}_i = PRG(x_i)$. If so it outputs $\mathsf{sanitizer}$. If not, it outputs signer.

A construction in the random oracle model can be proposed using the Delerablée-Pointcheval group signature with any chameleon hash function. While a construction in the standard model can be obtained thanks to the Groth group construction [16] and a chameleon hash function proved in the standard model like some of the functions proposed by Atenieze and De Meideros in [3].

B Idea of Proofs

B.1 Our Construction for a Non-Transparent Sanitizable Group Signature

According to our model for a Non-Transparent group signature scheme $NT - \mathcal{SGS}$, our construction requires the following properties to be secure.

- **Opener's Protection**

Proof (idea). If the Adversary breaks this property, she can obtain a valid pair (message,signature) such that either the opening algorithm outputs $\perp$ either the opening manager fails to give a correct proof of opening. Thus she can uses this pair to break the traceability property of the group signature (thus the [15] or the [16] in our practical examples). □

– **Signer's Protection**

Proof (idea). A successful adversary (with the role of the sanitizer) has to output a message and a signature such that she has either modified a non admissible part of the message, or output a message for which he has not received the right to sanitize it, or output a message he has sanitized while the judge says the output signature comes from a signer. The two first cases imply to modify either ADM or the original message in the group signature and thus break the unforgeability property of the group signature. In the last case, either $\mathcal{A}$ obtains a group signature of corrupted users such that a judge would consider it from the target user and it breaks the non-frameability of the group signature or the adversary modify the initial message in the signature and thus obtain a forge on the group signature. □

– **Sanitizer's Protection**

Proof (idea). A successful adversary in that case obtain a pair (message, signature) such that it does not come from the targeted honest sanitizer but the public algorithm $JudgeSanit$ falsely accuse him to have produced the pair. Thus the Adversary was able to obtain a forge on the chameleon hash function to obtain $m \neq m^*$ such that their chameleon hash are equal. Thus we can use this to break the collision resistant of the chameleon hash function. □

– **Unlinkability**

Proof (idea). In that case, the adversary, playing the role of group manager and the sanitizer (but not the one of the opening manager, for obvious reasons), is able to distinguish, between two honest group users of her choice, which one has produced a given santizable group signature. Thus we can use the adversary to win the anonymity of the corresponding signature group (the [15] one or the [16] one in our examples). □

B.2 Our Privacy-Preserving Billing System

– **Unlinkability**

Proof (idea). In order to break this property the Adversary can corrupt the Service Provider.
She outputs, with a probability non-negligibly better than 1/2, the identity of the signer of a signature in a set of two signers she chose. Then she can use this knowledge to find the good signer each time in the unlinkability experiment of the sanitizable group signature and so break the property. □

– **Service Secrecy**

Proof (idea). In order to break this property the Adversary can corrupt the Billing Provider.
She outputs the good set of services, with a probability non-negligibly better than 1/2. Then she can break the indistinguishability of the $IND - CPA$ encryption scheme. □

– **Non-frameability**

Proof (idea). As a request is a sanitizable group signature, if an Adversary is able to construct a valid request such that the $\mathcal{BP}$, who plays the opener, can prove that the request comes from a user who did not participate to its construction, then the Adversary can use the same request as a pair (signature, message) to construct a judge-accepted proof τ and then break the Signer's Protection of the Sanitizable Group Signature scheme.
On the other hand, if an Adversary is able to construct a valid request such that the $\mathcal{BP}$ considers that the request comes from nobody then the Adversary can use the same request as a pair (signature, message) to break the opener validity of the Sanitizable Group Signature scheme. □

– **Non-repudiation**

Proof (idea). As a request is a sanitizable group signature, if an Adversary is able to construct a valid billing request such that $\mathcal{SP}$ is unable to produce a proof π that she does not produce the request while $\mathcal{SP}$ did not participate to its construction, then the same request can be used to break the signer accountability part of the responsibility experiment of the Sanitizable Group Signature scheme. □

Résumé

Les problématiques de respect de la vie privée sont aujourd'hui indissociables des technologies modernes. Dans ce contexte, cette thèse s'intéresse plus particulièrement aux outils cryptographiques et à la façon de les utiliser pour répondre à ces nouvelles questions.

Dans ce mémoire, je m'intéresserai tout d'abord aux preuves de connaissance sans divulgation qui permettent notamment d'obtenir la propriété d'anonymat pour les usagers de services de télécommunications. Je proposerai ainsi une nouvelle solution de preuve de connaissance d'un secret appartenant à un intervalle, ainsi que la première étude comparative des preuves existantes sur ce sujet. Je décrirai ensuite une nouvelle méthode permettant de vérifier efficacement un ensemble de preuves de type "Groth-Sahaï", accélérant ainsi considérablement le travail du vérifieur pour de telles preuves.

Dans un second temps, je m'intéresserai aux signatures caméléons. Celles-ci permettent de modifier, sous certaines conditions, un message signé. Ainsi, pour ces schémas, il est possible d'exhiber, à l'aide d'une trappe, une signature valide du signataire initial sur le message modifié. Je proposerai d'abord un nouveau schéma qui est à ce jour le plus efficace dans le modèle simple. Je m'intéresserai ensuite à certaines extensions de ce modèle qui ont pour vocation de donner au signataire les moyens de garder un certain contrôle sur les modifications faites a posteriori sur le message initial. Je décrirai ainsi à la fois le nouveau modèle de sécurité et les schémas associés prenant en compte ces nouvelles extensions.

Enfin, je présenterai un ensemble d'applications se basant sur les briques cryptographiques introduites ci-dessus et qui permettent d'améliorer la protection de la vie privée des utilisateurs. J'aborderai tout particulièrement les problématiques d'abonnement, d'utilisation ou de facturation de services, ainsi que la gestion de contenus protégés dans un groupe hiérarchisé.

Mots-clés: Cryptographie à clé publique, signatures caméléons, preuves de connaissance, protection de la vie privée.

Abstract

Privacy is, nowadays, inseparable from modern technology. This is the context in which the present thesis proposes new cryptographic tools to meet current challenges.

Firstly, I will consider zero-knowledge proofs of knowledge, which allow in particular to reach the anonymity property. More precisely, I will propose a new range proof system and next give the first comparison between all existing solutions to this problem. Then, I will describe a new method to verify a set of "Groth-Sahaï" proofs, which significantly decreases the verification time for such proofs.

In a second part, I will consider sanitizable signatures which allow, under some conditions, to manipulate (we say "sanitize") a signed message while keeping a valid signature of the initial signer. I will first propose a new scheme in the classical case. Next, I will introduce several extensions which enable the signer to obtain better control of the modifications done by the "sanitizer". In particular, I will propose a new security model taking into account these extensions and give different schemes achieving those new properties.

Finally, I will present different applications of the above cryptographic tools that enhance customer privacy. In particular, I will consider the questions of subscription, use and billing of services and also address the issue of managing protected content in a hierarchical group.

Keywords: Public key cryptography, sanitizable signatures, proofs of knowledge, privacy.