Les codes correcteurs au service de la cryptographie symétrique et asymétrique

Abderrahman Daif

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

assystem

École doctorale
Cognition, **L**angage, **I**nteraction

Spécialité
Informatique

THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VIII

Présentée par

Abderrahman DAIF

Pour obtenir le grade de

DOCTEUR

Sujet de la thèse :

Les codes correcteurs au service de la cryptographie symétrique et asymétrique.

Soutenue le 3 mai 2019 devant le jury composé de :

M. Claude Carlet	Directeur de thèse
M. Cédric Tavernier	Co-directeur de thèse
M. Sylvain Duquesne	Rapporteur
M. Damien Vergnaud	Rapporteur
M. Sylvain Guilley	Examinateur
Mme. Sihem Mesnager	Examinateur
M. Emmanuel Prouff	Examinateur

Remerciements

Tout d'abord je tiens à remercier mes deux directeurs de thèse, Claude Carlet et Cédric Tavernier qui ont gentiment accepté d'encadrer cette thèse, et qui ont énormément contribué à sa réussite. Merci à eux pour leur disponibilité permanente, leurs encouragements, et leur soutien.

Je tiens également à remercier le professeur Sylvain Guilley avec qui j'ai eu la chance d'échanger sur divers projets, et qui m'a apporté son conseil à plusieurs reprises.

Un grand merci aussi aux rapporteurs et aux membres du jury, qui ont pris le soin de relire cette thèse, et de me faire part de leurs précieux conseils.

Je remercie mes collègues de l'équipe Cyber-sécurité chez Assystem, ainsi que les membres de la BU Connect qui ont tous contribué, chacun à sa manière, à la réussite de cette thèse. Je remercie en particulier Gregory Landais qui n'a jamais hésité à apporter son soutien et à partager ses idées avec moi.

Je remercie mes amis et camarades de classe, les membres de P8, COC et OP.

Enfin, je remercie ma famille pour leur soutien, en particulier mes parents, mon épouse, et mon fils adoré.

Merci à toutes ces personnes qui ont fait de cette thèse une réussite !

Résumé

Dans la première partie de cette thèse nous étudions les différentes méthodes de masquage permettant de lutter contre les attaques physiques sur les cryptosystèmes embarqués. Rappelons les données du problème. Juste après la naissance du nouveau standard de chiffrement symétrique appelé AES au début de ce siècle, des travaux ont montré que les attaques par canaux auxiliaires (SCA) constituent une menace pour les systèmes de chiffrement implémentés dans des dispositifs embarqués. La consommation électrique pendant une opération de chiffrement ou le temps de calcul peuvent divulguer des informations sur des valeurs intermédiaires manipulées au cours de l'exécution de l'algorithme de chiffrement. Ceci entraîne des attaques pratiques sur les composants électriques. Les rayonnements électromagnétiques, et même les sons émis par les machines à rotor peuvent également donner à l'attaquant une fenêtre de visibilité sur les valeurs intermédiaires au cours du calcul. Ce type d'attaques fournit à l'adversaire une version bruitée de certaines valeurs comme le poids de Hamming de certaines données sensibles. Par ailleurs, les attaques par injection de faute (FIA) consistent à agir sur les conditions environnementales du système (tension, température, rayonnement, lumière, ...) pour générer des erreurs pendant l'exécution de l'algorithme de chiffrement ou déchiffrement et observer ensuite le comportement résultant. De telles attaques peuvent être conçues en ciblant simplement un transistor avec un faisceau laser, ce qui génère un dysfonctionnement lié à la modification de la fréquence du travail pendant le calcul, entraînant une mauvaise valeur de certains bits.

Nous présentons quelques méthodes de masquage pour protéger l'AES contre les SCA et FIA. Chacune de ces méthodes repose sur une structure particulière (le partage du secret, le produit scalaire, les codes correcteurs...). Nous étudions également le détail de la structure de l'AES pour mieux évaluer comment s'implémentent ces méthodes et mesurer la complexité de chacune. Pour ce faire, nous reconstruisons chacune des transformations qui composent l'AES en utilisant ces différentes méthodes de masquage. Cette étude montre la difficulté de masquer à la fois les deux opérations qui interviennent dans les fonctions de l'AES (addition et multiplication) et en même temps de détecter les erreurs potentielles. Pour surmonter ce problème, nous avons conçu un masquage basé sur les codes LCD. Cette

méthode a été introduite initialement en 2014 par Claude Carlet et Sylvain Guilley, et permet de protéger contre les deux types d'attaques. Cependant, le masquage de la multiplication restait un problème ouvert. Pour le résoudre nous avons choisi de restructurer cette méthode en utilisant une représentation de corps finis au lieu d'espaces vectoriels. La nouvelle méthode permet ainsi de masquer l'addition et la multiplication, de détecter les erreurs potentielles, et apporte également un gain intéressant en terme de complexité algorithmique (nous passons d'une complexité quadratique à une complexité linéaire).
Dans la deuxième partie, nous étudions les problématiques liées à la gestions des clés pour les systèmes de chiffrement asymétriques. En 1976, Diffie et Hellman ont marqué le début d'une nouvelle ère dans le domaine de la cryptographie avec l'invention de la cryptographie asymétrique (à clé publique). La cryptographie asymétrique est préférable à la cryptographie symétrique, puisqu' elle permet de communiquer en toute sécurité sans passer par l'échange préalable d'une clé secrète. Deux ans après l'invention de la cryptographie asymétrique, le premier algorithme de chiffrement a été proposé par Rivest, Shamir et Adleman, ce schéma est connu jusqu'à nos jours sous le nom de RSA. Bien que ces systèmes soient efficaces en particulier pour échanger les clés, ils ne permettent malheureusement pas d'identifier le propriétaire d'une clé publique. Il a fallu donc trouver une solution qui permette de créer ce lien entre la clé publique et l'identité de son possesseur dans le but de se protéger contre l'usurpation d'identité.
Nous étudions donc les différentes infrastructures de gestion de clés. En particulier les schémas basés sur les certificats (PKI), et le chiffrement basé sur l'identité (IBE). Nous étudions les travaux réalisés ces dernières années sur IBE. Nous exposons les exigences que doit satisfaire un système de gestion de clés, ces exigences concernent particulièrement le problème de révocation, l'autorité de séquestre, et la décentralisation. Enfin nous présentons une version flexible d'IBE pour satisfaire ces exigences, et qui répond à besoin réel dans le monde industriel.

Abstract

In the first part of this thesis, we are studying different masking methods to protect embedded cryptosystems against physical attacks. Just after the birth of the advanced encryption standard called AES, various studies have shown that side channel attacks (SCA) represent a threat to cryptosystems implemented in embedded devices. The measurement of power consumption during an encryption operation, or the calculation of the time taken by calculi may disclose information about the manipulated values during the execution of the encryption algorithm. This leads to practical attacks on the electrical components. The electromagnetic radiation and even the sounds emitted by the rotor machines can also give the attacker a window of visibility on the intermediate values during the calculation. This type of attack provides a noisy version of values (such as the Hamming weight of some sensitive data) to the opponent. In addition, fault injection attacks (FIA) consist in acting on the environmental conditions of the system (voltage, temperature, radiation...) to generate faults during the execution of the encryption or decryption algorithm and then observe the resulting behavior. Such attacks can be designed by simply illuminating a transistor with a laser beam, which generates a malfunction related to the change of the working frequency during the calculation and which causes a bad value of certain bits.

We are presenting some masking methods to protect the AES against SCA and FIA. Each of these methods is based on a particular structure (secret sharing, inner product, correcting codes ...). We are also studying the details of the AES to better evaluate in a finer way how these methods apply and measure the complexity of each. To do this, we reconstruct each of the transformations that composes the AES using these different masking methods.

This study shows the difficulty of both masking in the two operations that compose the AES (addition and multiplication) and in the same time detecting potential errors. To overcome this problem, we designed a masking method based on LCD codes. This method initially introduced in 2014 by Claude Carlet and Sylvain Guilley protects against both types of attacks. However, masking the multiplication remained an open problem. To solve this problem we chose to restructure this method using a representation of finite fields instead of vector spaces. The new

method thus makes it possible to mask the addition and the multiplication, to detect the potential errors, and brings an interesting gain in terms of algorithmic complexity (we go from a quadratic complexity to a linear complexity).
In the second part, we study issues related to key management for asymmetric encryption systems. In 1976, Diffie and Hellman marked the beginning of a new era in the field of cryptography with the invention of public key cryptography. Public key cryptography is preferable to symmetric cryptography since it allows secured communication without the need for prior exchange of a secret key. Two years after the invention of public key cryptography, the first encryption scheme was proposed by Rivest, Shamir and Adleman, this scheme is known today as the RSA. Although these systems are effective for exchanging keys, they unfortunately do not identify the owner of a public key. It was therefore necessary to find a solution that would create this link between the public key and the identity of its owner in order to protect against identity usurpation.
In this thesis, we are therefore studying different key management infrastructures. Particularly, certificate-based schemes (PKI) and identity-based encryption (IBE). We are studying the work done in recent years on IBE. We outline the requirements that must be met by a key management system. These requirements relate particularly to the problem of revocation, key-escrow, and decentralization. Finally, we present a flexible version of IBE to meet these requirements.

Table des matières

1 | Introduction générale

Sommaire

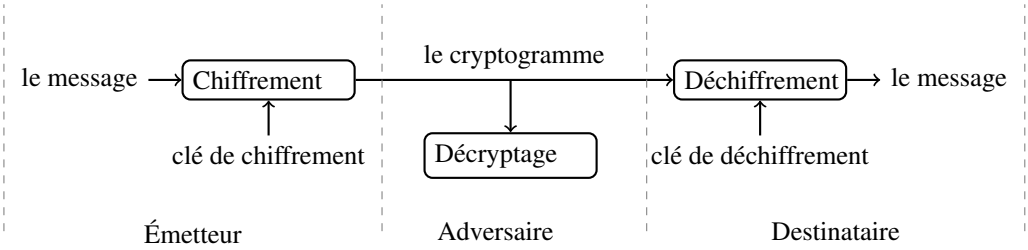
1.1 L'organisation du document

Dans ce document nous allons traiter deux sujets distincts, mais faisant partie tous les deux du domaine de la protection de l'information.
Le premier que nous allons traiter concerne les contre-mesures pour lutter contre les attaques par canaux auxiliaires et les attaques par injection de faute qui visent les composants électroniques. Tout d'abord nous exposons l'état de l'art des différentes techniques existantes, remontrons les avantages et les limites de chacune, puis nous présentons une contre-mesure alternative basée sur les codes correcteurs.
Le deuxième sujet est relatif aux problématiques liées à la gestion des clés publiques (génération, distribution, renouvellement et révocation). Dans la partie II du document nous verrons un état de l'art des différentes solutions qui sont proposées par la communauté des cryptologues, notamment la cryptographie basée sur l'identité. Nous étudierons les points forts et les points faibles de chacune des solutions par rapport aux besoins réels dans les grandes infrastructures, et enfin nous présenterons une architecture qui répond de manière optimale à ces exigences.

1.2 Introduction à la cryptographie

Le développement rapide des communications par internet impacte massivement nos vies quotidiennes, et il engendre un besoin impérieux de sécurisation des informations et des technologies associées. D'où le rôle de plus en plus capital de la cryptographie. La cryptographie constitue un moyen de protection de la vie privée grâce à un ensemble de techniques bien établies et de normes pour protéger les communications contre les écoutes, la falsification, et les attaques de différents types.
La cryptographie a très longtemps été basée sur le traitement du langage, les manipulations consistaient souvent à faire des substitutions (remplacement des lettres) ou des transpositions (mélange des lettres). Depuis l'antiquité, l'une des utilisations les plus célèbres est le chiffre de César, nommé en référence à Jules César qui l'utilisait pour ses communications secrètes. Le chiffre de César consiste simplement à décaler les lettres de l'alphabet. Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair (le message original) par une lettre à distance fixe dans l'ordre de l'alphabet. Cette distance représente la clé de chiffrement de ce système. Pour déchiffrer, le destinataire doit posséder la clé de déchiffrement qui est l'opposé de la clé de chiffrement. Dans le cas de l'alphabet latin, le chiffre de César n'a que 26 clés possibles, il est donc assez facile de casser ce crypto-système.
La cryptographie moderne est fortement basée sur la théorie de l'information et les problèmes difficiles en mathématiques, tels que la factorisation des entiers, le logarithme discret, l'apprentissage en présence d'erreurs (*The Learning With*

Errors (LWE)), l'apprentissage en présence de bruit (*Learning Parity with Noise* (LPN)) ...
La sécurité des systèmes cryptographiques est donc basée sur la difficulté supposée à résoudre ces problèmes, ce qui rend ces algorithmes difficiles à briser.



1.3 Vocabulaire

Le chiffrement est un procédé cryptographique qui permet de rendre la compréhension d'un message impossible à toute personne qui ne détient pas la clé de déchiffrement.

Le déchiffrement est l'opération inverse du chiffrement. Il permet d'extraire le message clair à partir du message chiffré (le cryptogramme) à l'aide de la clé de déchiffrement.

Le décryptage est l'opération qui permet d'extraire le message clair du cryptogramme sans utiliser la clé de déchiffrement. Le décryptage n'est possible que si l'adversaire est capable de trouver une faille dans le système de chiffrement utilisé.

La cryptanalyse est la science qui consiste à étudier et à trouver les failles potentielles dans les systèmes cryptographiques qui existent.

La cryptologie est la science qui combine la cryptographie et la cryptanalyse.

1.3.1 La cryptographie symétrique

On dit qu'un système cryptographique est symétrique lorsque les clés de chiffrement et de déchiffrement se déduisent facilement l'une de l'autre. Les deux clés sont donc sensibles et doivent être maintenues secrètes pour garantir la sécurité. Ce mode de chiffrement à toujours été utilisé depuis l'antiquité pour des fins militaires, politiques ou personnelles. Il reste la méthode la plus simple et la plus rapide en terme de calculs. Aujourd'hui encore, avec l'évolution rapide qu'a connu la cryptographie symétrique ce dernier siècle depuis l'invention d'Enigma, la machine de chiffrement utilisée par les allemands durant la seconde guerre mondiale, jusqu'à

l'*Advanced Encryption Standard* (AES) [DR00], la cryptographie symétrique reste le moyen de chiffrement le plus sûr. Cependant, avec le développement technologique, il est impossible de faire un échange de clé pour chaque transaction (Achat sur internet, connexion sécurisée à distance ...), de plus, pour chaque utilisateur il faut sécuriser un nombre de clés équivalent au nombre de correspondants. Ces inconvénients sont devenus un handicap pour le développement des systèmes de communication dans les années quatre-vingt. Ceci a poussé la communauté des cryptologues à introduire un autre type de cryptographie appelée "cryptographie asymétrique".

1.3.2 La cryptographie asymétrique

En 1976, Diffie et Hellman ont marqué le début d'une nouvelle ère dans le domaine de la cryptographie avec l'invention de la cryptographie à clé publique (appelée aussi : la cryptographie asymétrique) [DH76]. Contrairement à la cryptographie symétrique, la clé de chiffrement et la clé de déchiffrement ne sont pas identiques et même, la seconde ne se déduit pas facilement de la première, de sorte que chaque utilisateur peut posséder une clé publique pour le chiffrement, et une clé secrète pour le déchiffrement. Pour la communication, l'expéditeur chiffre son message en utilisant la clé de chiffrement du destinataire (qui est publique), et envoie ensuite le cryptogramme. Le destinataire, étant donné qu'il est le seul à détenir la clé de déchiffrement, est donc le seul à pouvoir déchiffrer le cryptogramme, et en extraire le message.
La cryptographie à clé publique est préférable à la cryptographie symétrique, puisqu'elle permet de communiquer en toute sécurité sans passer par un échange préalable des clés. Chaque utilisateur peut garder secrète sa clé de déchiffrement et publier la clé de chiffrement associée. Ainsi, si n utilisateurs veulent communiquer de manière confidentielle via un crypto-système à clé publique, ils auront besoin de n couples de clés (publique/secrète), tandis que dans la cryptographie symétrique, il est nécessaire de générer $\binom{n}{2} = \frac{n(n-1)}{2}$ clés. Mais, en termes de rapidité et de complexité des calculs, les crypto-systèmes à clé publique connus aujourd'hui sont beaucoup moins efficaces que les crypto-systèmes symétriques, et ils ont également besoin de clés beaucoup plus longues pour assurer le même niveau de sécurité. C'est pourquoi la cryptographie symétrique est encore largement utilisée et étudiée de nos jours. Les crypto-systèmes à clé publique sont principalement utilisés pour assurer le partage des clés secrètes (symétriques) de manière confidentielle, et pour réaliser des signatures numériques. La figure 1.1 illustre la différence entre les deux modèles de cryptographie.
Deux ans après l'invention de la cryptographie à clé publique, le premier schéma de chiffrement a été proposé par Rivest, Shamir et Adleman [RSA78], ce schéma est connu jusqu'à nos jours sous le nom de RSA. Cette invention a orienté le

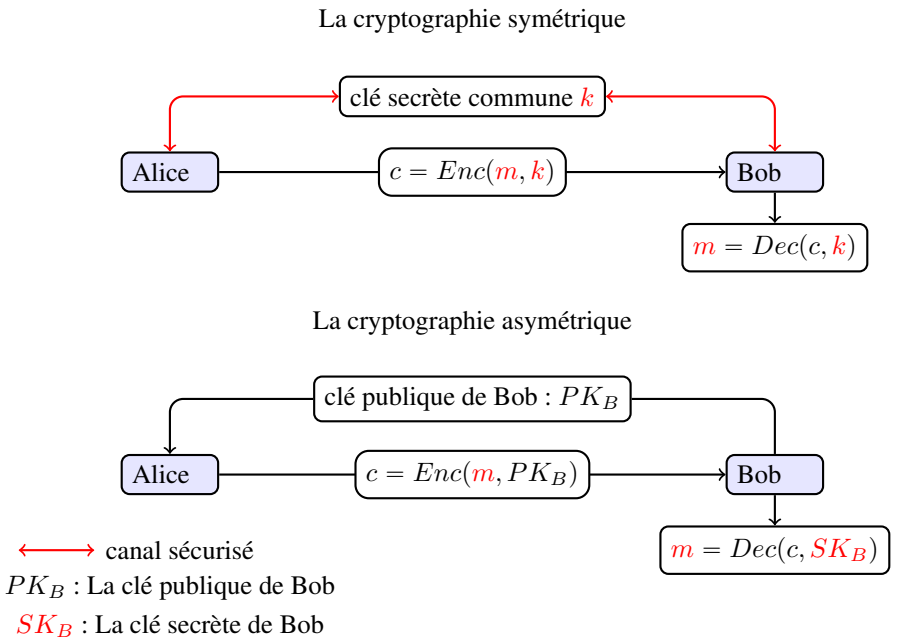


FIGURE 1.1 – Comparaison entre la cryptographie symétrique et asymétrique

développement de la cryptographie en parallèle avec le développement informatique pour fournir progressivement un ensemble d'outils permettant d'assurer les principaux objectifs de la sécurité informatique :

La confidentialité : elle consiste à rendre l'information inintelligible à d'autres personnes que les seuls acteurs de la transaction.

L'intégrité : elle consiste à déterminer si les données n'ont pas été altérées durant la communication (de manière fortuite ou intentionnelle).

La non-répudiation : elle consiste à garantir qu'aucun des correspondants ne pourra nier la transaction.

L'authentification : elle consiste à garantir à chacun des correspondants que son partenaire est bien celui qu'il croit être.

KERCKHOFFS exprime dans son traité intitulé "La cryptographie militaire" que la sécurité d'un crypto-système ne doit reposer que sur le secret de la clé, et donc tous les autres paramètres peuvent être supposés publiquement connus. Ce principe est devenu une pièce essentielle pour la conception d'un crypto-système. Son importance se reflète dans le fait de permettre à de nombreux cryptanalystes de s'attaquer aux crypto-systèmes existants et ainsi en assurer la robustesse.
Depuis le commencement de la cryptographie à clé publique, la recherche sur ce sujet n'a pas cessé de proposer de nouveaux systèmes et d'affaiblir ou renforcer

les existants. Cryptographes et cryptanalystes s'affrontent afin de concevoir et d'évaluer des systèmes à la fois rapides et fiables.

Exemple 1 (RSA)
Soient p, et q deux grands nombres premiers distincts (de longueur supérieur à 2048 bits par exemple), et $n = pq$. On note $\varphi(n) = (p-1)(q-1)$ l'indicateur d'Euler.

- La clé secrète est constituée de p, q, et d'un entier d premier avec $\varphi(n)$.
- La clé publique est constituée de n, et d'un entier e premier avec $\varphi(n)$ tel que : $ed = 1$ modulo $\varphi(n)$.
- L'opération de chiffrement est définie par : $x \mapsto x^e$ modulo n.
- L'opération de déchiffrement est définie par : $x \mapsto x^d$ modulo n.

∎

Bien qu'on ne sache pas encore le démontrer, la seule attaque générale connue actuellement contre RSA reste la factorisation de n. Cependant, certains choix malheureux des paramètres ou une mauvaise implémentation peuvent compromettre la sécurité. Selon les dernières recommandations de sécurité relatives au *Transport Layer Security* (TLS) publié par l' ANSSI [1] [ANS16], d'ici 2030, les clés RSA doivent avoir une taille minimale de 2048 bits, et les clés ECDSA (*Elliptic Curve Digital Signature Algorithm*) doivent avoir une taille minimale de 256 bits.
Les cryptosystèmes basés sur la théorie des nombres, tel que RSA, ne permettent malheureusement pas d'identifier le propriétaire d'une clé publique. Il a fallu donc trouver une solution qui permettra de créer ce lien entre la clé publique et l'identité de son possesseur dans le but de se protéger contre l'usurpation de l'identité. Pour répondre à ce besoin, Shamir en 1984 a introduit le concept de *la cryptographie basée sur l'identité* [Sha84] connu aussi sous le nom d'IBE (*Identity Based Encryption*). Dans ce type de crypto-systèmes la clé publique est remplacée par une chaîne quelconque de caractères (un identifiant, une adresse mail, un numéro de téléphone ...). La clé secrète est extraite grâce à une autorité de confiance à partir de la clé publique, et par conséquent, protège en plus contre l'usurpation d'identité.
Depuis que la notion d'IBE a été posé en 1984, plusieurs propositions de schémas IBE ont été proposées [DQ86, HJW00, MY91, TI89, Tan87]. Cependant, aucune de celles-ci n'est pleinement satisfaisante. En attendant de construire ce crypto-système, une autre solution a déjà été mise en œuvre, c'est ce qu'on appelle *la cryptographie basée sur les certificats*, aussi connue sous le nom de la PKI (*Public Key Infrastructure*). Il s'agit d'un ensemble de procédures qui combinent le chiffrement, le hachage et la signature numérique afin de garantir la confidentialité,

1. Agence Nationale de la Sécurité des Systèmes d'Information

l'intégrité, la non-répudiation et l'authentification.

1.3.3 La théorie de la complexité

La théorie de la complexité est un domaine central dans la théorie de l'information. Elle s'intéresse à l'étude de la complexité intrinsèque des tâches de calcul. La complexité algorithmique est une fonction qui exprime la consommation d'un programme en espace mémoire et/ou en temps de calcul durant son exécution en fonction de la taille des valeurs en entrée, ceci permet d'évaluer et comparer la performance des algorithmes. On dit qu'un algorithme est rapide si sa complexité algorithmique s'exprime par une fonction polynomiale (linaire, quadratique, cubique ...) ou logarithmique de la taille des entrées. D'autre part, un algorithme est lent si sa complexité est exponentielle.

Définition 1 (La machine de TURING)
Une machine de TURING est un modèle mathématique de calcul inventé en 1936 par ALAN TURING. Ce modèle définit une machine abstraite qui manipule des symboles appartenant à un alphabet Γ sur une bande (mémoire) selon un tableau de règles Q (les états). La mémoire est un ruban infini de cases qui contiennent chacune un élément de l'alphabet, la première case de ce ruban comporte un symbole unique d qui désigne le début de la mémoire, et la tête de lecture pointe toujours sur cette case au début du programme. Le tableau de règles comporte toujours un état initial (e_i) qui s'exécute en premier, un état final e_f qui arrête le programme, et des états intermédiaires qui dépendent de la machine. Chaque état permet d'exécuter des actions comme : déplacer la tête de lecture à droite (D) ou à gauche (G), écrire un symbole sur la case sélectionnée, ou bien effacer le contenu de la case.

Exemple 2
Soit $M = (\Gamma, Q)$ la machine de TURING qui permet de calculer le successeur d'un entier dans une représentation binaire. M est définie par :

- L'alphabet binaire $\Gamma = \{0, 1\}$;
- L'ensemble des états $Q = \{e_i, e_1, e_2, e_3, e_f\}$ tels que :
  - e_i : l'état initial qui pointe sur le premier caractère de la mémoire, et qui parcoure tout le ruban (vers la droite D) jusqu'à trouver la première case vide "_", ensuite passe à l'état e_1.
  - e_1 : la tête de lecture pointe sur le dernier chiffre du mot sur le ruban, s'il s'agit de 0 il le remplace par 1 et il passe à l'état final e_f, sinon il remplace le 1 par 0 et déplace la tête de lecture vers la gauche. L'état continue à chercher le dernier 0 du mot pour le remplacer par 1 et passer à l'état final. Si le mot n'est composé que de 1, alors l'état passe à e_2.

— e_2 : Le mot maintenant n'est composé que des 0, cet état permet de remplacer le premier caractère du mot par 1, et passer ensuite à l'état e_3.
— e_3 : Enfin l'état e_3 permet d'ajouter le caractère 0 à la fin du mot, et passer ensuite à l'état finale.

Soit n le mot stocké sur le ruban, pour calculer $n + 1$ par la machine de TURING M nous distinguons trois cas possibles :

	Le cas facile	la cas intermédiaire	Le pire des cas
	n est pair	n est impair	n est composé que de 1
$n =$	11010	10111	11111
e_i	d11010_	d10111_	d11111_
e_1	d11010_	d10110_	d11111_
e_2	-	-	d00000_
e_3	-	-	d10000_
e_f	d11011_	d11000_	d100000_

Dans le pire des cas, la tête de lecture pour cette machine parcoure le ruban $3log_2(n)$ fois, ce qui signifie que la complexité de cet algorithme est logarithmique.

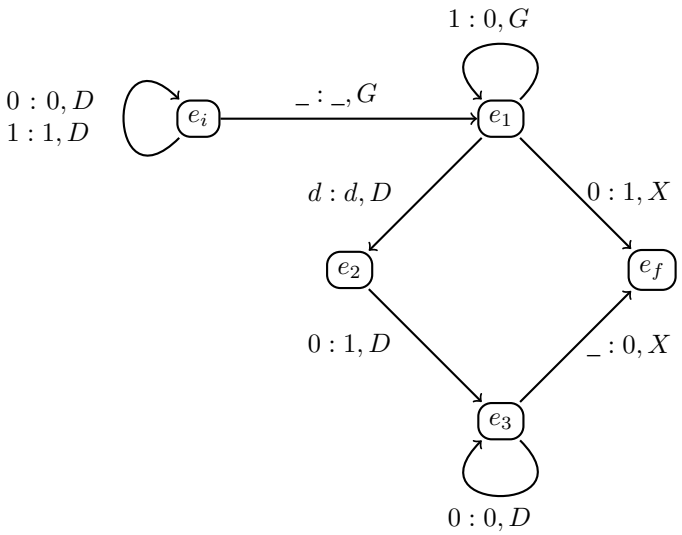


FIGURE 1.2 – Une machine de TURING pour calculer le successeur d'un entier

∎

Malgré la simplicité du modèle, il est possible pour n'importe quel algorithme de construire une machine de TURING (MdT) capable de le simuler. Elle est particu-

lièrement utile en informatique théorique, en particulier dans les domaines de la complexité algorithmique et de la calculabilité.
Nous distinguons deux familles de MdT :

- La MdT déterministe (MdTD) : il s'agit du modèle classique où chaque fonction de transition $f : \Gamma \times Q \to \Gamma \times Q \times \{D, G, X\}$ associe un seul triplet $(e', s', h) \in \Gamma \times Q \times \{D, G, X\}$ à tout couple $(e, s) \in \Gamma \times Q$ où e désigne l'état courant de la MdTD et s le symbole lu dans la case courante de la bande.
- La MdT non-déterministe (MdTND) : Dans ce modèle, la fonction de transition f peut associer plusieurs triplets (e', s', h) au même couple (e, s). Plus formellement la fonction de transition d'une MdTND est définie par : $f : \Gamma \times Q \to 2^{\Gamma \times Q \times \{D,G,X\}}$.

Il est toujours possible de construire un algorithme pour résoudre n'importe quel problème mathématique, cependant, il est difficile de déterminer s'il existe des algorithmes polynomiaux pour tous les problèmes existants. En effet, à ce jour, il existe des problèmes que nous ne savons pas encore résoudre en temps polynomial comme les problèmes de factorisation et du logarithme discret.

Définition 2 (le problème de factorisation)
Soit n un nombre composé, et $p_1, \ldots, p_u$ les facteurs premiers de n (i.e. $n = \prod_{i=1}^{u} p_i^{a_i}$ avec $a_i \geqslant 1$). Le problème de factorisation consiste à trouver l'ensemble $\{p_1, \ldots, p_u\}$ étant donné n.

Définition 3 (le problème du logarithme discret)
Soit G un groupe cyclique engendré par g d'ordre n :

$$G = \{1, g, g^2, \ldots, g^{n-1}\} \ .$$

Étant donné $y \in G$, le problème du logarithme discret consiste à trouver un entier $0 \leqslant u \leqslant n - 1$ tel que :

$$y = g^u \ .$$

Il existe deux façons de voir un problème :

- Problème de décision : il s'agit d'un problème dont la réponse est binaire (oui ou non), comme par exemple le test de primalité qui consiste à décider si un entier est premier ou pas.
- Problème de calcul : il s'agit de déterminer l'existence de la solution et de la trouver.

En d'autres termes, les problèmes de décision permettent de tester les solutions cherchées par les problèmes de calculs, par exemple, si nous posons le problème de calcul qui consiste à retrouver les facteurs premiers d'un entier n (le problème de factorisation), un problème de décision associé consiste à tester si un nombre premier p est facteur de n.

Étant donné qu'il existe toujours des problèmes que nous savons pas résoudre en temps polynomial, et que nous avons aucune preuve sur l'existence d'une telle solution. les spécialistes en théorie de la complexité ont classé les problèmes mathématiques en 3 classes fondamentales :

- La classe P : L'ensemble des problèmes de décision qui admettent un algorithme de résolution en temps polynomial.
- La classe NP : classe des problèmes où les solutions des problèmes de décision peuvent être vérifiées en temps polynomial par une MdTND.
- La classe NP complet (NPC) : un problème est NPC s'il est NP, et en plus, il existe une réduction polynomiale de tous les problèmes NP vers celui-ci. Cela implique que le problème NPC est au moins aussi difficile que tous les autres problèmes de la classe NP.

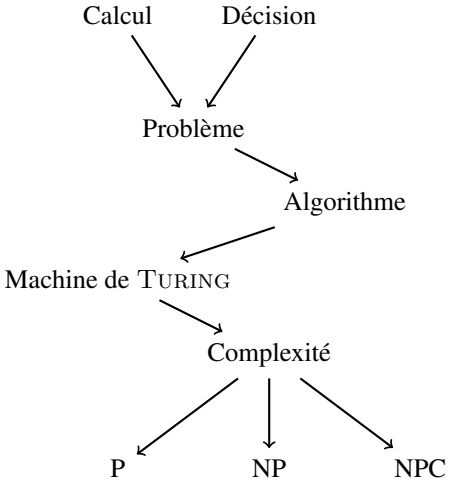


FIGURE 1.3 – Le rôle des MdT pour classifier les problèmes mathématiques

1.3.4 Les fonctions de hachage

Définition 4 (Fonction à sens unique)
Une fonction à sens unique est une fonction qui peut être aisément calculée, mais qui est difficile à inverser.

Une fonction de hachage est une fonction à sens unique qui permet de calculer une empreinte de longueur fixe à partir d'une donnée de taille quelconque fournie en entrée, cette empreinte peut ensuite servir à identifier rapidement la donnée initiale.

$$\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^k \ .$$

Les fonctions de hachage ont une importance majeure en informatique, et particulièrement dans le domaine de la sécurité. Par exemple, pour éviter de stocker en clair les mots de passe sur une machine pour des raisons de sécurité, les systèmes d'exploitation stockent uniquement l'empreinte du mot de passe, ainsi, il suffit de comparer l'empreinte stockée avec celle du mot de passe demandé. Une autre utilisation importante des fonctions de hachage est la signature numérique.
Il existe une grande variété de fonctions de hachage : *Secure Hash Algorithm* (SHA), *Message Digest* (MD), *The Elliptic Curve Only Hash* (ECOH), HAVAL, Tiger... La famille SHA a été conçue par la NSA [2] et publiée par NIST [3] pour être utilisée avec la signature numérique standard *Digital Signature Standard* (DSS). Il existe une similitude entre la famille SHA et la famille MD. Cependant, la famille SHA est composée de fonctions mathématiques supplémentaires et produit une empreinte de 160 bits au lieu de 128 bits, ce qui la rend plus résistante face aux attaques à force brute, y compris les attaques basées sur le paradoxe des anniversaires qui consiste à chercher des collisions.

Définition 5 (Le paradoxe des anniversaires)
Le paradoxe des anniversaires concerne la probabilité que, dans un ensemble de n personnes choisies au hasard, une paire d'entre eux ait le même anniversaire. La probabilité n'atteint 100% que lorsque le nombre de personnes atteint 367 (puisqu'il n'y a que 366 anniversaires possibles, y compris le 29 février). Cependant, la probabilité de $99,9\%$ est atteinte avec seulement 70 personnes, et 50% de probabilité avec 23 personnes.
Bien entendu, il ne s'agit pas d'un paradoxe dans le sens d'une contradiction logique, mais parce qu'il s'agit d'une vérité mathématique qui contredit l'intuition. En effet, intuitivement, la probabilité pour 23 personnes est très inférieure à 50%.

Définition 6 (Fonction de hachage)
Une fonction de hachage $\mathcal{H}$ est dite résistante aux collisions s'il est difficile de trouver deux entrées qui hachent la même sortie ; c'est-à-dire qu'il est difficile de trouver deux entrées x et y telles que $\mathcal{H}(a) = \mathcal{H}(b)$ avec $a \neq b$.

1.3.5 La signature numérique

La signature numérique est une des applications de la cryptographie à clé publique, il s'agit de la transposition dans le monde numérique de la signature manuscrite. Elle permet de garantir l'identité du signataire, l'intégrité et la provenance du document signé. Pour signer un document ou un message m, il est possible (pour

2. National Security Agency
3. National Institute of Standards and Technology

certain algorithmes comme RSA) de chiffrer l'empreinte de ce message $\mathcal{H}(m)$ avec la clé secrète du signataire SK.

$$Sign(m) = Enc(\mathcal{H}(m), SK) \ .$$

Ainsi, pour vérifier la signature, il suffit de déchiffrer la signature du message avec la clé publique du signataire et tester si elle est identique à l'empreinte du message :

$$\mathcal{H}(m) \stackrel{?}{=} Dec(Sign(m), PK) \ .$$

Type d'algorithme	Chiffrement	Signature	Hachage	Échange de clés
La cryptographie asymétrique				
RSA	✓	✓	-	✓
ECC	✓	✓	-	✓
Diffie-Hellman	-	-	-	✓
El Gamal	✓	✓	-	✓
DSA	-	✓	-	-
La cryptographie symétrique				
AES	✓	-	-	-
3DES	✓	-	-	-
Blowfish	✓	-	-	-
IDEA	✓	-	-	-
RC4	✓	-	-	-
SAFER	✓	-	-	-
Les fonctions de hachage				
MD2, MD4, MD5	-	-	✓	-
sha1,sha224, sha256, ...	-	-	✓	-
HAVAL	-	-	✓	-
Tiger	-	-	✓	-

TABLE 1.1 – Les fonctionnalités des différents algorithmes cryptographiques

Une signature numérique qui possède une validité juridique au niveau Européen permet les fonctionnalités suivantes :

Caractère unique : Chaque document, dossier ou transaction est signé(e) avec une signature unique, et donc non réutilisable.

Intégrité : La signature ne peut être ni modifiée ni altérée. Toute tentative de modification sera automatiquement détectée.

Incontestable : La signature ne doit pouvoir être réalisée que par la personne possédant le certificat correspondant.

Vérification : L'identité du signataire doit être confirmée par une autorité de confiance.

Il faut bien noter que ces garanties sont assurées sous l'hypothèse que le problème mathématique utilisé (par exemple la factorisation pour le cas de RSA) est un problème difficile. Donc, d'un point de vue théorique ; il existe toujours une probabilité très négligeable de reproduire une fausse signature.
La fonction de hachage garantit l'intégrité du message, la signature de l'empreinte fournit l'authentification et la non-répudiation, et le chiffrement permet d'assurer la confidentialité.
Dans le tableau 1.1 nous avons listé quelques algorithmes cryptographiques connus.

1.4 L'infrastructure à clé publique

1.4.1 Le protocole de Diffie-Hellman

Le protocole de Diffie-Hellman (DH) du nom de ses auteurs Diffie et Hellman [DH76] est une méthode d'échange sécurisé de clés cryptographiques sur un canal public. La sécurité de cette méthode est basée sur le problème du logarithme discret (voir la définition 3 page 9).
L'implémentation la plus simple (qui est celle d'origine) de ce protocole utilise le groupe multiplicatif d'entiers modulo p, où p est premier, et g est une racine primitive modulo p (un générateur). Cela permet que le secret partagé qui en résulte ait une distribution uniforme sur l'intervalle 1 à $p - 1$.
Le processus commence quand deux parties (Alice et Bob par exemple), s'accordent sur les paramètres publics p et g, ensuite :

- Alice choisit un nombre secret a, et calcule la valeur publique : g^a ;
- Bob choisit également un nombre secret b, et calcule la valeur publique : g^b ;
- Alice et Bob échangent leurs valeurs publiques g^a et g^b ;
- Alice calcule : $g^{ab} = (g^b)^a$.
- Bob calcule : $g^{ab} = (g^a)^b$.

Ainsi, la clé partagée devient : g^{ab} que seuls, Alice et Bob semblent capables de construire.

1.4.2 L'attaque de l'homme au milieu

"*Man-In-The-Middle*" (MITM) qu'on peut traduire en français par *l'homme au milieu* est une attaque basée sur l'usurpation d'identité, et qui vise les protocoles

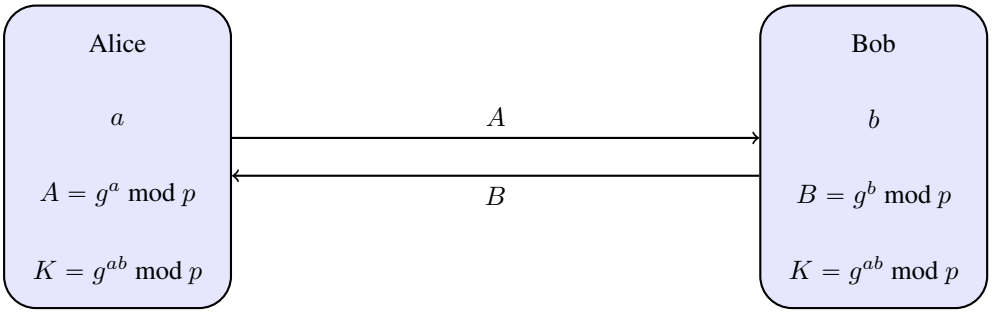


FIGURE 1.4 – Diffie-Hellman

d'échange de clés. L'attaquant s'introduit discrètement au milieu d'une communication entre deux parties qui croient communiquer directement entre elles. Il établit ensuite des connexions indépendantes avec les victimes et relaie les messages entre eux pour leur faire croire qu'ils parlent directement les uns aux autres sur une connexion privée, alors qu'en réalité toute la conversation est contrôlée par l'attaquant.

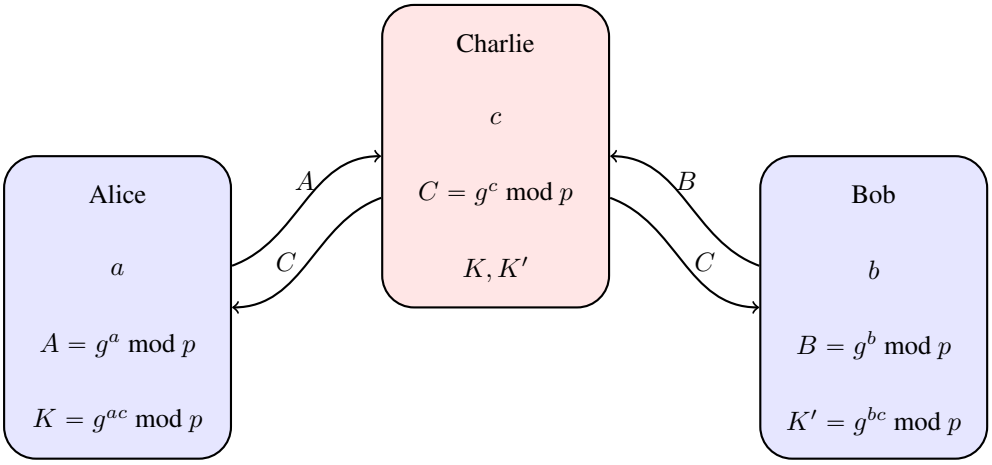


FIGURE 1.5 – L'attaque de l'homme du milieu

Pour le protocole DH (Figure 1.5), l'attaquant (Charlie) intervient au moment de l'échange de clé entre Alice et Bob, il fait croire à Alice que c'est lui Bob, et lui envoie sa clé publique C au lieu de B, de la même façon, il fait croire à Bob que c'est lui Alice et lui envoie sa clé publique C au lieu de A. A la fin, l'attaquant Charlie possède deux clés privées K et K' qui lui permettront de déchiffrer toutes les communications entre Alice et Bob. Pour rester transparent, Charlie doit déchiffrer tous les messages d'une des victimes et les rechiffrer par la clé du destinataire.

La méthode la plus simple pour résoudre ce problème d'usurpation d'identité consiste à faire signer toutes les clés par une autorité de confiance. Le document obtenu par ce procédé s'appelle alors *un certificat*.

1.4.3 Le principe des certificats

La cryptographie à clé publique représente une avancée remarquable dans l'histoire de la cryptographie. Elle permet aux utilisateurs d'échanger un secret sans être en contact direct. Cependant, un problème se présente, il s'agit de l'authenticité des clés publiques. En effet, la clé publique ne porte aucune information sur l'identité de son possesseur. Les utilisateurs doivent alors s'assurer de l'authenticité des clés publiques pour lutter contre l'usurpation d'identité.
L'infrastructure de gestion de clés (IGC), aussi connue sous le nom de PKI : *Public Key Infrastructure*, se compose de programmes, de formats de données, de procédures, de protocoles de communication, de politiques de sécurité et de mécanismes cryptographiques à clé publique fonctionnant de manière globale afin de permettre à un large éventail de personnes dispersées de communiquer de manière sécurisée et normalisée. En d'autres termes, une PKI établit un niveau de confiance dans un environnement. La PKI est un cadre d'authentification ISO [4] qui utilise la cryptographie à clé publique et la norme X.509. Le cadre a été configuré pour permettre l'authentification sur différents réseaux privés ou publics. Les protocoles et les algorithmes particuliers ne sont pas spécifiés, c'est pourquoi la PKI est considérée comme un cadre et non une technologie spécifique.
La PKI fournit l'authentification, la confidentialité, la non-répudiation et l'intégrité des messages échangés. C'est un système hybride d'algorithmes de chiffrement symétriques et asymétriques, qui ont été discutés dans des sections antérieures.
Il existe une différence entre la cryptographie à clé publique et la PKI. La cryptographie à clé publique est un type d'algorithme de chiffrement, tandis que la PKI comme son nom l'indique est une infrastructure. L'infrastructure suppose que l'identité du récepteur peut être assurée facilement par des certificats et qu'un algorithme asymétrique effectuera automatiquement le processus d'échange de clés. L'infrastructure contient de ce fait les éléments qui identifieront les utilisateurs, créeront et distribueront des certificats, les maintiendront et révoqueront des certificats, distribueront et maintiendront des clés de chiffrement et permettront à toutes les technologies de communiquer et de travailler ensemble dans un environnement sécurisé.
Un certificat électronique est un document numérique permettant de valider le lien entre une clé publique et son propriétaire. Il contient principalement les éléments suivant :

4. International Organization for Standardization

— Une clé publique ;
— L'identité du dépositaire de la clé secrète associée à cette clé publique ;
— Une plage de validité du certificat : date de début - date de fin ;
— L'identité de l'autorité de certification (CA) ;
— Une description de l'algorithme de chiffrement compatible avec la clé publique, ainsi que la longueur des clés ;
— Une description de l'algorithme de signature utilisé pour signer ce certificat ;
— Une description de la fonction de hachage utilisée pour la signature ;
— La signature électronique des informations précédentes par l'autorité de certification (CA).

Un certificat électronique doit respecter un format standard appelé " *la norme X.509* ". Cette norme a été définie par l'union internationale des télécommunications (UIT) en 1988 pour s'adapter aux différents protocoles informatiques (TLS, S/MIME, SET, IPsec, ...).

1.4.4 Le fonctionnement de l'infrastructure de gestion de clés (PKI)

Plusieurs autorités de rôles différents s'occupent de la gestion des certificats dans une PKI :

— *Registration Authority* (RA) : L'autorité d'enregistrement qui authentifie les entités et génère le couple (clé secrète, clé publique) durant la phase d'enrôlement.
— *Certification Authority* (CA) : L'autorité de certification, qui signe les demandes des certificats et les listes des révocations.
— *Directory Service* (DS) : Le service d'annuaire utilise la norme LDAP (*Lightweight Directory Access Protocol*) pour stocker de manière sécurisée tous les certificats, et les fournir aux utilisateurs (exemple : *Active directory* de Microsoft).
— *Revocation Service* (RS) : Le service de révocation qui gère la liste des certificats révoqués CRL (Certificate Revocation List).

Pour bien administrer la génération et la distribution des clés par ces différentes autorités, il faut respecter un protocole précis. Il existe trois phases importantes dans ce protocole :

1. **La phase d'enrôlement :** Cette phase décrit comment générer un certificat d'utilisateur. Pour ce faire, ce dernier s'authentifie auprès de l'autorité d'enregistrement (RA). Il s'agit souvent d'un officier de sécurité qui assure le bon fonctionnement du système. Si l'autorité décide de valider cette requête, elle génère un couple de clés secrète et publique au format "csr" (Certificate

Signing Request) pour l'utilisateur[5]. Elle les envoie ensuite à l'utilisateur en question via un canal privé. L'utilisateur à son tour envoie sa requête (la clé publique), à l'autorité de certification (CA) pour la signer. Enfin, la CA génère un certificat associé à la clé publique, et l'identité de son possesseur. Une copie de ce certificat est ensuite placée dans la base de données du système qui est gérée par le service d'annuaire (DS), pour qu'elle puisse être accessible aux autres utilisateurs.

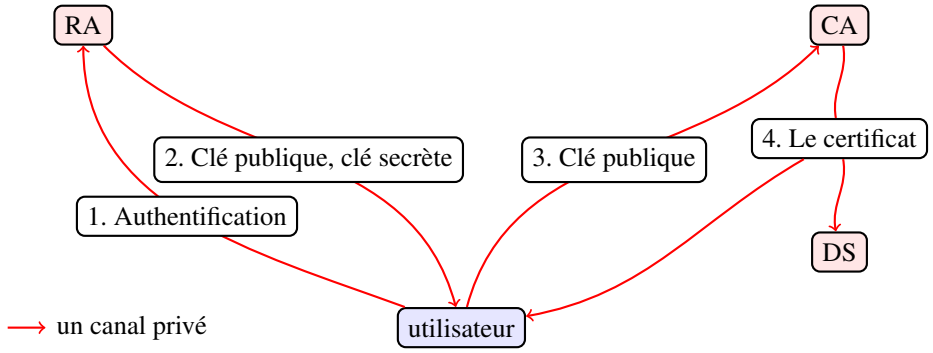


FIGURE 1.6 – La phase d'enrôlement dans une PKI

2. **La phase de communication :** Cette phase décrit le protocole à respecter pour une communication sécurisée entre deux utilisateurs. On prend l'exemple d'Alice et Bob qui essaient de communiquer, Alice représente l'émetteur, et Bob le récepteur. Alice commencera d'abord par récupérer le certificat de Bob (`Bob.cert`) au près du DS. Elle devra ensuite vérifier les éléments suivants :
   — l'identité qui figure sur le certificat est-elle effectivement celle du récepteur, en l'occurrence Bob ;
   — la signature de la CA figurante sur le certificat ;
   — la non-expiration du certificat ;
   — l'absence de révocation du certificat auprès de la CRL.
   Une fois tous ces tests satisfaits, Alice pourra envoyer son message chiffré avec la clé qui figure sur le certificat.
3. **La phase de révocation :** La phase de révocation est une pièce indispensable dans un système de gestion de clés. C'est le seul moyen d'exclure un utilisateur qui possède déjà un certificat valide. Pour révoquer un certificat, le plus simple est de générer une liste des certificats révoqués (CRL), ce fichier au format PEM (*Privacy Enhanced Mail*) est signé par la suite par la

5. Idéalement, c'est l'utilisateur qui est censé générer ses propre clés, mais souvent, en pratique c'est la RA qui s'en charge pour des raisons de sécurité, telle que la sauvegarde des clés, et l'assurance de leurs niveau de résistance face aux différentes attaques.

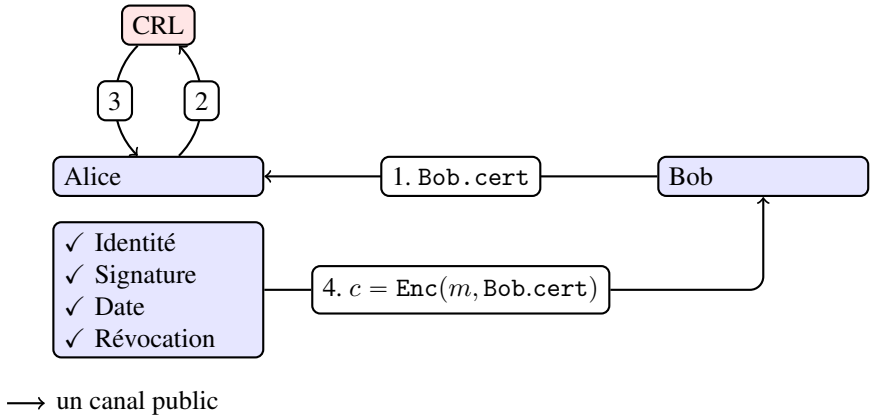


FIGURE 1.7 – La phase communication dans une PKI

CA pour éviter toute falsification. Enfin, la CRL est publiée sur le réseau pour la rendre accessible à tous les utilisateurs. Il est possible que les utilisateurs s'informent en quasi temps réel de l'état du certificat avec le protocole OCSP [6]. Il est également primordial de mettre à jour la CRL de manière périodique pour que les utilisateurs puissent se synchroniser.

Il existe plusieurs logiciels libres pour implémenter une PKI tels que `EJBCA`, `OpenCA`, `OpenXPKI`... la plupart de ces logiciels sont basés sur des bibliothèques cryptographiques libres comme `Openssl` ou `GnuPG`.

1.5 Le chiffrement basé sur l'identité (IBE)

1.5.1 Le principe général

En 1984, dans [Sha84] SHAMIR a introduit un type de chiffrement asymétrique dans lequel la clé publique pourrait être une chaîne librement choisie. Ce type de cryptographie s'appelle " *La cryptographie basée sur l'identité* " et elle est connue sous le nom de *Identity Based Encryption* (IBE). Dans un tel schéma, il existe quatre algorithmes :

1. `Configuration` : génère les paramètres globaux du système et une clé maître ;
2. `Extraction` : utilise la clé maître pour générer la clé privée associée à une chaîne de caractère ;
3. `Chiffrement` : chiffre les messages à l'aide de l'identité du destinataire ;
4. `Déchiffrement` : déchiffre les messages à l'aide de la clé privée correspondante.

6. Online Certificate Status Protocol

La principale innovation de ce schéma est l'utilisation de l'identité de l'utilisateur (adresse mail, Numéro de téléphone, identifiant...) en tant que clé publique pour le chiffrement et la vérification de signature. Ceci réduit considérablement la complexité de la gestion des clés. En effet, dans un tel schéma, contrairement au cas de la PKI, il n'est nul besoin de créer ni de gérer les certificats. De plus, une seule autorité est nécessaire pour gérer le système. Cette autorité s'appelle *Private Key Generator* (PKG). Elle sert comme son nom l'indique à générer (extraire) la clé secrète de l'utilisateur à partir de son identité grâce à la clé maître.

Après que le problème a été posé en 1984, il y a eu plusieurs propositions pour les schémas IBE [DQ86, HJW00, MY91, TI89, Tan87]. Mais jusqu'aux résultats de Boneh et Franklin [BF01], aucune de ces propositions n'était totalement satisfaisante. Certaines solutions exigeaient que les utilisateurs ne se connaissent pas. D'autres solutions exigeaient que le PKG passe beaucoup de temps pour chaque demande de génération de clé privée. Certaines solutions nécessitaient un matériel anti-effraction. La figure 1.8 résume l'architecture d'IBE pour la phase d'enrôlement (a, b) et la phase de communication (1, 2, 3, 4).

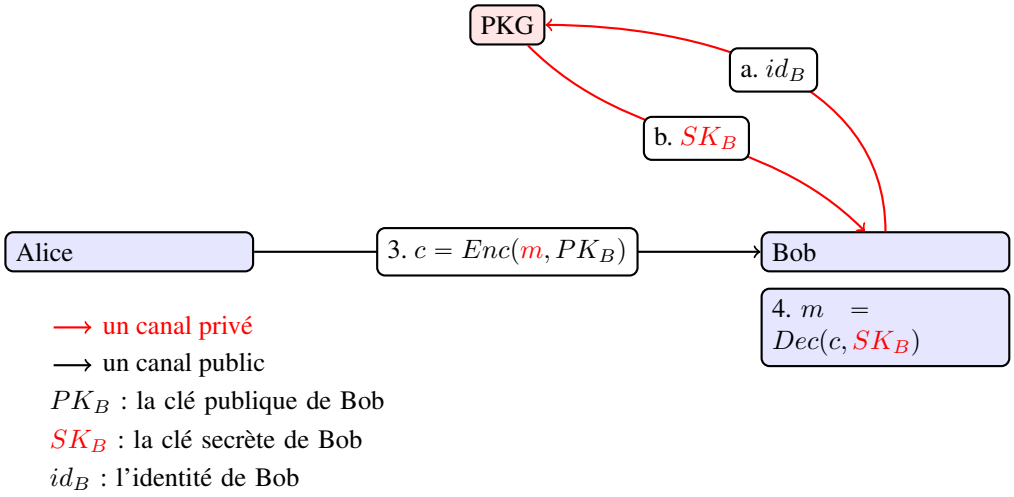


Figure 1.8 – La distribution des clés dans un schéma IBE

La mise en œuvre de l'IBE de Boneh et Franklin repose sur un groupe bilinéaire composé des points d'une courbe elliptique et d'une fonction bilinéaire correspondante pour effectuer les calculs nécessaires. Dans la partie II, nous allons étudier de manière détaillée ces outils. Nous étudierons les améliorations proposées au fur et à mesure par la communauté durant ces dernières années. Nous proposerons enfin l'architecture, qui, selon nous, répond aux exigences du monde industriel notamment celles des infrastructures critiques telles que les "smart grid" [ABB+15].

1.5.2 Les exigences d'un système de gestion de clé

Un système de gestion de clé doit effectuer plusieurs tâches. Dans un environnement d'entreprise par exemple, qu'il s'agisse de protéger les e-mails, les bases de données ou les documents. Nous avons regroupé les critères nécessaires pour un système efficace en six exigences :

1. **Fournir des clés de chiffrement.** Les utilisateurs et les applications doivent pouvoir chiffrer les données pour tous les destinataires. Si un utilisateur ne peut pas accéder facilement à la clé publique d'une grande variété de destinataires internes, groupes d'annuaires, clients ou partenaires, il se peut qu'il lui soit impossible de chiffrer ces données. Dans ce cas si un utilisateur ne peut pas facilement chiffrer les données, le processus métier [7] cessera de fonctionner ou, pire, l'utilisateur stockera en clair les données sur des supports non protégés.
2. **Authentifier les utilisateurs et livrer les clés de déchiffrement.** Un système de gestion des clés doit pouvoir à la fois authentifier et fournir des clés aux groupes et aux utilisateurs. Le système d'authentification doit s'intégrer à l'infrastructure existante, sinon les coûts d'infrastructure, et d'authentification des utilisateurs seront doublés.
3. **Gérer conjointement les clés avec les partenaires.** Dans de nombreux cas, il est souhaitable d'échanger des données chiffrées avec des partenaires, dans lesquels chaque partenaire gère uniquement les clés pour ses propres utilisateurs. De cette façon, un message de la société A peut s'adresser aux destinataires de la société B sans exiger que la société A connaisse comment authentifier les utilisateurs de la société B. La société B peut donc gérer sa propre infrastructure d'authentification et de gestion des messages sans avoir besoin de coordonner avec la société A.
4. **Fournir des clés aux composants d'infrastructure de confiance.** Lors du chiffrement du courrier ou des fichiers dans une entreprise, les processus métiers et techniques intermédiaires, tels que l'audit, la numérisation de contenu ou les antivirus, peuvent nécessiter l'accès au contenu des données chiffrées. Le système de gestion des clés doit être capable de fournir des clés sur ces composants.
5. **Récupérer les clés.** Dans le cas d'un utilisateur révoqué ou quittant l'organisation, un administrateur doit pouvoir accéder aux données chiffrées. Les récentes règles concernant le "*e-discovery*" [8] ont fait de cette exigence une

7. Un processus métier est une collection d'activités ou de tâches structurées ou liées qui produisent un service ou un produit spécifique (servir un objectif particulier) pour un client.

8. La procédure *e-discovery* incite les entreprises à archiver automatiquement tous les documents et communications électroniques dans le cadre de la recherche de preuves pouvant être

importance primordiale. Si un système de gestion de clés laisse un élément de données dans un état où il ne peut pas être déchiffré, cela peut représenter une responsabilité légale importante et une perturbation des processus métiers.

6. **Étendre le réseau.** Les entreprises prospères sont amenées à grandir, et parfois elles le font très rapidement. Un système de gestion de clé efficace doit tenir compte d'un volume de transaction croissant et être déployable dans des locaux géographiquement dispersés. Les entreprises doivent envisager comment la gestion des clés fonctionnera si, et quand, elles rencontreront 10 ou même 1000 fois le volume de transactions qu'elles traitent actuellement.

1.5.3 Comparaison entre PKI et IBE

Pour comparer deux architectures de gestion de clés publiques telles que la PKI et IBE, nous allons d'abord étudier la compatibilité entre ces systèmes et la liste des six exigences listées auparavant.

1.5.3.1 La PKI

1. Les systèmes à clé publique ont un problème inhérent relatif à la localisation d'une clé publique d'un utilisateur spécifique. Les utilisateurs des systèmes basés sur des certificats sont souvent confrontés au fait qu'ils ne peuvent pas trouver un certificat pour un utilisateur spécifique. Ce qui rend la communication sécurisée impossible. Les groupes présentent un problème similaire, car la mise en œuvre des certificats pour les groupes nécessite un système de distribution de clés privées et le schéma d'authentification des certificats doit permettre de transférer les utilisateurs à l'intérieur et à l'extérieur des groupes.
2. En général, l'authentification des utilisateurs dans un système basé sur les certificats doit se produire avant que l'utilisateur puisse recevoir un message. Cela signifie qu'il doit y avoir une étape d'inscription (enrôlement) qui se déroule avant que les données puissent être chiffrées. Dans le cas d'un utilisateur provisoire il peut s'agir d'un défaut insurmontable. En supposant que cette étape soit surmontée, le destinataire peut déchiffrer les données tant que la clé privée correspondante est disponible.
3. Les communications entre partenaires peuvent être effectuées dans un système de chiffrement à clé publique, mais un répertoire partagé doit être mis en place.

utilisées dans un procès civil ou commercial.

4. Pour fournir des clés à un système d'archivage intermédiaire, un système de partage de clé privée doit être déployé.
5. Les clés peuvent être récupérées si une base de données de clé privée est maintenue, par exemple un HSM [9]. Si cette base de données est compromise ou endommagée, les clés peuvent devenir irrécupérables. Les clients peuvent être programmés pour chiffrer des données vers un intermédiaire de confiance, ce qui permet de récupérer les messages via cet intermédiaire.
6. Théoriquement, les systèmes à clé publique peuvent s'adapter en fonction de la taille du réseau. Cependant, l'expérience avec les grands déploiements a montré que les problèmes d'interface utilisateur concernant la gestion et la révocation des certificats et des clés rendent l'utilisation à grande échelle de la PKI très coûteuse.

1.5.3.2 IBE

1. En utilisant IBE, les clés sont toujours disponibles pour tous les destinataires. La clé de chiffrement est dérivée simplement de l'identité du récepteur. Les groupes sont traités aussi facilement, car une clé peut être faite à partir d'un nom de groupe aussi facilement qu'un nom individuel.
2. IBE s'interface avec les infrastructures d'authentification existantes, de sorte que toutes les ressources d'authentification déjà déployées (par exemple, les répertoires ou l'authentification web) peuvent être réutilisées. L'expérience client pour obtenir une clé de déchiffrement peut être identique à la connexion à un portail, par exemple. Nous allons voir dans la partie II que nous pouvons améliorer l'IBE de Boneh et Franklin pour pouvoir distribuer les clés secrètes sur un canal public en utilisant le masquage.
3. IBE permet à l'expéditeur de sélectionner un serveur de clé local, un serveur de clé d'un partenaire ou un service pour protéger les données, en fonction de ses besoins particuliers.
4. Parce qu'IBE génère mathématiquement toutes les clés de déchiffrement sur le serveur (PKG), le serveur peut régénérer de manière sécurisée les clés des composants d'infrastructure au besoin.
5. Dans un système basé sur IBE, toutes les clés sont générées à partir de la clé maître stockée sur le serveur de clé. Tant qu'elle est récupérable, toute clé peut être régénérée de manière sécurisée.

9. *Hardware Security Module* : Il s'agit d'un dispositif informatique physique qui protège et gère les clés cryptographiques pour une authentification forte. Ces modules se présentent généralement sous la forme d'une carte à puce ou d'un périphérique externe qui se connecte directement à un ordinateur ou à un serveur réseau.

6. IBE permet d'ajouter des applications et des transactions supplémentaires avec très peu ou pas d'infrastructure de gestion de clé supplémentaire. Les serveurs de clés peuvent fonctionner indépendamment, ce qui autorise une dispersion géographique et un bon équilibrage de charges.

1.5.4 Conclusion

Les approches traditionnelles de la gestion des clés, y compris la gestion des clés symétriques et, plus récemment, la PKI, n'ont pas été convaincantes au regard des exigences d'un système de gestion de clé. De plus, elles sont difficiles à utiliser, à mettre en œuvre et à gérer, et extrêmement coûteuses. En revanche, le chiffrement basé sur l'identité (IBE) répond parfaitement aux exigences en terme de chiffrement des données, d'authentification, de gestion des clés avec des partenaires extérieurs, de fourniture des clés aux composants d'infrastructure fiables, de restauration des clés et d'adaptation avec la croissance envisageable du réseau dans le future.

1.6 Les codes correcteurs

La théorie du codage est née avec la publication de Shannon intitulée "*A mathematical theory of communication*" sur la théorie de la communication [SW63] et les travaux de Hamming sur les codes [Ham50]. Au cours du demi-siècle écoulé, la théorie du codage s'est transformée en une discipline en protection de l'information qui croise les mathématiques et l'informatique et qui a des applications dans presque tous les domaines de la communication tels que la transmission par satellite ou par téléphone cellulaire, l'enregistrement sur des supports de stockage comme les disques, les clés USB, et d'autres supports. Le rôle des codes correcteurs est de corriger les erreurs de transmission ou de lecture de données numériques, ou les erreurs survenant au cours de leur inscription sur un support physique ou encore lorsque les données subissent une altération sur le support de stockage.
Le message transmis est constitué de "mots" de longueur fixe k et écrits dans un alphabet $\mathcal{A}$ (dans le cas binaire l'alphabet est l'ensemble $\{0, 1\}$ que l'on notera $\mathbb{F}_2$) ; chaque mot subit par la suite une opération d'encodage qui donne aux mots transmis une certaine structure afin de pouvoir détecter et corriger les erreurs potentielles. Pour y parvenir, les codes augmentent la longueur du message à transmettre. On obtient alors un mot de code d'une longueur $n > k$ qui peut être transmis sur un canal bruité ou stocké sur un support susceptible de subir des modifications. La structure du mot de code (qui dépend du code utilisé lors de l'encodage) permet de reconstituer le message initialement envoyé, même lorsqu'un nombre suffisamment faible de ses bits est perdu ou altéré. La figure 1.9

schématise le système de communication sur un canal bruité.

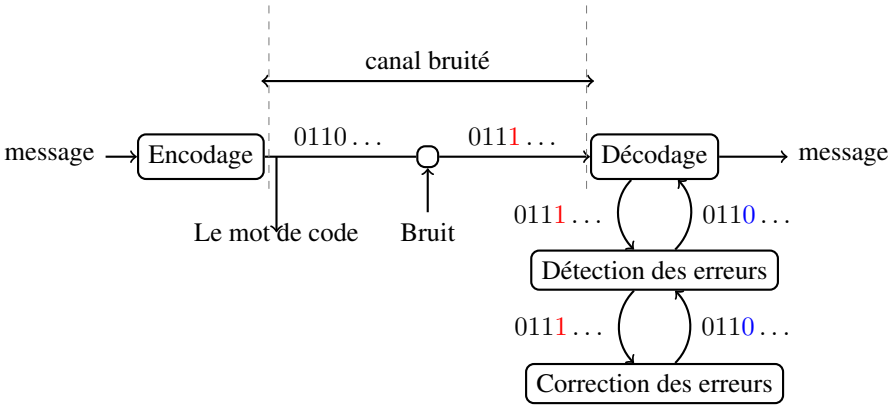


FIGURE 1.9 – Le schéma général d'un système de communication

Il existe plusieurs familles de codes correcteurs d'erreurs, qui peuvent être linéaires ou pas. Dans cette partie, c'est la famille des codes linéaires qui retiendra particulièrement notre attention. D'ailleurs, nous l'utiliserons ultérieurement pour présenter une solution de masquage basée sur ce type de codes.

1.6.1 L'encodage

Soit $\mathbb{F}_q$ un corps fini à q éléments. Un code linéaire $\mathcal{C}$ de dimension k et de longueur n est un sous-espace vectoriel de $\mathbb{F}_q^n$ de dimension k. Si $q = 2$ le code est dit binaire, sinon, on parle de code linéaire de base q. L'application d'encodage pour un code linéaire $\mathcal{C}$ est une application linéaire injective de $\mathbb{F}_q^k$ dans $\mathbb{F}_q^n$ (voir la figure 1.10).

Définition 7 (la matrice génératrice)
Soit $\mathcal{C}$ un code linéaire de longueur n et de dimension k dans $\mathbb{F}_q^n$. On appelle une matrice génératrice de $\mathcal{C}$ toute matrice $G \in \mathbb{F}_q^{k\times n}$ dont les lignes forment une base de $\mathcal{C}$:

$$\forall c \in \mathcal{C} \quad \exists! x \in \mathbb{F}_q^k \quad | \quad xG = c \ .$$

Un code linéaire est donc défini ainsi :

$$\mathcal{C} = \left\{ c = xG \quad | \quad \forall x \in \mathbb{F}_q^k \right\} \ . \tag{1.1}$$

Définition 8 (Le poids de HAMMING)
Le poids de HAMMING d'un vecteur binaire v correspond au nombre de 1 qu'il contient, et on le note $\omega_H(v)$.

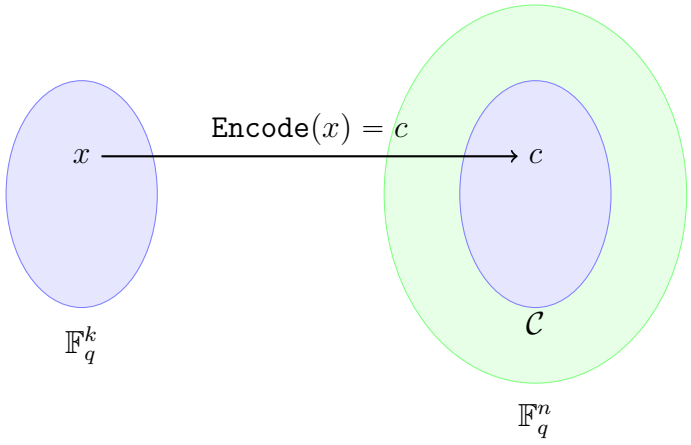


FIGURE 1.10 – L'encodage pour les codes linéaires

Définition 9 (La distance de HAMMING)
La distance de HAMMING entre deux vecteurs binaires u et v est égale au poids de HAMMING de $u - v$, et on la note $d_H(u, v)$.

Définition 10 (La distance minimale)
Étant donné un code $\mathcal{C}$, sa distance minimale $d_\mathcal{C}$ est définie comme la distance de HAMMING minimale entre deux mots de $\mathcal{C}$:

$$d_\mathcal{C} = \min \{d_H(x, y) \quad | \quad x, y \in \mathcal{C}, x \neq y\} \ . \tag{1.2}$$

Les paramètres d'un code linéaire sont notés $[n, k, d_\mathcal{C}]$, avec n la longueur du code, k la dimension, et $d_\mathcal{C}$ la distance minimale.

1.6.2 Le décodage

Définition 11 (Le code dual)
Le code dual d'un code linéaire $\mathcal{C}$ est le sous espace vectoriel de $\mathbb{F}_q^n$ noté $\mathcal{C}^\perp$ et égale à :

$$\mathcal{C}^\perp = \left\{d \in \mathbb{F}_q^{n-k} \quad | \quad \forall c \in \mathcal{C}, \langle c, d\rangle = 0\right\} \ , \tag{1.3}$$

où $\langle \cdot, \cdot \rangle$ désigne le produit scalaire usuel modulo q.

Dans le reste du document nous allons noter $\mathcal{D}$ le code dual de $\mathcal{C}$. On appelle matrice de parité (aussi appelée : matrice de contrôle) la matrice génératrice du code dual ; on la note souvent $H \in \mathbb{F}_q^{(n-k)\times n}$. Nous pouvons donc construire le code dual ainsi :

$$\mathcal{D} = \left\{d = yH \quad | \quad \forall y \in \mathbb{F}_q^{n-k}\right\} \ . \tag{1.4}$$

Propriété 1
Soient $\mathcal{C}$ un code linéaire de paramètres $[n, k, d]$, $\mathcal{D}$ son code dual, et $G \in \mathbb{F}_q^{k\times n}$ et $H \in \mathbb{F}_q^{(n-k)\times n}$ les matrices génératrices de $\mathcal{C}$ et $\mathcal{D}$ respectivement. On a :

$$G.H^T = 0 \ . \tag{1.5}$$

Propriété 2 (Capacité de correction)
La capacité de correction (maximale) d'un code linéaire dépend de sa distance minimale d, et elle vaut :

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor \ . \tag{1.6}$$

La méthode la plus simple (aussi coûteuse) pour décoder s'appelle le "décodage par syndrome". Soit $x \in \mathbb{F}_q^k$ l'information à coder, $c = xG \in \mathbb{F}_q^n$ le mot de code associé à x, et $e \in \mathbb{F}_q^n$ un vecteur d'erreur, tel que $\omega_H(e) \leqslant t$, avec t la capacité de correction. On note $v = c + e$ le vecteur reçu. Pour détecter la présence des erreurs il suffit de calculer le syndrome : $\varepsilon = vH^T = (c+e)H^T = xGH^T + eH^T = eH^T$ (par l'équation 1.5), si le syndrome est nul alors $e = 0$, sinon $e \neq 0$. Pour corriger l'erreur, il faut parcourir tous les vecteurs e' de poids inférieur ou égal à t et vérifier si $e'H^T \stackrel{?}{=} \varepsilon$, et puisque ce système d'équations n'admit qu'une seule solution alors il suffit de trouver une seule pour pouvoir corriger l'erreur.

Propriété 3
Soit n la longueur du code, et t la capacité de correction. La complexité algorithmique du décodage par syndrome correspond au nombre de vecteurs d'erreur possibles de poids inférieur ou égal à t :

$$\sum_{i=1}^{t} \binom{n}{i} \ .$$

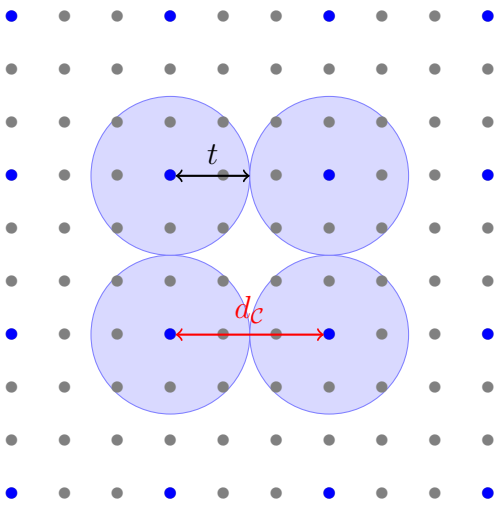
Preuve : Il suffit de remarquer que si l'erreur est de poids i, alors le nombre de vecteurs de poids i est $\binom{n}{i}$. Donc, le nombre de vecteurs d'erreurs possibles est : $\sum_{i=1}^{t} \binom{n}{i}$. ■

1.6.3 Les codes parfaits

Définition 12 (Code parfait)
Soit $\mathcal{C}$ un code de longueur n et de capacité de correction t. Le code $\mathcal{C}$ est dit *parfait* si et seulement si :

$$\mathbb{F}_2^n = \bigcup_{c\in\mathcal{C}} \left\{ x \in \mathbb{F}_2^n \quad | \quad d_H(c, x) \leqslant t \right\} \ .$$

En d'autres termes, les boules fermées et disjointes de rayon t et centrées en les mots de code couvrent tout l'espace vectoriel.



$d_{\mathcal{C}}$: la distance minimale du code
t : la capacité de correction
• • : les éléments de l'espace vectoriel
• : les éléments du code $\mathcal{C}$

FIGURE 1.11 – Illustration d'un code parfait

Théorème 1 (La borne de SINGLETON)
Tout code linéaire $\mathcal{C}$ de paramètres $[n, k, d_{\mathcal{C}}]$ vérifie l'inégalité suivante :

$$d_{\mathcal{C}} \leqslant n - k + 1 \ .$$

Preuve : Pour un code linéaire $[n, k, d_{\mathcal{C}}]$ sur un corps fini $\mathbb{F}$, tel que G sa matrice génératrice, et H sa matrice de parité. Le rang de H est égal à $n - k$. Donc il y a au plus $n - k$ colonnes dans H linéairement indépendantes. Il existe donc dans $\mathcal{C}$ des mots de poids $n - k + 1$. On obtient :

$$d_{\mathcal{C}} \leqslant n - k + 1.$$

De manière plus générale, considérons $\mathcal{C}$ un code qui n'est pas nécessairement linéaire, défini sur un alphabet K de cardinal q, de longueur n, de dimension k, et de cardinal $M = q^k$. Soient deux mots x et y obtenus en supprimant les $d_{\mathcal{C}} - 1$ dernières coordonnées de deux mots de code différents de $\mathcal{C}$. Les mots x et y ont au moins une coordonnée différente. On en déduit que le code obtenu à partir de $\mathcal{C}$

en éliminant les $d-1$ dernières coordonnées de chaque mot de code est de cardinal M, que $M \leqslant q^{n-d_{\mathcal{C}}+1}$ et donc que $k \leqslant n - d_{\mathcal{C}} + 1$. ■

Définition 13 (Code MDS)
Un code linéaire $\mathcal{C}$ vérifiant $d_{\mathcal{C}} = n - k + 1$ est dit MDS (*Maximum Distance Separable*), i.e. sa matrice de parité est de rang $d - 1$.

1.6.4 Les codes de Reed-Muller binaires

Les codes de Reed-Muller sont une famille de codes correcteurs d'erreurs linéaires. Cette famille de codes doit son nom aux travaux de Reed et Muller. En 1954, Muller [Mul54] a découvert le principe de ces codes, et indépendamment, Reed [Ree54] a proposé la méthode de décodage. Ces codes, comme nous allons le voir dans cette section, ont une capacité de détection et correction qui s'adapte selon le besoin (en assurant un compromis entre la capacité de correction et la cardinalité). Entre 1969 et 1973 les codes de Reed-Muller proposaient le meilleur compromis de l'époque, et ils ont été choisi par les satellites Mariner 9 et Viking du NASA [10] pour la transmission des images de Mars vers la Terre. On leur a ensuite préféré les codes de Reed-Solomon.

Définition 14 (Un monôme)
Soit $x = (x_1, \ldots, x_m) \in \mathbb{F}_2^m$ un vecteur binaire de longueur m. Un monôme est une fonction booléenne $\varphi_I : \mathbb{F}_2^m \to \mathbb{F}_2$ qui renvoie le produit de quelques coordonnées de x selon l'ensemble $I \subseteq \{1, \ldots, m\}$:

$$\varphi_I(x) = \prod_{i \in I} x_i \ ,$$

Exemple 3
Les monômes possibles pour le vecteur $x = (x_1, x_2, x_3)$ sont : 1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, $x_1x_2x_3$. ▮

Le degré d'un monôme correspond au nombre de coordonnées que nous multiplions. Par conséquent, seule la fonction constante 1 est de degré 0.
Il existe plusieurs façons de représenter une fonction booléenne (un mot de code). La plus classique c'est *la forme algébrique normale* (FAN) , qui consiste à représenter la fonction sous sa forme polynomiale (comme dans l'exemple 4). Une autre façon, et qui nous intéresse particulièrement dans ce manuscrit, est la forme vectorielle, elle consiste à représenter la fonction par sa table de vérité, donc un vecteur de longueur 2^m (pour un code binaire).

10. National Aeronautics and Space Administration

Nous remarquons que toute fonction booléenne est composée de la somme de plusieurs monômes, et que son degré correspond au degré maximal des monômes qui interviennent dans sa forme algébrique normale.

Exemple 4
Soit la fonction booléenne $f(x_1, x_2, x_3) = 1 + x_2 + x_1x_3$, cette fonction est composée du monôme 1 de degré 0, du monôme x_2 de degré 1, et du monôme x_1x_3 de degré 2, il s'agit donc d'une fonction de degré 2. ∎

Définition 15 (Les codes de Reed-Muller binaires)
Le code de Reed-Muller (RM) est l'ensemble des fonctions booléennes à m variable, de degrés au plus r, et on le note :

$$RM(r, m) = \{f : \mathbb{F}_2[x_1, \ldots, x_m] \to \mathbb{F}_2 \quad | \quad deg(f) \leqslant r\} \ .$$

Un code de Reed-Muller dépend donc des deux paramètres r et m.

Exemple 5
Soit $\mathcal{C} = RM(1, 3)$ le code de Reed-Muller d'ordre 1 à 3 variables. La liste des mots de code qui appartiennent à $\mathcal{C}$ est :

la FAN	la table de vérité
0	0 0 0 0 0 0 0 0
1	1 1 1 1 1 1 1 1
x_1	0 1 0 1 0 1 0 1
$x_1 + 1$	1 0 1 0 1 0 1 0
x_2	0 0 1 1 0 0 1 1
$x_2 + 1$	1 1 0 0 1 1 0 0
$x_2 + x_1$	0 1 1 0 0 1 1 0
$x_2 + x_1 + 1$	1 0 0 1 1 0 0 1
x_3	0 0 0 0 1 1 1 1
$x_3 + 1$	1 1 1 1 0 0 0 0
$x_3 + x_1$	0 1 0 1 1 0 1 0
$x_3 + x_1 + 1$	1 0 1 0 0 1 0 1
$x_3 + x_2$	0 0 1 1 1 1 0 0
$x_3 + x_2 + 1$	1 1 0 0 0 0 1 1
$x_3 + x_2 + x_1$	0 1 1 0 1 0 0 1
$x_3 + x_2 + x_1 + 1$	1 0 0 1 0 1 1 0

∎

Propriété 4 (La dimension d'un code RM)
La dimension du code $RM(r, m)$ est donnée par la formule :

$$k = \sum_{i=0}^{r} \binom{m}{i} \ .$$

Preuve : Le code $RM(r,m)$ est un sous-espace vectoriel de $\mathbb{F}_{2^{2^m}}$, et la famille des monômes de degré au plus r forme une base pour ce sous-espace vectoriel. Pour démontrer la formule précédente il suffit de remarquer que le nombre de monômes de degré i vaut $\binom{m}{i}$, ensuite compter le nombre de monômes de degré 0 jusqu'à r. ■

La matrice génératrice d'un code $RM(r,m)$ est construite à partir de la table de vérité des monômes qui appartiennent à ce code. Chaque ligne correspond à un monôme dont elle est la liste des valeurs. Pour l'exemple précédent il s'agit des monômes $\{1, x_1, x_2, x_3\}$, mais en général l'ensemble des monômes est défini par :

$$\left\{ \prod_{i=1}^{m} x_i^{v_i} \quad | \quad v \in \mathbb{F}_2^m \quad , \quad \omega_H(v) \leqslant r \right\} .$$

La matrice génératrice du code $RM(1,3)$ est donc définie par :

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{matrix}$$

Propriété 5
Pour tout $0 < r < m$ nous avons la propriété suivante :

$$RM(r,m) = \left\{ (u, u+v) \in \mathbb{F}_2^{2^m} \quad | \quad u \in RM(r, m-1), \quad v \in RM(r-1, m-1) \right\} .$$

Preuve : Soit $f \in RM(r,m)$, nous avons :

$$f(x_1, \ldots, x_m) = x_1 f_1(x_2, \ldots, x_m) + f_2(x_2, \ldots, x_m) ,$$

avec f_1 une fonction de degré au plus $r-1$ appartenant au code $RM(r-1, m-1)$, et f_2 une fonction de degré au plus r appartenant au code $RM(r, m-1)$. Nous avons :

$$f(x_1, \ldots, x_m) = \begin{cases} f_2(x_2, \ldots, x_m) . & \text{si } x_1 = 0 \\ \\ f_1(x_2, \ldots, x_m) + f_2(x_2, \ldots, x_m) . & \text{si } x_1 = 1 \end{cases}$$

Soient $u, v \in \mathbb{F}_2^{2^{m-1}}$ les tables de vérité de f_2 et f_1 respectivement. La table de vérité de f est donc définie par : $(\underbrace{u}_{x_1=0}, \underbrace{u+v}_{x_1=1})$. ■

Propriété 6 (La distance minimale d'un code RM)
La distance minimale d'un code de Reed-Muller de paramètres (r, m) est 2^{m-r}.

Preuve : On montre par récurrence sur m. Pour $m = 1$ nous avons :
- $RM(0,1) = \{(0,0),(1,1)\}$,
- $RM(1,1) = \{(0,0),(0,1),(1,0),(1,1)\}$.

Supposons que la distance minimale pour $RM(r, m-1)$ est $d_{m-1} = 2^{m-1-r}$. D'après la propriété 5 nous avons : $RM(r,m) = \{(u, u+v) \quad | \quad u \in RM(r, m-1), \quad v \in RM(r-1, m-1)\}$, donc $d_m = min\{\omega_H(u) + \omega_H(u+v)\}$ (avec ω_H désignant le poids de HAMMING) et puisque $RM(r-1, m-1) \subset RM(r, m-1)$ donc $u + v \in RM(r, m-1)$. D'après l'hypothèse de récurrence nous avons :
- Pour $u \neq v$: $\omega_H(u+v) \geqslant 2^{m-1-r}$ et $\omega_H(u) \geqslant 2^{m-1-r}$ donc $\omega_H(u, u+v) \geqslant 2^{m-1-r} + 2^{m-1-r} = 2^{m-r}$. Ainsi, sans perte de généralité, pour $v = 0$: $d_m = min\{\omega_H(u) + \omega_H(u)\} = 2^{m-r}$.
- Pour $u = v$: $\omega_H(u, u+v) = \omega_H(v, 0) = \omega_H(v)$, et puisque $v \in RM(r-1, m-1)$, alors $\omega_H(u, u+v) \geqslant 2^{m-r}$.

■

Propriété 7 (Le dual d'un code RM)
Le code dual d'un code de Reed-Muller de paramètres (r, m), avec $r < m$ est le code de Reed-Muller de paramètres $(m - r - 1, m)$:

$$RM(r,m)^{\perp} = RM(m-r-1, m) \ .$$

Preuve : Pour $m = 2$ nous avons :

$$\begin{array}{rcl} RM(0,2) & = & \{ \ (0\,0\,0\,0), (1\,1\,1\,1) \ \} \ . \\ RM(1,2) & = & \{ \ (0\,0\,0\,0), (0\,1\,0\,1), (1\,0\,1\,0), (1\,1\,1\,1), \\ & & \ (0\,0\,1\,1), (0\,1\,1\,0), (1\,0\,0\,1), (1\,1\,0\,0) \ \} \ . \end{array}$$

Les deux codes sont orthogonaux, pour le prouver il suffit de remarquer que tous les mots de $RM(1,2)$ sont de poids pairs.
Maintenant, supposons que la propriété 7 est vraie pour $m-1$, et montrons qu'elle est vraie aussi pour m, nous avons :

$$\begin{array}{ll} RM(r, m-1)^{\perp} & = RM(m-r-2, m-1) \ . \\ RM(r-1, m-1)^{\perp} & = RM(m-r-1, m-1) \ . \end{array}$$

Et d'après la propriété 5 nous avons :

$$RM(r,m) = \Big\{(u, u+v) \in \mathbb{F}_2^{2^m} \quad | \quad u \in RM(r,m-1), \; v \in RM(r-1,m-1)\Big\}.$$

$$RM(m-r-1,m) = \Big\{(u', u'+v') \in \mathbb{F}_2^{2^m} \quad | \quad u' \in RM(m-r-1,m-1), \; v' \in RM(m-r-2,m-1)\Big\}.$$

Donc $\langle v, u' \rangle = \langle u, v' \rangle = 0$, et puisque $RM(r-1,m-1) \subset RM(r,m-1)$, donc le produit scalaire $\langle v, v' \rangle$ vaut aussi zéro. Calculons maintenant le produit scalaire entre un élément de $RM(r,m)$ et un élément de $RM(m-r-1,m)$:

$$\begin{aligned} \langle (u, u+v), (u', u'+v') \rangle &= \langle u, u' \rangle + \langle (u+v), (u'+v') \rangle \\ &= \langle u, u' \rangle + \langle u, u' \rangle + \langle u, v' \rangle + \langle u', v \rangle + \langle v, v' \rangle \\ &= 0 \; . \end{aligned}$$

Ainsi, nous avons prouvé que : $RM(m-r-1,m) \subset RM(r,m)^{\perp}$, pour démontrer l'égalité, il suffit de montrer que les deux codes ont la même dimension (donc même cardinalité) :

$$\begin{aligned} dim(RM(r,m)^{\perp}) &= 2^m - dim(RM(r,m)) \\ &= 2^m - \sum_{i=0}^{r} \binom{m}{i} \\ &= \sum_{i=0}^{m} \binom{m}{i} - \sum_{i=0}^{r} \binom{m}{i} \\ &= \sum_{i=r+1}^{m} \binom{m}{i} \\ &= \sum_{i=0}^{m-r-1} \binom{m}{i} \\ &= dim(RM(m-r-1,m)) \; . \end{aligned}$$

■

1.7 L'AES

En 1997, un appel d'offre a été lancé par NIST [11] qui avait pour but d'élaborer un algorithme de chiffrement symétrique qui remplace le standard du chiffrement des données (DES [12]). Ainsi en 2000, le standard de chiffrement avancé connu sous le nom AES [13] [DR00] a été choisi parmi 15 candidats et devint ensuite le nouveau standard pour le chiffrement civil. L'AES repose sur les principes de *diffusion* et *confusion* énoncés par SHANNON [Sha49] :

11. National Institute of Standards and Technology
12. Data Encryption Standard
13. Advanced Encryption Standard

La diffusion : signifie que si nous changeons un ou plusieurs bits du texte en clair, en moyenne la moitié des bits du texte chiffré devrait changer, et de même, si l'on change un bit du cryptogramme, environ la moitié des bits en message clair devraient changer. En d'autres termes, la redondance statistique dans le texte en clair est dissipée dans les statistiques du texte chiffré, ainsi, aucune corrélation entre l'entrée et la sortie ne doit s'apercevoir.

La confusion : signifie que chaque bit du cryptogramme est calculé à partir de plusieurs bits de la clé dans l'objectif de rendre le lien entre la clé et le texte chiffré aussi complexe que possible.

L'AES est un réseau de substitution et de permutation, qui opère sur des blocs de taille 128, 196 ou 256 avec une clé de longueur 128, 196 ou 256 bits respectivement. Il faut 10 tours pour une clé de 128 bits, 12 pour une clé de 192 bits et 14 pour une clé de 256 bits.

	Longueur de la clé	Nombre de tours
AES-128	128	10
AES-192	192	12
AES-256	256	14

TABLE 1.2 – Le nombre de tours en fonction de la taille des clés.

Un tour comprend plusieurs étapes de traitement (des transformations) qui agissent sur une valeur de 128 bits (pour le cas AES-128), constituée de 16 octets arrangés en une matrice 4×4. Ces transformations sont basées sur des opérations sur le corps fini $K = \mathbb{F}_2[\alpha]/P(\alpha)$ tel que $P(\alpha) = \alpha^8+\alpha^4+\alpha^3+\alpha+1$. Chacune des classes d'équivalence de ce corps est représentée par un polynôme de degré au plus 7 à coefficients dans $\mathbb{F}_2$. Pour simplifier nous allons exprimer chaque polynôme de ce corps $Q(\alpha) = \sum_{i=0}^{7} c_i\alpha^i$ par deux chiffres en hexadécimal, tel que la représentation binaire de cet entier correspond aux coefficients du polynôme qu'il représente (i.e. $(c_7 \dots c_0)$).

Exemple 6

$$\begin{aligned} Q(\alpha) &= \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha \\ &= 0 \cdot \alpha^7 + 0 \cdot \alpha^6 + 1 \cdot \alpha^5 + 1 \cdot \alpha^4 + 1 \cdot \alpha^3 + 1 \cdot \alpha^2 + 1 \cdot \alpha + 0 \ . \end{aligned}$$

est représenté par l'entier 00111110 en binaire, et qui vaut $3E$ en hexadécimal. Donc le polynôme $Q(\alpha)$ sera représenté par l'entier $3E$. ∎

Pour chaque élément non nul x dans K, il existe un élément inverse que l'on notera x^{-1} tel que : $x \cdot x^{-1} = 1$. Nous allons également noter x_i le i^{eme} coefficient du polynôme x pour $0 \leqslant i \leqslant 7$.

1.7.1 Chiffrement

L'algorithme de chiffrement AES est composé des quatre transformations suivantes :

— `SubBytes` : aussi connue sous le nom "S-Box", c'est une fonction bijective non linéaire $S : K \rightarrow K$ qui assure la propriété de confusion. Elle permet de substituer indépendamment les octets d'un bloc grâce à une table de substitution prédéfinie, cette table est calculée grâce à la fonction suivante :

$$S(x)_i = x_i^{-1} \oplus x_{(i+4) \bmod 8}^{-1} \oplus x_{(i+5) \bmod 8}^{-1} \oplus x_{(i+6) \bmod 8}^{-1} \oplus x_{(i+7) \bmod 8}^{-1} \oplus c_i \ , \quad (1.7)$$

pour $0 \leqslant i < 8$, avec c_i désigne le i^{eme} coefficient de l'élément $c = 63$, et l'opération $\oplus$ qui désigne le XOR.

— `ShiftRows` : cette transformation opère indépendamment sur les lignes de la matrice en entrée $M \in K^{4\times4}$. Elle opère une permutation circulaire sur les octets dans chaque ligne. Pour l'AES-128, la première ligne est laissée inchangée. Chaque octet de la deuxième ligne est décalé d'un pas vers la gauche. De même, la troisième et quatrième ligne sont décalées par des décalages de deux et trois respectivement. Si on considère $M \in K^{4\times4}$ la matrice en entrée alors :

$$\texttt{ShiftRows}(M_{i,j}) = M_{i,(i+j) \bmod 4} \mid \forall\ 0 \leqslant i, j \leqslant 3 \ .$$

— `MixColumns` : Dans cette étape, les quatre octets de chaque colonne de la matrice en entrée sont combinés à l'aide d'une transformation linéaire inversible. Cette transformation consiste en un produit matriciel entre une matrice prédéfinie notée $L \in K^{4\times4}$ et la matrice en entrée M :

$$\texttt{MixColumns}(M) = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot M \ .$$

Avec `ShiftRows`, `MixColumns` fournit la propriété de diffusion dans le chiffrement.

— `AddRoundKey` : Dans cette étape, la sous-clé est combinée avec la matrice en entrée. Pour chaque tour, une sous-clé dérivée de la clé principale à l'aide du générateur de clés de Rijndael (qui lui aussi utilise les transformations précédentes) ; Chaque sous-clé est de la même taille que la matrice M. La sous-clé est ajoutée en combinant chaque octet de la matrice avec l'octet correspondant de la sous-clé à l'aide d'une addition dans K (notée $\bigoplus$).

Durant l'opération de chiffrement, ces transformations sont exécutées dans l'ordre indiqué dans la figure 1.12.

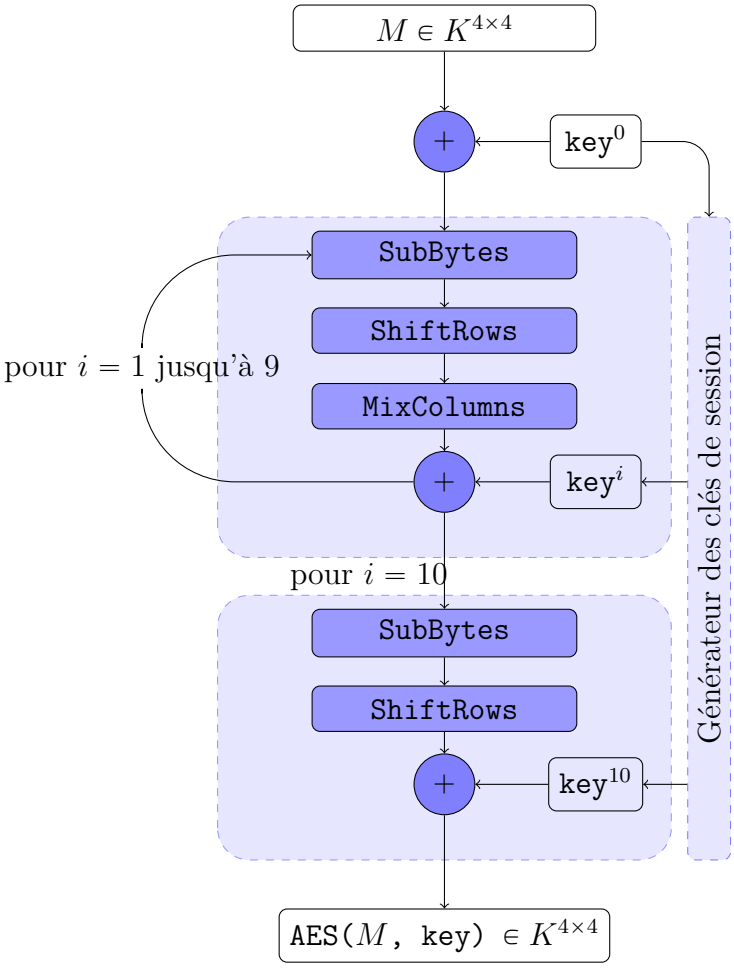


FIGURE 1.12 – L'opération de chiffrement pour l'AES-128

1.7.2 Déchiffrement

Pour déchiffrer, il suffit d'appliquer l'opération inverse de chacune des transformations que nous avons vues précédemment :

— `InvSubBytes` : L'opération inverse de `SubBytes` peut être calculée grâce à la fonction suivante $S' : x \mapsto y^{-1}$ tel que :

$$y_i = x_{(i+2) \bmod 8} \oplus x_{(i+5) \bmod 8} \oplus x_{(i+7) \bmod 8} \oplus c_i \ , \tag{1.8}$$

pour $0 \leqslant i < 8$.

— `InvShiftRows` :

$$\texttt{InvShiftRows}(M_{i,j}) = M_{i,(i-j) \bmod 4} \mid \forall\ 0 \leqslant i, j \leqslant 3 \ ;$$

— `InvMixColumns` : Pour inverser la transformation `MixColumns`, il suffit de multiplier la matrice en entrée par la matrice inverse de L qui est égale à :

$$L^{-1} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}.$$

— `AddRoundKey` : L'addition et la soustraction sont identiques dans un corps de caractéristique 2.

Première partie

LES TECHNIQUES DE MASQUAGE POUR LUTTER CONTRE LES ATTAQUES PAR CANAUX AUXILIAIRES (SCA) ET LES ATTAQUES PAR INJECTION DE FAUTES (FIA)

1 | L'état de l'art sur le masquage

Sommaire

Juste après la naissance de l'AES au début de ce siècle, des travaux ont montré que les attaques par canaux auxiliaires constituent une menace pour les cryptosystèmes implémentés dans des dispositifs embarqués. La consommation électrique pendant une opération de chiffrement ou le temps de calcul peuvent divulguer des informations sur des valeurs intermédiaires manipulées au cours de l'algorithme de chiffrement, ceci entraîne des attaques pratiques sur les composants électriques [Koc96, KJJ99]. Les rayonnements électromagnétiques [QS01, GMO01, RR01, QS02], et même les sons émis par les machines à rotor [Kah74] peuvent également donner à l'attaquant une fenêtre de visibilité sur les valeurs intermédiaires au cours du calcul. Ce type d'attaques fourni à l'adversaire une version bruitée de valeurs telles que le poids de HAMMING de certaines données sensibles.

Par ailleurs, les attaques par injection de fautes consistent à agir sur les conditions environnementales du système (tension, horloge, température, rayonnement, lumière, ...) pour générer des défauts pendant l'exécution du chiffrement et observer ensuite le comportement résultant. De telles attaques peuvent être conçues en éclairant simplement un transistor avec un faisceau laser (l'effet photovoltaïque) [SA02, AK97] ; ce qui génère un dysfonctionnement lié à la modification de la fréquence du travail pendant le calcul, et qui entraîne une mauvaise valeur de certains bits. L'utilisation d'une erreur lors d'un calcul pour deviner la clé secrète a été observée pratiquement dans les implémentations de RSA qui utilisent le théorème chinois des restes [JLQ99, BDL01, AK97]. En ce qui concerne le chiffrement symétrique et en particulier l'AES ; une attaque simple et directe a été proposée par BLÖMER [BS03] qui vise à changer un seul bit juste après la première addition de la clé (`AddRoundKey`). L'objectif est de réinitialiser un seul bit dans l'état interne M^0 (en général, M^i indique l'état au début du i-ème tour), et observer si la valeur du texte chiffré a changé. Si le texte chiffré a changé ou a été détecté comme défectueux par un circuit de détection de défaut, l'attaquant saura que la valeur correcte du bit est 1, sinon elle est égale à 0. Étant donné que le bit modifié est le résultat d'un XOR entre le texte en clair supposé connu et la clé secrète, l'attaquant pourra donc deviner facilement la valeur du bit concerné de la clé. Bien que cette attaque puisse en principe récupérer la clé de chiffrement en entier, elle a été jugée pratiquement impossible en raison du temps précis requis pour l'injection de fautes et de la précision stricte sur la position exacte de l'injection.

En raison de la très grande variété d'attaques SCA (*Side Channel Attack*) et FIA (*Fault Injection Attack*) signalées contre les systèmes et les dispositifs cryptographiques, des efforts importants ont été déployés pour concevoir des contre-mesures avec une sécurité prouvable. Ces contre-mesures partent toutes de l'hypothèse qu'un dispositif cryptographique dispose d'une partie sécurisée de mémoire et que seuls les calculs peuvent fuiter. Les contre-mesures les plus courantes pour lutter contre de telles attaques sont *le masquage* [GP99, CJRR99] et *le brouillage*

[RPD09]. *Le brouillage* est une solution simple qui consiste à ajouter une série aléatoire d'opérations pendant le (dé)chiffrement afin de camoufler les fuites émises par le composant pendant l'exécution des différentes opérations de l'algorithme de (dé)chiffrement. Tandis que *le masquage* consiste à exprimer chaque variable intermédiaire sensible apparaissant dans le calcul en plusieurs parties indépendantes. Il est indispensable que ces parties ne puissent révéler aucune information sur la donnée sensible s'il en manque au moins une.
Si nous considérons $f : E \to E$ une des transformations qui composent un cryptosystème symétrique quelconque (AES-128 par exemple), le masquage de cette transformation consiste à concevoir une fonction indéterministe (une variable aléatoire) $\texttt{mask} : E \to F$, ainsi qu'une transformation alternative $f' : F \to F$ telles que :

$$f'(\texttt{mask}(x)) = \texttt{mask}(f(x)) \ ,$$

où x représente la donnée sensible à protéger, et $\texttt{mask}(x)$ le masqué associé à x. Il faut noter également que la transformation alternative f' ne doit révéler aucune information sur la donnée sensible x durant son exécution, si c'est le cas, on dit alors que f' est *Probing secure* (voir la définition 16 page 43). D'un autre côté, pour réussir à détecter et éventuellement corriger les erreurs potentielles susceptibles d'être injectées, il faudra que l'espace d'arrivée F de la fonction de masquage soit un code correcteur d'erreurs prédéfini et qui offre la possibilité de détecter ou corriger les injections d'erreurs. Pour cela, nous allons voir dans le chapitre 2 qu'il est possible de concevoir un masquage détecteur d'erreurs.

1.1 Le masquage booléen

Plusieurs solutions ont été proposées pour protéger le chiffrement symétrique des attaques par canaux auxiliaires. La méthode de masquage la plus classique, appelée masquage booléen, consiste à exprimer la donnée sensible x comme somme de plusieurs parties $x_0, x_1 \ldots, x_d$ (ce qu'on appelle en anglais "*shares*") telles que : $x = \bigoplus_{i=0}^{d} x_i$ et pour tout ensemble $S \subsetneq \{x_0, \ldots, x_d\}$ [1], les éléments de S ne révèlent aucune information sur x. Le masquage consiste ensuite à appliquer les opérations qui composent le système de chiffrement à chacune de ces parties sans impliquer directement la donnée sensible x dans le processus de calcul.
Ce type de décomposition est $\mathbb{F}_2$-linéaire, par conséquent, le masquage des transformations additives est simple à réaliser. En effet, si on considère $(x_0, \ldots, x_d)$ et $(y_0, \ldots, y_d)$ les *shares* associés aux données sensibles x et y respectivement (i.e. $x = \bigoplus_{i=0}^{d} x_i, \quad y = \bigoplus_{i=0}^{d} y_i$) alors, on peut considérer $((x_0 \oplus y_0), \ldots, (x_d \oplus y_d))$ les *shares* associés à $x \oplus y$ (car : $x + y = \bigoplus_{i=0}^{d}(x_i \oplus y_i)$).
D'un autre côté, il est moins intuitif de masquer les transformations non additives (non-linéaires). En effet, nous allons voir dans les schémas qui suivent que le masquage des opérations telles que la multiplication demande - dans le meilleur des cas - une complexité quadratique.
Chaque transformation f qui compose l'algorithme de chiffrement (ou déchiffrement) doit être remplacée par une fonction f' telle que $f'(x_0, \ldots, x_d) = (y_0, \ldots, y_d)$ et que $\bigoplus_{i=0}^{d} y_i = f(x)$. Dans ce cas de figure, l'adversaire aura besoin d'au moins $d + 1$ variables intermédiaires pour construire la donnée sensible x, c'est ce qu'on appelle un masquage d'ordre d. Cependant, un masquage d'ordre d est toujours théoriquement vulnérable à une SCA d'ordre $(d + 1)$ qui exploite les fuites liées à $d + 1$ variables intermédiaires en même temps [Mes00, RPD09, SP06]. Dans la pratique les effets de bruit impliquent que la complexité d'une attaque d'ordre d augmente exponentiellement avec d.

1.1.1 Le schéma Ishai-Sahai-Wagner (ISW)

Une première solution pour masquer les fonctions non linéaires a été proposée par Ishai *et al.* dans [ISW03]. Le schéma présenté permet de sécuriser l'implémentation matérielle de tout circuit à n'importe quel ordre choisi d. Ils décrivent un moyen de transformer le circuit pour se protéger en un nouveau circuit (traitant des valeurs masquées) de sorte qu'aucun sous-ensemble de d de ses fils ne révèle des informations sur les valeurs sensibles. Pour ce faire, ils supposent - sans perte de généralité - que le circuit à protéger est exclusivement composé de portes logiques "NON" (que nous allons noter $\neg$) et "ET" (que nous allons noter $\wedge$). Dans ce contexte, la

1. Le cardinal de l'ensemble S est strictement inférieur au nombre de parties.

protection de la porte logique de type "NON" revient à protéger l'addition binaire (XOR) (car $\neg x = x \oplus 1$), ce qui a l'avantage de se généraliser comme l'ont montré Rivain et Prouff (voir ci-dessous) aux mêmes opérations dans $\mathbb{F}_{2^n}$, et qui s'applique directement à l'AES. La sécurisation d'une porte "NOT" pour n'importe quel ordre d est simple puisque $x = \bigoplus_{i=0}^{d} x_i$ implique $\neg x = \neg x_0 \oplus \bigoplus_{i=1}^{d} x_i$. La principale difficulté est donc de sécuriser les portes "ET".

Soient $(x_0, \ldots, x_d), (x'_0, \ldots, x'_d) \in \mathbb{F}_2^d$, les *shares* associés à x et x' respectivement (i.e. $x = \bigoplus_{i=0}^{d} x_i \in \mathbb{F}_2$ et $x' = \bigoplus_{i=0}^{d} x'_i \in \mathbb{F}_2$). Pour sécuriser la porte ET, il faut construire une fonction $(x_0, \ldots, x_d), (x'_0, \ldots, x'_d) \mapsto (y_0, \ldots, y_d)$ telle que : $\bigoplus_{i=0}^{d} y_i = (\bigoplus_{i=0}^{d} x_i)(\bigoplus_{i=0}^{d} x'_i)$. Pour répondre à cette question, Ishai *et al.* ont suggéré la solution suivante :

- Pour tout $0 \leqslant i < j \leqslant d$, choisir un bit aléatoire $r_{i,j}$ (le masque),
- Pour tout $0 \leqslant i < j \leqslant d$, calculer $r_{j,i} = (r_{i,j} \oplus x_i x'_j) \oplus x_j x'_i$,
- Pour tout $0 \leqslant i \leqslant d$ calculer $y_i = x_i x'_i \oplus \bigoplus_{j=0, j\neq i}^{d} r_{i,j}$.

L'utilisation du masque $r_{i,j}$ est essentielle dans ce schéma, c'est lui qui permet de protéger les opérations effectuées grâce à cette couche aléatoire. Il faut noter également que l'utilisation des parenthèses pour calculer $r_{j,i}$ indique l'ordre dans lequel les opérations sont effectuées, ce qui est primordial pour la *probing security*. En effet, en respectant cet ordre, le masque $r_{i,j}$ interviendra dans toutes les opération "$\oplus$" afin de masquer les valeurs sensibles.

Définition 16 (*Probing Attack)*
L'attaque PA (en anglais : *Probing Attack*) fait partie de la famille SCA, elle consiste à placer une aiguille métallique sur un des fils d'une carte électronique et lire la valeur transportée par ce fil pendant le calcul.

Définition 17 (t-Probing Security)
On dit qu'un algorithme atteint un niveau de sécurité *t-PS* (*t-Probing security*) si chaque ensemble de t variables intermédiaires (au plus) de l'exécution ne laisse fuir aucune information sur les variables sensibles.

Exemple 7
Pour $d = 2$ on obtient :

$i \setminus j$	r			y
	0	1	2	
0	$-$	$r_{0,1}$	$r_{0,2}$	$x_0 x'_0 \oplus r_{0,1} \oplus r_{0,2}$
1	$r_{0,1} \oplus x_0 x'_1 \oplus x_1 x'_0$	$-$	$r_{1,2}$	$x_1 x'_1 \oplus r_{1,0} \oplus r_{1,2}$
2	$r_{0,2} \oplus x_0 x'_2 \oplus x_2 x'_0$	$r_{1,2} \oplus x_1 x'_2 \oplus x_2 x'_1$		$x_2 x'_2 \oplus r_{2,0} \oplus r_{2,1}$

On peut donc vérifier que :

$$
\begin{aligned}
y_0 &= x_0x'_0 \oplus \underbrace{r_{0,1} \oplus r_{0,2}}_{r'_0} \\
y_1 &= x_1x'_1 \oplus x_1x'_0 \oplus x_0x'_1 \oplus \underbrace{r_{0,1} \oplus r_{1,2}}_{r'_1} \\
y_2 &= x_2x'_2 \oplus x_2x'_0 \oplus x_0x'_2 \oplus x_2x'_1 \oplus x_1x'_2 \oplus \underbrace{r_{0,2} \oplus r_{1,2}}_{r'_2} \\
xx' &= (x_0 \oplus x_1 \oplus x_2)(x'_0 \oplus x'_1 \oplus x'_2) \\
&= \underbrace{x_0x'_0}_{y_0 - r'_0} \oplus \underbrace{x_1x'_1 \oplus x_1x'_0 \oplus x_0x'_1}_{y_1 - r'_1} \oplus \underbrace{x_2x'_2 \oplus x_2x'_0 \oplus x_0x'_2 \oplus x_2x'_1 \oplus x_1x'_2}_{y_2 - r'_2} \\
&= y_0 \oplus y_1 \oplus y_2 \ .
\end{aligned}
$$

Comme nous l'avons mentionné précédemment, l'ordre d'exécution des opérations est très important, l'idée consiste à faire intervenir le masque $r_{i,j}$ (avec $i < j$) dans chacune des additions ($\oplus$). Comme nous le constatons dans cet exemple les shares obtenus : y_0, y_1 et y_2 contiennent tous une trace de ce masque (qui est respectivement r'_0, r'_1 et r'_2).
Pour vérifier que ce masquage est 2-PS, il suffit de calculer toutes les combinaisons possibles de a et b dans $\{r_{0,1}, r_{0,2}, r_{1,0}, r_{1,2}, r_{2,0}, r_{2,1}, y_0, y_1, y_2\}$ et constater qu'aucune combinaison de révèle des informations sur les données sensibles. D'un autre côté, si nous combinons 3 variables, par exemple : $y_0 \oplus y_1 \oplus y_2$ nous pouvons obtenir l'information xx'. ∎

Ce schéma permet de masquer les deux opérations logiques "ET" et "NON", ces deux opérations représentent la base de toutes les transformations qui composent un cryptosystème. Cependant, ce schéma induit un surcoût important pour le circuit masqué. En effet, toutes les portes logiques de type "ET" sont encodées en utilisant $(d+1)^2$ portes de type "ET" et $2d(d+1)$ portes de type "XOR", et nécessite la génération de $d(d+1)/2$ bits aléatoires à chaque cycle d'horloge.
Pour réduire le problème de surcoût, Rivain et Prouff [RP10] ont présenté une solution similaire avec des améliorations remarquables au niveau de la complexité des calculs.

1.1.2 Le schéma de Rivain et Prouff (RP)

Rivain et Prouff [RP10] ont présenté une version améliorée du schéma précédent. L'idée consiste à adapter ce qui était fait pour des opérations sur $\mathbb{F}_2$ à des opérations sur $\mathbb{F}_{2^n}$, permettant ainsi d'opérer directement sur des données de taille plus grande.

Soit $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[\alpha]/P(\alpha)$ le corps fini utilisé pour AES (voir la section 1.7 page 32), soit $a \in \mathbb{F}_{2^8}$ une donnée sensible, et $a_0, \ldots, a_d$ les *shares* associés à a définis par : $a = \bigoplus_{i=0}^{d} a_i$, pour simplifier nous allons apposer une flèche en haut de la variable pour designer qu'il s'agit du mot masqué : $\vec{a} = (a_0, \ldots, a_d) \in \mathbb{F}_{2^8}^{d+1}$.
L'algorithme 1 permet de masquer un octet selon la méthode de Rivain et Prouff, l'algorithme opère sur une donnée de taille 8 bits (un octet), et renvoie un vecteur de taille $d + 1$ octets.

Algorithm 1 `RPMaskByte` — Complexité : d additions

1: **entrée :** une donnée sensible $a \in \mathbb{F}_{2^8}$ et l'ordre du masquage d
2: **sortie :** $\vec{a} = (a_0, \ldots, a_d) \in (\mathbb{F}_{2^8})^{d+1}$ tel que $\bigoplus_{i=0}^{d} a_i = a$
3: $a_0 \leftarrow a$
4: **pour** $i = 1$ **jusqu'à** d **faire :**
5: $\quad a_i \xleftarrow{\$} \mathbb{F}_{2^8}$
6: $\quad a_0 \leftarrow a_0 \oplus a_i$
7: **renvoyer** $\vec{a}$

L'algorithme 2 représente l'opération inverse de l'algorithme précédent, il permet d'extraire la donnée sensible à partir d'une donnée masquée.

Algorithm 2 `RPUnmaskByte` — Complexité : $(d + 1)$ additions

1: **entrée :** un masqué $\vec{a} = (a_0, \ldots, a_d) \in (\mathbb{F}_{2^8})^{d+1}$
2: **sortie :** la donnée sensible $a \in \mathbb{F}_{2^8}$ tel que $\bigoplus_{i=0}^{d} a_i = a$
3: $a \leftarrow 0$
4: **pour** $i = 0$ **jusqu'à** d **faire :**
5: $\quad a \leftarrow a \oplus a_i$
6: **renvoyer** a

Dans le schéma de Rivain et Prouff, une donnée sensible $a \in \mathbb{F}_{2^8}$ dépend des $d+1$ shares $a_0, \ldots, a_d$, il s'agit donc d'un masquage d'ordre d et qui protège contre toute attaque d'ordre inférieur à $d + 1$. Pour appliquer ce schéma sur l'AES il faudra reconstruire toutes les transformations `(Inv)SubBytes`, `(Inv)MixColumns` et `AddRoundKey` qui composent l'AES (la transformation `(Inv)ShiftRows` n'opère pas sur les données, il s'agit juste d'une permutation des octets, il est donc inutile de la masquer). Comme nous l'avons expliqué dans la section 1.7, ces transformations se composent essentiellement de trois types d'opérations dans $\mathbb{F}_{2^8}$: l'addition, la multiplication et l'inverse.

1.1.2.1 La linéarité du masquage

Définition 18 (La linéarité)
Soient E et F deux espaces vectoriels sur un corps K, on dit que $f : E \to F$ est une fonction linéaire si pour tous $x, y \in E$ et $\lambda, \mu \in K$ nous avons :

$$f(\lambda x + \mu y) = \lambda f(x) + \mu f(y) \ .$$

Propriété 8
La variable aléatoire $\texttt{RPMaskByte} : \mathbb{F}_8 \to \mathbb{F}_8^{d+1}$ satisfaisant : $\texttt{RPMaskByte}(x) = (x_0, \ldots, x_d)$ tels que $\sum_{i=0}^{d} x_i = x$ est linéaire.

Preuve :
Soient $a, b \in \mathbb{F}_{2^8}$ deux valeurs sensibles, et $\vec{a}, \vec{b} \in \mathbb{F}_{2^8}^{d+1}$ deux vecteurs tels que : $\vec{a} = \texttt{RPMaskByte}(a)$ et $\vec{b} = \texttt{RPMaskByte}(b)$. Nous avons :

$$\begin{aligned} \texttt{RPMaskByte}(a) \oplus \texttt{RPMaskByte}(b) &= \vec{a} \oplus \vec{b} \\ &= (a_0, \ldots, a_d) \oplus (b_0, \ldots, b_d) \\ &= (a_0 \oplus b_0, \ldots, a_d \oplus b_d) \\ &= \texttt{RPMaskByte}(a \oplus b) \ . \end{aligned}$$

Donc la fonction RPMaskByte est bien linéaire. ■
En général la linéarité d'une fonction de masquage est une propriété importante. Elle permet de masquer plus facilement les opérations linéaires dans le corps fini utilisé. Les opérations comme l'addition, la multiplication par un scalaire, et l'exponentiation avec un exposant sous la forme 2^n sont toutes des opérations linéaires faciles à masquer.
Comme nous l'avons montré dans la preuve précédente, pour masquer l'addition il suffit d'additionner les *shares* des valeurs en entrée respectivement.

Algorithm 3 RPAdd — Complexité : $(d+1)$ additions

1: **entrée :** $\vec{a} = \texttt{RPMaskByte}(a)$ et $\vec{b} = \texttt{RPMaskByte}(b)$
2: **sortie :** $\vec{c} = \texttt{RPMaskByte}(a \oplus b)$
3: **pour** $i = 0$ **jusqu'à** d **faire :**
4: $\quad c_i \leftarrow a_i \oplus b_i$
5: **renvoyer** $\vec{c}$

1.1.2.2 La multiplication par un scalaire

Un cas particulier de multiplication qui sera intéressant pour plus tard, c'est la multiplication entre un mot masqué et une valeur non masquée. Souvent, dans un chiffrement symétrique comme l'AES, les données sensibles sont multipliées

par des valeurs publiques. Par exemple : La transformation `MixColumns` (voir la section 1.7 page 32) consiste à multiplier la matrice intermédiaire par une matrice prédéfinie, il est donc inutile de masquer les données publiques.
La multiplication par un scalaire est une opération linéaire, ceci simplifie son masquage. Soient $a, \lambda \in \mathbb{F}_{2^8}$ tels que a une donnée sensible associé à $\vec{a} \in \mathbb{F}_{2^8}^{(d+1)}$ et λ une donnée publique non masquée. Nous avons $\lambda\vec{a} = \lambda a_0 \oplus \cdots \oplus \lambda a_d$ donc nous pouvons considérer $\vec{c} = (\lambda a_0, \ldots, \lambda a_d)$ le masqué associé à $\lambda\vec{a}$.

Algorithm 4 `RPMult2` Complexité : $(d+1)$ multiplications

1: **entrée :** $\vec{a} = \texttt{RPMaskByte}(a)$ et λ
2: **sortie :** $\vec{c} = \texttt{RPMaskByte}(\lambda a)$
3: **pour** $i = 0$ **jusqu'à** d **faire :**
4: $\quad c_i = \lambda a_i$
5: **renvoyer** $\vec{c}$

1.1.2.3 La multiplication entre deux données masquées

La multiplication entre deux vecteurs n'est pas linéaire, ce qui mène à des calculs supplémentaires pour garder la même structure du masque. L'algorithme de Rivain et Prouff est identique à celui de l'opération "ET" du schéma ISW de la section précédente. En revanche, cette fois on traite des éléments du corps fini $\mathbb{F}_{2^8}$. Comme pour la multiplication ISW, il est important de bien respecter les parenthèses dans la ligne 6 de l'algorithme pour atteindre la *probing security*.

Algorithm 5 `RPMult` Complexité : $(d+1)^2$ multiplications et $2d(d+1)$ additions

1: **entrée :** $\vec{a} = \texttt{RPMaskByte}(a)$ et $\vec{b} = \texttt{RPMaskByte}(b)$
2: **sortie :** $\vec{c} = \texttt{RPMaskByte}(ab)$
3: **pour** $i = 0$ **jusqu'à** d **faire :**
4: $\quad$ **pour** $j = i + 1$ **jusqu'à** d **faire :**
5: $\quad\quad r_{i,j} \xleftarrow{\$} \mathbb{F}_{2^8}$
6: $\quad\quad r_{j,i} \leftarrow (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$
7: **pour** $i = 0$ **jusqu'à** d **faire :**
8: $\quad c_i \leftarrow a_i b_i$
9: $\quad$ **pour** $j = 0$ **jusqu'à** d **faire :**
10: $\quad\quad$ **si** $j \neq i$ **alors :**
11: $\quad\quad\quad c_i \leftarrow c_i \oplus r_{i,j}$
12: **renvoyer** $\vec{c}$

Étant donné que la complexité de cet algorithme est quadratique (donc coûteuse),

les auteurs ont proposé de distinguer le cas particulier ou $a = b$. En effet, l'exponentiation avec un exposant de la forme q^n dans un corps $\mathbb{F}_q$ est toujours linéaire.

1.1.2.4 L'exponentiation

Les algorithmes précédents (`RPAdd`, `RPMult` et `RPMult2`) sont suffisants pour masquer les transformations `AddRoundKey` et `(Inv)MixColumns`. Cependant, la transformation `(Inv)SubBytes` (les équations 1.7 et 1.8 pages : 34, 35) est composée d'un autre type d'opérations qui est l'inverse ($x \to x^{-1}$).
Soit : $K = \mathbb{F}_2[\alpha]/P(\alpha)$, le corps fini utilisé dans l'AES, nous avons :

$$\forall x \in K, \quad x^{-1} = x^{254} \ .$$

Propriété 9
Soient $\mathbb{F}_{p^k}$ un corps fini de caractéristique p premier, et n un entier naturel. La fonction $f(x) = x^{p^n}$ est linéaire dans $\mathbb{F}_{p^k}$. Par conséquent :

$$\forall d \in \mathbb{N}^*, \ \forall a_0, \ldots, a_d \in \mathbb{F}_{p^k}, \ (a_0 + \cdots + a_d)^{p^n} = a_0^{p^n} + \cdots + a_d^{p^n} \ .$$

Nous pouvons distinguer deux types d'exponentiation. D'abord l'exponentiation avec un exposant de la forme p^n (dans notre cas $p = 2$ et $n = 8$), ce type de puissance est linéaire dans $\mathbb{F}_{p^n}$ d'après la propriété 9, elle est donc plus facile à calculer. Pour ce faire, il suffit d'appliquer l'exponentiation à chacune des parts indépendamment.

Algorithm 6 `RPExp2` — Complexité : $n \cdot d$ multiplications

1: **entrée :** $\vec{a} =$ `RPMaskByte`(a) et un entier de la forme 2^n
2: **sortie :** $\vec{c} =$ `RPMaskByte`(a^{2^n})
3: **pour** $i = 0$ **jusqu'à** d **faire :**
4: $\quad c_i = a_i^{2^n}$
5: **renvoyer** $\vec{c}$

L'autre type d'exponentiation, ce sont toutes les autres formes d'exposant qui ne sont pas des puissances de 2. Pour nous le but est de calculer a^{254} en utilisant les deux algorithmes `RPMult` et `RPExp2`. On remarque que ce dernier algorithme est beaucoup moins gourmand en termes de complexité que le premier. Rivain et Prouff ont donc présenté un algorithme d'exponentiation pour l'exposant 254 qui utilise le moins de multiplications possible et plus de carrés.

Algorithm 7 `RPInverse` Complexité : $4d^2 + 15d + 4$ multiplications $8d^2 + 12d$ additions

1:	**entrée :** $\vec{a} = \texttt{RPMaskByte}(a)$	
2:	**sortie :** $\vec{c} = \texttt{RPMaskByte}(a^{-1})$	
3:	$\vec{z} \leftarrow \texttt{RPExp2}(\vec{a}, 2)$	$z \leftarrow a^2$
4:	$\texttt{RPRefresh}(\vec{z})$	
5:	$\vec{c} \leftarrow \texttt{RPMult}(\vec{z}, \vec{a})$	$c \leftarrow a^3$
6:	$\vec{w} \leftarrow \texttt{RPExp2}(\vec{c}, 4)$	$w \leftarrow a^{12}$
7:	$\texttt{RPRefresh}(\vec{w})$	
8:	$\vec{c} \leftarrow \texttt{RPMult}(\vec{c}, \vec{w})$	$c \leftarrow a^{15}$
9:	$\vec{c} \leftarrow \texttt{RPExp2}(\vec{c}, 16)$	$c \leftarrow a^{240}$
10:	$\vec{c} \leftarrow \texttt{RPMult}(\vec{c}, \vec{w})$	$c \leftarrow a^{252}$
11:	$\vec{c} \leftarrow \texttt{RPMult}(\vec{c}, \vec{z})$	$c \leftarrow a^{254}$
12:	**renvoyer** $\vec{c}$	

Pour bien assurer le niveau de sécurité souhaité il est important que les entrées de la fonction `RPMult` soient indépendantes, d'où le besoin d'ajouter une fonction de rafraîchissement. En effet, la non-utilisation de la fonction `RPRefresh` de la ligne 4 par exemple impactera le calcul de $\vec{c}$ de la ligne 5, nous remarquons qu'à cette ligne l'algorithme multiplie $\vec{a} = (a_0, a_1, \ldots, a_d)$ par $\vec{z} = (a_0^2, a_1^2, \ldots, a_d^2)$. Donc pour camoufler cette dépendance entre $\vec{a}$ et $\vec{z}$, il est primordial d'insérer une fonction de rafraîchissement. Pour ce faire, l'algorithme `RPRefresh` permet de générer des masques aléatoires $r_1, \ldots, r_d$, et les appliquer à la donnée masquée.
Soit $a = a_0 \oplus \bigoplus_{i=1}^{d} a_i$, nous avons : $a = a_0 \oplus (r_1 \oplus \cdots \oplus r_d) \oplus \bigoplus_{i=1}^{d} a_i \oplus r_i$, nous obtenons ainsi un autre mot masqué $\vec{a} = (a_0 \oplus (r_1 \oplus \cdots \oplus r_d), a_1 \oplus r_1, \ldots, a_d \oplus r_d)$.

Algorithm 8 `RPRefresh` Complexité : $2d$ additions

1:	**entrée :** $\vec{a} = \texttt{RPMaskByte}(a)$
2:	**sortie :** $\vec{c} = \texttt{RPMaskByte}(a)$
3:	$c_0 \leftarrow x_0$
4:	**pour** $i = 1$ **jusqu'à** d **faire :**
5:	$\quad r \xleftarrow{\$} \mathbb{F}_{2^8}$
6:	$\quad c_0 \leftarrow c_0 \oplus r$
7:	$\quad c_i \leftarrow x_i \oplus r$
8:	**renvoyer** $\vec{c}$

1.1.2.5 Le masquage de l'AES

Comme nous l'avons présenté dans la section 1.7, l'AES est composé de quatre transformations qui s'exécutent dans un ordre spécifique et de manière itérative

(voir 1.12). Les deux transformations `MixColumns` et `AddRoundKey` représentent la partie linéaire du programme, et c'est la partie la plus facile à masquer. Tandis que `SubBytes` représente la partie non-linéaire qui par conséquent est plus difficile à masquer.
L'AES opère sur une matrice d'octets $(M_{i,j})_{1\leqslant i,j\leqslant 4}$ dans $\mathbb{F}_{2^8}$. Notons : $(\vec{M}_{i,j})_{1\leqslant i,j\leqslant 4}$ la matrice masquée associée à un message M pour le tour en cours, et $(\vec{K}_{i,j})_{1\leqslant i,j\leqslant 4}$ la matrice masquée de la clé K pour le même tour, i.e. $\vec{M}_{i,j} = (M_{i,j,0}, \ldots, M_{i,j,d})$ tel que : $M_{i,j} = \bigoplus_{z=0}^{d} M_{i,j,z}$ et $\vec{K}_{i,j} = (K_{i,j,0}, \ldots, K_{i,j,d})$ telle que : $K_{i,j} = \bigoplus_{z=0}^{d} K_{i,j,z}$. Nous rappelons que l'objectif dans cette construction est de concevoir pour chaque transformation f de l'AES une transformation alternative f' telle que :

$$f'(\texttt{mask}(a)) = \texttt{mask}(f(a)) \ .$$

Algorithm 9 `RPMask` — Complexité : $16d$ additions

1: **entrée :** une matrice $M \in \mathbb{F}_{2^8}^{4\times 4}$ et l'ordre du masquage d
2: **sortie :** $\vec{M} \in (\mathbb{F}_{2^8}^{d+1})^{4\times 4}$ tel que $\vec{M}_{i,j} = \texttt{RMMaskByte}(M_{i,j}, d)$
3: **pour** $i = 1$ **jusqu'à** 4 **faire :**
4: **pour** $j = 1$ **jusqu'à** 4 **faire :**
5: $\vec{M}_{i,j} \leftarrow \texttt{RPMaskByte}(M_{i,j}, d)$
6: **renvoyer** $\vec{M}$

Pour masquer l'opération `AddRoundKey`, il suffit d'additionner grâce à l'algorithme `RPAdd` chaque octet de la matrice intermédiaire $\vec{M}$ avec son équivalent dans la matrice qui correspondant à la clé de session $\vec{K}$.

Algorithm 10 `RPAddRoundKey` — Complexité : $16(d+1)$ additions

1: **entrée :** $\vec{M} = \texttt{RPMask}(M)$ et $\vec{K} = \texttt{RPMask}(K)$
2: **sortie :** $\vec{C}$ tel que $\vec{C} = \texttt{RPMask}(\texttt{AddRoundKey}(M, K))$
3: **pour** $i = 1$ **jusqu'à** 4 **faire :**
4: **pour** $j = 1$ **jusqu'à** 4 **faire :**
5: $\vec{C}_{i,j} \leftarrow \texttt{RPAdd}(\vec{M}_{i,j}, \vec{K}_{i,j})$
6: **renvoyer** $\vec{C}$

L'algorithme `MixColumns` consiste à multiplier la matrice intermédiaire $\vec{M}$ par la matrice prédéfinie :

$$L = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \ .$$

La version masquée de l'algorithme `MixColumns` consistera donc à remplacer la multiplication ordinaire par `RPMult2`.

Algorithm 11 `RPMixColumns` Complexité : $64(d+1)$ multiplications et $64(d+1)$ additions

entrée : $\vec{M} = \texttt{RPMask}(M)$
sortie : $\vec{C}$ tel que $\vec{C} = \texttt{RPMask}(\texttt{MixColumns}(M))$
pour $i = 1$ **jusqu'à** 4 **faire :**
  pour $j = 1$ **jusqu'à** 4 **faire :**
    $\vec{C}_{i,j} \leftarrow \vec{0}$
    pour $u = 1$ **jusqu'à** 4 **faire :**
      $\vec{C}_{i,j} \leftarrow \texttt{RPAdd}(\texttt{RPMult2}(\vec{M}_{i,u}, L_{u,j}), C_{i,j})$
renvoyer $\vec{C}$

Étant donné que l'algorithme `SubBytes` opère sur les octets de la matrice intermédiaire $\vec{M}$ de manière indépendante, alors il suffira de concevoir un algorithme qui permettra de masquer la transformation pour un seul octet, et l'appliquer ensuite sur les seize octets de la matrice $\vec{M}$.

Soient $a = \bigoplus_{i=0}^{d} a_i \in \mathbb{F}_2^8$ une donnée sensible et $\vec{a} = \texttt{RPMaskByte}(a)$ un masqué associé à cette donnée. Nous noterons, $y = a^{-1}$ l'inverse de a dans $\mathbb{F}_{2^8}$ (i.e. $ay = 1$) et $\vec{y} = \texttt{RPMaskByte}(y)$. L'opération `SubBytes` consiste à calculer $c = S(a)$ tel que :

$$c_j = y_j \oplus y_{(j+4) \bmod 8} \oplus y_{(j+5) \bmod 8} \oplus y_{(j+6) \bmod 8} \oplus y_{(j+7) \bmod 8} \oplus 63_j \ ,$$

avec c_j désigne le i^{eme} bit de l'octet c. Ainsi, nous obtiendrons :

$$\begin{aligned} c_j &= (y_0)_j \oplus (y_0)_{(j+4) \bmod 8} \oplus (y_0)_{(j+5) \bmod 8} \oplus (y_0)_{(j+6) \bmod 8} \oplus (y_0)_{(j+7) \bmod 8} \oplus : b_0 \\ &\quad (y_1)_j \oplus (y_1)_{(j+4) \bmod 8} \oplus (y_1)_{(j+5) \bmod 8} \oplus (y_1)_{(j+6) \bmod 8} \oplus (y_1)_{(j+7) \bmod 8} \oplus : b_1 \\ &\quad \vdots \\ &\quad (y_d)_j \oplus (y_d)_{(j+4) \bmod 8} \oplus (y_d)_{(j+5) \bmod 8} \oplus (y_d)_{(j+6) \bmod 8} \oplus (y_d)_{(j+7) \bmod 8} \oplus : b_d \\ &\quad 63_j \\ &= b_0 \oplus \cdots \oplus b_d \oplus 63 \ . \end{aligned}$$

Nous pouvons donc considérer $\vec{b} = (b_0 \oplus 63, b_1, \ldots, b_d) = \texttt{RPMaskByte}(S(a))$.

Algorithm 12 `RPSubByte` Complexité : $4d^2 + 15d + 4$ multiplications et $8d^2 + 43d + 33$ additions

1: **entrée :** $\vec{a} = \texttt{RPMaskByte}(a)$
2: **sortie :** $\vec{c}$ tel que $\vec{c} = \texttt{RPMaskByte}(\texttt{SubByte}(c))$
3: $\vec{y} \leftarrow \texttt{RPInverse}(\vec{a})$
4: $\vec{c} \leftarrow \vec{0}$
5: **pour** $i = 0$ **jusqu'à** d **faire :**
6: **pour** $j = 0$ **jusqu'à** 8 **faire :**
7: $(c_i)_j \leftarrow (y_i)_j \oplus (y_i)_{(j+4) \bmod 8} \oplus (y_i)_{(j+5) \bmod 8} \oplus (y_i)_{(j+6) \bmod 8} \oplus (y_i)_{(j+7) \bmod 8}$
8: $c_0 \leftarrow c_0 \oplus 63$
9: **renvoyer** $\vec{x}'$

La ligne 5 de l'algorithme permet de parcourir les shares de $\vec{c}$, tandis que la ligne 6 permet de parcourir les bits.
Pour résumer, voici la complexité totale des quatre transformations de l'AES, en fonction du nombre d'additions et multiplications dans $\mathbb{F}_{2^8}$.

	La multiplication	L'addition
`SubBytes`	$64d^2 + 240d + 64$	$128d^2 + 688d + 528$
`ShiftRows`	0	0
`MixColumns`	$64(d + 1)$	$64(d + 1)$
`AddRoundKey`	0	$16(d + 1)$

TABLE 1.1 – Le nombre d'opérations pour chacune des transformations de l'AES avec la méthode masquage RP.

1.2 Le masquage basé sur les polynômes

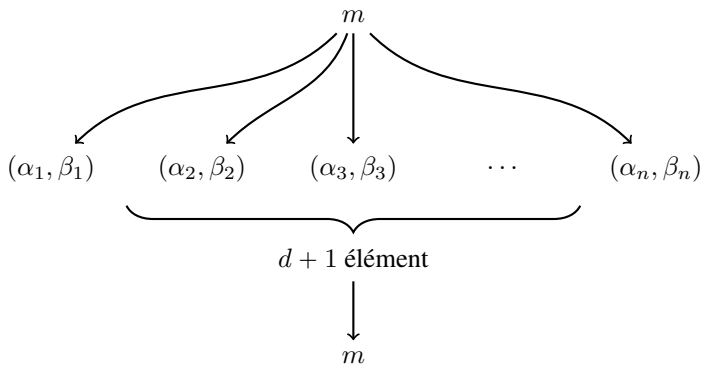
Le masquage basé sur les polynômes a été introduit séparément par Prouff et Roche [PR11], et Goubin et Martinelli [GM11]. Cette méthode combine d'une part le schéma de Shamir sur le partage du secret "*Shamir's Secret Sharing Scheme*" (SSS) [Sha79], qui permet de partager un secret entre un groupe d'entités, et d'autre part les techniques sécurisées de calcul multipartite [BGW88].

1.2.1 Le schéma de Shamir sur le partage du secret

Les schémas de partage du secret peuvent être très utiles pour la gestion des clés cryptographiques. Afin de protéger les données, nous pouvons les chiffrer, mais pour protéger la clé secrète, nous avons besoin d'une méthode différente. Dans son livre "Introduction To Combinatorial Mathematics"[Liu68] Liu avait introduit le problème suivant :

> Onze scientifiques travaillent sur un projet secret. Ils souhaitent verrouiller les documents dans un cabinet afin que le cabinet puisse être ouvert si et seulement si six ou plus des scientifiques sont présents. Quel est le plus petit nombre de serrures nécessaires ? Quel est le plus petit nombre de clés pour les verrous que chaque scientifique doit porter ?

Il est facile de constater que dans le cas de ce problème, le nombre minimal de serrures nécessaire est : $\binom{11}{6} = 462$ et le nombre minimal de clés par personne est $\binom{10}{5} = 252$. Cependant, cette solution reste théorique à cause du nombre exponentiel de clés nécessaires, et qui devient de plus en plus important lorsque le nombre d'utilisateurs augmente. Pour répondre à cette problématique d'un point de vue cryptographique Shamir a proposé un schéma de partage du secret qui nécessite un nombre de clés constant, dans ce schéma chaque utilisateur possède une seule clé quel que soit le nombre d'utilisateurs. Dans son article [Sha79], Shamir a présenté un schéma qui permet de diviser une donnée m en n morceaux de telle sorte que m soit facilement reconstruite à partir de $d+1$ morceaux quels qu'ils soient (avec $d < n$), mais même une connaissance complète de d morceaux ne révèle absolument aucune information sur la donnée secrète m.
Soit $f : F \to E$ une fonction polynomiale de degré d définie par : $f(x) = c_0 + \sum_{i=1}^{d} c_i x^i$. Soient $\alpha_1, \alpha_2, \ldots, \alpha_n \in F^*$ des valeurs distinctes non nulles, avec un nombre n d'utilisateurs strictement inférieur au cardinal de F. Nous noterons $\beta_i = f(\alpha_i) \in E$ l'image de α_i par f pour $1 \leqslant i \leqslant n$. Le schéma SSS consiste à protéger l'information c_0 (le coefficient de degré 0 de f) et la rendre accessible si et seulement si au moins $d+1$ utilisateurs réunissent leurs clés. Pour ce faire, il suffit de transmettre à chaque utilisateur une clé privée β_i (l'entrée α_i peut être supposée publique), et maintenir la fonction f secrète pour tous les utilisateurs.



Ainsi, par interpolation, pour construire f et trouver le coefficient c_0 (sachant que $c_0 = f(0)$) il faut et il suffit d'avoir au moins $d+1$ entrées et leurs images par f.

Définition 19 (L'interpolation de Lagrange)
Étant donné un ensemble de $n+1$ points : $(\alpha_0, \beta_0), \ldots, (\alpha_n, \beta_n)$ dans $F \times E$, tels que tous les α_i sont différents, il existe un polynôme unique f de degré n tel que $f(\alpha_i) = \beta_i$ avec $0 \leqslant i \leqslant n$. Ce polynôme s'appelle *le polynôme d'interpolation de Lagrange*, et il est défini par la formule suivante :

$$f(x) = \sum_{j=0}^{n} \beta_j \left(\prod_{i=0, i\neq j}^{n} \frac{x - \alpha_i}{\alpha_j - \alpha_i} \right) .$$

Pour vérifier cette formule, il suffit de remarquer que :

$$\prod_{i=0, i\neq j}^{n} \frac{x - \alpha_i}{\alpha_j - \alpha_i} = \begin{cases} 1 \text{ si } x = \alpha_j \\ 0 \text{ si } x \neq \alpha_j \end{cases} ,$$

donc, pour n'importe quel α_k nous avons :

$$\begin{aligned} f(\alpha_k) &= \beta_k \underbrace{\left(\prod_{i=0, i\neq k}^{n} \frac{\alpha_k - \alpha_i}{\alpha_k - \alpha_i} \right)}_{=1} + \textstyle\sum_{j=0, j\neq k}^{n} \beta_j \underbrace{\left(\prod_{i=0, i\neq j}^{n} \frac{\alpha_k - \alpha_i}{\alpha_j - \alpha_i} \right)}_{=0} \\ &= \beta_k \end{aligned}$$

Il est également possible d'interpréter f comme un système d'équations défini par :

$$\underbrace{\begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \ldots & \alpha_0^d \\ 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^d \\ & & & & \\ \vdots & & & & \vdots \\ & & & & \\ 1 & \alpha_d & \alpha_d^2 & \ldots & \alpha_d^d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ \\ \vdots \\ \\ c_d \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \\ \vdots \\ \\ \beta_d \end{bmatrix}}_{B} \tag{1.1}$$

Étant donné que les α_i sont tous distincts et non nuls, la matrice A (qui est publique) est donc inversible et nous pourrons ainsi calculer les coefficients de f donnés par : $C = A^{-1} \times B$. Comme nous pouvons le constater, ce système n'est soluble que si nous avons réuni au moins $d + 1$ clés (α_i, β_i), et le nombre de clés pour chaque utilisateur reste constant quel que soit le nombre d'utilisateurs au total.
Cette façon d'encoder l'information en utilisant des systèmes d'équations linéaires a été introduite initialement par Reed et Solomon dans leur célèbre article sur le code de Reed-Solomon [RS60].

1.2.2 Les codes de Reed-Solomon

Un code Reed-Solomon (RS) est un code correcteur d'erreurs décrit pour la première fois dans un article de Reed et Solomon en 1960 [RS60]. Grace à leurs distance minimale optimale, les codes RS ont été utilisés pour protéger les information sur les CD-ROM, les DVD, les communications sans fil, les communications spatiales, etc.
Il existe différentes procédures de codage pour les codes RS et, par conséquent, il existe différentes façons de décrire l'ensemble des mots de code. Dans la vue originale de Reed et Solomon, dans un code RS de paramètres $[n, k, d]$ chaque mot de code est une séquence de valeurs de fonction d'un polynôme de degré inférieur à k. Pour obtenir un mot de code, le message est interprété comme la description d'un polynôme P de degré inférieur à k sur le corps fini $\mathbb{F}_q$. À son tour, le polynôme P est évalué en $k \leqslant n \leqslant q$ points distincts.
L'idée consiste à considérer le mot source $m = (m_0, \ldots, m_{k-1})$ comme un polynôme sur un corps fini, où les symboles qui composent le message (par exemple, les octets) sont des coefficients de ce polynôme $P_m(x) = m_0 + m_1 x^1 + \cdots + m_{k-1} x^{k-1}$. Ainsi, le mot de code associé à m est $c = (P_m(x_i))_{1 \leqslant i \leqslant n}$, où x_i sont des points distincts de $\mathbb{F}_q$. Pour reconstruire le polynôme $P_m(x)$ il suffit d'obtenir k points $(x_i, P_m(x_i))$, on en déduit alors que la distance minimale d'un code RS est $n - k + 1$ et sa capacité de correction vaut : $\left\lfloor \frac{n-k}{2} \right\rfloor$. Ainsi, les codes RS atteignent bien la borne de Singleton, et sont alors des codes MDS.

1.2.3 Le masquage basé sur le partage du secret

Le masquage booléen comme celui que nous avons présenté précédemment permet de sécuriser efficacement les implémentations contre les SCA du premier ordre. Il est cependant particulièrement vulnérable aux SCA d'ordre supérieur en raison des propriétés physiques intrinsèques des dispositifs électroniques [Mes00]. En 2011, Goubin et Martinelli [GM11] en parallèle avec Prouff et Roche [PR11] ont présenté une nouvelle méthode de masquage basée sur le schéma SSS de la section

précédente. Le masquage d'une information sensible $a \in \mathbb{F}_{2^8}$ consiste à générer une fonction $f_a : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ de degré d telle que $f(x) = a + \sum_{i=1}^{d} c_i x^i$, et renvoyer le vecteur $\vec{a} = (f_a(\alpha_1), \ldots, f_a(\alpha_n))$ avec $\alpha_1, \ldots, \alpha_n$ des éléments publics dans $\mathbb{F}_{2^8}$ et $n = 2d + 1$.
Soit A une matrice dans $\mathbb{F}_{2^8}^{n \times n}$ définie par :

$$A_{i,j} = \alpha_i^j \ . \tag{1.2}$$

Pour masquer une valeur sensible a, il suffit générer un vecteur aléatoire $\vec{c} \in \mathbb{F}_{2^8}^n$ tel que $c_0 = a$, et calculer ensuite :

$$\texttt{mask}(a) = A.\vec{c} \ . \tag{1.3}$$

Algorithm 13 `mask` — Complexité : n^2 multiplications et n^2 additions

1: **entrée :** $a \in \mathbb{F}_{2^8}$
2: **sortie :** $\vec{a} \in \mathbb{F}_{2^8}^n$
3: $\vec{c} \xleftarrow{\$} \mathbb{F}_{2^8}^n$ — Génération des coefficients de f
4: $c_0 \leftarrow a$ — Le premier coefficient vaut a
5: $\vec{a} \leftarrow A.\vec{c}$
6: **renvoyer** $\vec{a}$

Pour démasquer nous avons besoin d'extraire le premier élément du vecteur $\vec{c}$ de l'algorithme 13. Nous avons : $\vec{c} = A^{-1}.\vec{a}$, donc :

$$a = \langle A_0^{-1}, \vec{a} \rangle \ , \tag{1.4}$$

avec A_0^{-1} correspond à la première ligne de la matrice inverse de A.

Algorithm 14 `unmask` — Complexité : n multiplications et n additions

1: **entrée :** $\vec{a} \in \mathbb{F}_{2^8}^n$
2: **sortie :** $a \in \mathbb{F}_{2^8}$
3: **renvoyer** $\langle A_0^{-1}, \vec{a} \rangle$

Nous remarquons que cette méthode de masquage consiste à utiliser les codes de Reed-Solomon pour protéger l'information, et comme $n = 2k + 1 > k$, cette méthode permet de détecter jusqu'à $k + 1$ erreur. Ainsi, ce masquage a l'avantage de profiter de nombreux résultats déjà publiés sur ce code.

Propriété 10
Le masquage basé sur le partage du secret est une généralisation du masquage booléen.

Pour démontrer cette propriété, il suffit de constater que le cas particulier $A_0^{-1} = (1, \ldots, 1)$ est un masquage booléen.
Le code RS étant linéaire, le masquage de l'addition et la multiplication par un scalaire sont triviales. Cependant, le produit de deux masqués ne peut pas être résolu avec la propriété linéaire de la transformation car la multiplication de deux polynômes de même degré donne un polynôme de degré double du polynôme d'origine. Goubin et Martinelli, ainsi que Prouff et Roche ont proposé deux approches différentes pour calculer la multiplication en utilisant le schéma de calcul multipartite BGW [BGW88, GRR98] qui utilise $2d + 1$ parts (d'ou le choix $n = 2d + 1$).

1.2.4 Le schéma sécurisé de calcul multipartite BGW

Ben-Or, Goldwasser et Wigderson [BGW88] ont présenté une méthode simple pour réduire le degré de la fonction $h(x) = f_a(x)f_b(x)$. Le but de ce schéma est de calculer de manière sécurisé $(f_{ab}(\alpha_1), \ldots, f_{ab})(\alpha_n)$ à partir de $(f_a(\alpha_1), \ldots, f_a(\alpha_n))$ et $(f_b(\alpha_1), \ldots, f_b(\alpha_n))$. Soit :

$$h(x) = h_0 + h_1 x + \cdots + h_d x^d + \cdots + h_{2d} x^{2d} \ ,$$

avec $h_0 = ab$, et soit : $\vec{s} = (s_0, \ldots s_{2d})$ tel que $s_i = h(\alpha_i) = f_a(\alpha_i)f_b(\alpha_i)$.
Soit $k(x)$ une troncature de h définie par :

$$k(x) = h_0 + h_1 x + \cdots + h_d x^d \ .$$

L'objectif est donc de trouver le vecteur $\vec{r} = (r_0, \ldots, r_{2d})$ tel que : $r_i = k(\alpha_i)$.

Proposition 1 ([BGW88])
Soient $\vec{s} = (s_0, \ldots, s_{2d})$ et $\vec{r} = (r_0, \ldots r_{2d})$ définis ci-dessus, il existe une matrice constante $M \in \mathbb{F}^{2d \times 2d}$ telle que :

$$\vec{r} = M\vec{s} \ .$$

Preuve :
Soient $\vec{h} = (h_0, \ldots, h_{2d}) \in \mathbb{F}^{2d}$, $\vec{k} = (h_0, \ldots, h_d, 0, \ldots, 0) \in \mathbb{F}^{2d}$, $A \in \mathbb{F}^{2d \times 2d}$ telle que $A_{i,j} = \alpha_i^j$ et $P \in \mathbb{F}^{2d \times 2d}$ une matrice de projection telle que :

$$P \cdot (x_0, \ldots x_{2d}) = (x_0, \ldots, x_d, 0, \ldots, 0) \ .$$

Nous avons :

$$\begin{aligned} \vec{r} &= A\vec{k} \\ &= AP\vec{h} \\ &= (APA^{-1})\vec{s} \ , \end{aligned}$$

car A est une matrice inversible. ■

Grâce à cette proposition nous pouvons construire une fonction k de degré d et calculer le vecteur $\vec{r} = k(\vec{\alpha})$ à partir des masques de a et b. Cependant, comme nous l'avons mentionné auparavant, les coefficients de k ne sont pas aléatoires, et donc, par mesure de sécurité il faudra rafraîchir le masque obtenu. Pour ce faire, l'idée consiste à construire un polynôme $k'(x) = k(x) + q(x)$, avec q un polynôme aléatoire défini par $q(x) = \sum_{i=1}^{2d} q_i x^i$ (On note que le coefficient de degré zéro est nul). Cela revient à additionner un masqué du produit ab (qui est le polynôme k) et le masqué de l'élément 0 (qui est le polynôme q).

1.2.5 L'algorithme de multiplication de Prouff et Roche

Soient a et b deux valeurs sensibles, et $(f_a(\alpha_i))_{1\leqslant i\leqslant n}$ et $(f_b(\alpha_i))_{1\leqslant i\leqslant n}$, les parts associées à a et b. Pour mieux visualiser l'algorithme de multiplication de Prouff et Roche il faudra imaginer n joueurs, tels que chaque joueur i possède deux parts $f_a(\alpha_i)$ et $f_b(\alpha_i)$, et calcule $w_i = f_a(\alpha_i)f_b(\alpha_i)$ pour construire un polynôme de degré $2d$ dont le coefficient de degré zéro est ab, ensuite, chaque joueur va "masquer" w_i en générant un polynôme $Q_i = w_i + \sum_{k=1}^{d} r_k \alpha_j^k$, et partager $Q_i(\alpha_j)$ avec chaque joueur j. Enfin, chaque joueur pourra "démasquer" le vecteur $Q_{*,i}$ (qui correspond à la $i$ème colonne de Q) qu'il a reçu pour en extraire sa part $f'_{ab}(\alpha_i)$.

Algorithm 15 `Mult` Complexité : $n^2(d+1) + n$ multiplications et $n^2(d+2)$ additions

entrée : $\texttt{mask}(a), \texttt{mask}(b) = (f_a(\alpha_i))_{1\leqslant i\leqslant n}, (f_b(\alpha_i))_{1\leqslant i\leqslant n} \in \mathbb{F}_{2^8}^n$.
sortie : $\texttt{mask}(ab)$.
$Q \in \mathbb{F}_{2^8}^{n\times n}$
$\vec{w} \in \mathbb{F}_{2^8}^n$
pour $i = 1$ **jusqu'à** n **faire :**
    $w_i \leftarrow f_a(\alpha_i)f_b(\alpha_i)$
    $r \leftarrow \mathbb{F}_{2^8}^d$
    pour $j = 1$ **jusqu'à** n **faire :**
        $Q_{i,j} \leftarrow w_i + \sum_{k=1}^{d} r_k \alpha_j^k$ Masquage de w_i
$\vec{c} \in \mathbb{F}_{2^8}^n$
pour $i = 1$ **jusqu'à** n **faire :**
    $c_i \leftarrow \texttt{unmask}(Q_{*,i})$
renvoyer $\vec{c}$

1.3 Le masquage basé sur le produit scalaire (IPM)

Le masquage IPM (*Inner Product Masking*) ou le masquage basé sur le produit scalaire a été introduit par Balasch *et al.* en 2012 [BFGV12, BFG15]. Comme son nom l'indique, l'opération de masquage consiste à générer deux vecteurs $\vec{u}$ et $\vec{v}$ de longueur $d+1$ tels que leur produit scalaire vaut m ($\langle \vec{u}, \vec{v} \rangle = m$), avec m désignant la donnée sensible. Ainsi, assurer une protection d'ordre d contre SCA.

Algorithm 16 `IPmask` Complexité : n multiplications et n additions

entrée : $m \in \mathbb{F}_{2^8}$.
sortie : $(\vec{u}, \vec{v})$ tels que $\langle \vec{u}, \vec{v} \rangle = m$.
$c \leftarrow 0 \in \mathbb{F}_{2^8}$
pour $i = 2$ **jusqu'à** n **faire :**
    $u_i \xleftarrow{\$} \mathbb{F}_{2^8}^*$
    $v_i \xleftarrow{\$} \mathbb{F}_{2^8}$
    $c \leftarrow c \oplus u_i v_i$
$u_1 \xleftarrow{\$} \mathbb{F}_{2^8}^*$
$v_1 \leftarrow (m \oplus c) u_1^{-1}$
renvoyer $(\vec{u}, \vec{v})$

Si nous considérons le cas particulier $\vec{u} = \vec{1} = (1, \ldots, 1)$, alors le masquage consistera à générer un vecteur $\vec{v} = (v_1, \ldots, v_n)$ tel que : $\sum_{i=1}^{n} v_i = m$. Ainsi, nous retrouvons la méthode de masquage classique que nous avons abordé précédemment. De la même façon, nous avons vu dans l'équation 1.4 page 56, que le masquage basé sur le partage du secret consiste à calculer un produit scalaire entre le vecteur $\vec{a} = (f(\alpha_0), \ldots, f(\alpha_d))$ et le vecteur $\vec{v} = A_0^{-1}$ qui correspond à la première ligne de l'inverse de la matrice A définie dans l'équation 1.1. Nous constatons ainsi que :

Propriété 11
Le schéma IPM est une généralisation du masquage classique et du masquage basé sur le partage du secret.

Dans le schéma original [BFGV12] les deux vecteurs $\vec{u}$ et $\vec{v}$ changent en permanence. Cependant, les auteurs ont proposé une autre version en 2015 [BFG15] dans laquelle le vecteur $\vec{u}$ est supposé public, avec $u_0 = 1$, et reste identique durant tout le calcul, ce choix permet d'optimiser la complexité algorithmique des calculs.

Algorithm 17 `IPmask2` Complexité : $n-1$ multiplications et $n-1$ additions

entrée : $m \in \mathbb{F}_{2^8}$.
sortie : $\vec{z}$ tel que $\langle \vec{u}, \vec{z} \rangle = m$.
$z \in \mathbb{F}_{2^8}^n$
$z_0 \leftarrow 0$
pour $i = 2$ **jusqu'à** n **faire :**
    $z_i \xleftarrow{\$} \mathbb{F}_{2^8}$
    $z_0 \leftarrow z_0 \oplus u_i z_i$
renvoyer $\vec{z}$

En fixant $\vec{u}$, on obtient un masquage linéaire :

$$\langle \vec{u}, \vec{z} \rangle + \langle \vec{u}, \vec{z'} \rangle = \langle u, z + z' \rangle \ .$$

Pour calculer la multiplication nous considérons la matrice $L \in \mathbb{F}_{2^8}^{n \times n}$ définie par :

$$\forall\ 1 \leqslant i, j \leqslant n, \quad L_{i,j} = u_i u_j \ .$$

Algorithm 18 `IPMult` Complexité : $3n^2 - n$ multiplications et $3n^2 - n - 1$ additions

entrée : $\vec{z} =$ `IPmask2`$(m), \vec{z'} =$ `IPmask2`(m').
sortie : $\vec{y} =$ `IPmask2`(mm').
$A \xleftarrow{\$} \mathbb{F}_{2^8}^{n \times n} \quad | \quad \sum_{i=1}^{n} \sum_{j=1}^{n} L_{i,j} A_{i,j} = 0$
$R \in \mathbb{F}_{2^8}^{n \times n} \quad | \quad R_{i,j} = z_i z'_j$
$B \leftarrow R + A$
$b \leftarrow 0 \in \mathbb{F}_{2^8}$
pour $i = 2$ **jusqu'à** n **faire :**
    pour $j = 1$ **jusqu'à** n **faire :**
        $b \leftarrow b + L_{i,j} B_{i,j}$
renvoyer $\vec{y} = (B_{1,1} + b, B_{1,2}, \ldots, B_{1,n})$

Démonstration :
Nous avons :

$$\begin{aligned} b &= \sum_{i=2}^{n} \sum_{j=1}^{n} L_{i,j} B_{i,j} \\ &= \sum_{i=2}^{n} \sum_{j=1}^{n} u_i u_j (z_i z'_j + A_{i,j}) \\ &= \sum_{i=2}^{n} (u_i z_i \sum_{j=1}^{n} u_j z'_j + L_{i,j} A_{i,j}) \\ &= \sum_{i=2}^{n} (u_i z_i m' + \sum_{j=1}^{n} L_{i,j} A_{i,j}) \\ &= (m' \sum_{i=2}^{n} u_i z_i) + (\sum_{i=2}^{n} \sum_{j=1}^{n} L_{i,j} A_{i,j}) \\ &= m'(m - z_1) + \sum_{j=1}^{n} L_{1,j} A_{1,j} \ . \end{aligned}$$

Ainsi, nous obtenons :

$$\begin{aligned}\langle \vec{u}, \vec{y} \rangle &= b + \sum_{i=1}^{n} u_i B_{1,i} \\ &= m'(m - z_1) + \sum_{j=1}^{n} L_{1,j} A_{1,j} + \sum_{i=1}^{n} u_i (R_{1,i} + A_{1,i}) \\ &= m'(m - z_1) + \sum_{j=1}^{n} L_{1,j} A_{1,j} + \sum_{i=1}^{n} u_i (z_1 z'_i + A_{1,i}) \\ &= m'(m - z_1) + \sum_{j=1}^{n} L_{1,j} A_{1,j} + (z_1 \sum_{i=1}^{n} u_i z'_i) + (\sum_{i=1}^{n} u_i A_{1,i}) \\ &= m'(m - z_1) + \sum_{j=1}^{n} L_{1,j} A_{1,j} + z_1 m' + \sum_{i=1}^{n} L_{1,i} A_{1,i} \\ &= mm' \ .\end{aligned}$$

Pour construire la matrice A de la ligne 3, il suffit de tirer une matrice aléatoire $A \xleftarrow{\$} \mathbb{F}_{2^n}$ telle que :

$$A_{n,n} = \Big[\sum_{j=1}^{n-1} \sum_{i=1}^{n} A_{i,j} L_{i,j} + \sum_{i=1}^{n-1} A_{i,n} L_{i,n} \Big] L_{n,n}^{-1} \ .$$

Ainsi, le coût de cette opération est : n^2 multiplications et n^2 additions.
Enfin, il est toujours avantageux d'avoir une implémentation particulièrement efficace pour calculer le carré. Nous avons :

$$\begin{aligned} m^2 &= (\sum_{i=1}^{n} u_i z_i)^2 \\ &= \sum_{i=1}^{n} (u_i z_i)^2 \\ &= \sum_{i=1}^{n} u_i (u_i z_i^2) \ .\end{aligned}$$

Ainsi, on en déduit l'algorithme suivant :

Algorithm 19 `Exp2` — Complexité : $2n$ multiplications

1: **entrée :** $\vec{z} =$ `IPmask2`(m).
2: **sortie :** $\vec{y} =$ `IPmask2`(m^2).
3: $\vec{y} \leftarrow \vec{0} \in \mathbb{F}_{2^8}^n$
4: **pour** $i = 1$ **jusqu'à** n **faire :**
5: $\quad y_i \leftarrow u_i z_i^2$
6: **renvoyer** $\vec{y}$

1.4 Le masquage basé sur les codes

Un inconvénient principal des schémas de masquage est la surcharge importante nécessaire à leur réalisation. Le coût des calculs est généralement important du fait du traitement du masque, et en particulier de la multiplication. De plus, les schémas de masquage supposent habituellement des masques aléatoires distribués uniformément, autrement dit, un générateur aléatoire suffisamment bon doit être implémenté en plus du système de chiffrement protégé. Par conséquent, le masquage ajoute généralement du temps et de l'espace à des implémentations cryptographiques aussi bien matérielles que logicielles.
Pour surmonter ces obstacles, il existe des schémas de masquage à faible entropie, ou ce qu'on appelle LEMS (*Low Entropy Masking Schemes*). Le but est d'offrir un compromis raisonnable entre la bonne protection contre les attaques SCA offerte par le masquage et le coût qui résulte de leur mise en œuvre. Pour mettre en œuvre de tels schémas, plusieurs solutions ont été proposées [PR07, CDG+14, CDGM12, Car13]. Une solution également appelée méthode RSM (*Rotating Sbox Masking*) [NSGD12] consiste à précalculer et stocker des boites S masquées qui correspondent à toutes les valeurs possibles de masques. D'autres consistent à calculer les boites S masquées pour le masque choisi ...
Dans cette section nous présentons une méthode simplifiée pour générer rapidement des masques uniformes. Cette méthode consiste à utiliser des codes correcteurs d'erreurs, et en particulier les codes linéaires.

1.4.1 Rappels

Soit k et n deux entiers naturels, tels que $k \leqslant n$. On note $\mathbb{F}_2^n$ l'espace vectoriel de longueur n dans $\mathbb{F}_2 = \{0, 1\}$. Soit $\mathcal{C}$ un sous espace vectoriel de $\mathbb{F}_2^n$ de dimension k, $\mathcal{C}$ est donc un code linéaire de longueur n de dimension k.

Définition 20 (espaces vectoriels supplémentaires)
Soit $\mathcal{C}$ et $\mathcal{D}$ deux sous espaces vectoriels de $\mathbb{F}_2^n$. On dit que $\mathcal{C}$ et $\mathcal{D}$ sont supplémentaires, si tout vecteur de $\mathbb{F}_2^n$ s'écrit de manière unique comme la somme d'un vecteur de $\mathcal{C}$ et d'un vecteur de $\mathcal{D}$:

$$\forall z \in \mathbb{F}_2^n, \quad \exists!(c, d) \in \mathcal{C} \times \mathcal{D} \quad | \quad z = c + d \ , \tag{1.5}$$

et on note $\mathbb{F}_2^n = \mathcal{C} \oplus \mathcal{D}$.

Propriété 12
Les propriétés suivantes sont équivalentes :

1. $\mathcal{C}$ et $\mathcal{D}$ sont supplémentaires,

2. l'application $\mathcal{C} \times \mathcal{D} \to \mathbb{F}_2^n, (c,d) \mapsto c+d$ est bijective,
3. $\mathcal{C} \cap \mathcal{D} = \{0\}$ et $dim(\mathcal{C}) + dim(\mathcal{D}) = n$.

Preuve :

— $1 \Leftrightarrow 2$: la surjectivité correspond à l'existence d'un couple (c,d) dans $\mathcal{C} \times \mathcal{D}$, pour tout z dans $\mathbb{F}_2^n$, et l'injectivité, correspond à l'unicité de (u,v), qui est par définition le cas pour $\mathcal{C}$ et $\mathcal{D}$ supplémentaires.

— $1 \Rightarrow 3$: Supposons qu'il existe un vecteur $x \in \mathcal{C} \cap \mathcal{D}$. Par définition :

$$\forall z \in \mathbb{F}_2^n, \quad \exists!(c,d) \in \mathcal{C} \times \mathcal{D} \quad | \quad z = c + d \ .$$

Or pour $z = 0$ nous avons : $z = 0 \oplus 0 = x \oplus x$, donc $x = 0$ car il existe un unique couple (c,d). Ainsi, $\mathcal{C} \cap \mathcal{D} = \{0\}$, et puisque $\mathcal{C} \oplus \mathcal{D} = \mathbb{F}_2^n$ alors $dim(\mathcal{C}) + dim(\mathcal{D}) = n$.

— $1 \Leftarrow 3$: Nous avons : $dim(\mathcal{C}) + dim(\mathcal{D}) = dim(\mathbb{F}_2^n) \Rightarrow \forall z \in \mathbb{F}_2^n, \quad \exists(c,d) \in \mathcal{C} \times \mathcal{D} \quad | \quad z = c + d$, et puisque $\mathcal{C} \cap \mathcal{D} = \{0\}$, alors ce couple (c,d) est unique.

■

Définition 21 (Le code dual)
Soit $\mathcal{C}$ un code linéaire de $\mathbb{F}^n$. L'orthogonal de $\mathcal{C}$ est un espace vectoriel que l'on note $\mathcal{C}^\perp$ défini par :

$$\mathcal{C}^\perp = \{d \in \mathbb{F}^n \quad | \quad \forall c \in \mathcal{C} \quad \langle d, c \rangle = 0\} \ .$$

Avec $\langle \ , \ \rangle$ désigne le produit scalaire usuel. Dans le cas des codes linéaires, on dit que $\mathcal{C}^\perp$ est le code dual de $\mathcal{C}$.

Définition 22 (Les codes LCD [Mas92])
Soit $\mathcal{C}$ un code linéaire de paramètres $[n,k,d_\mathcal{C}]$, et $\mathcal{D}$ son code dual. On dit que $\mathcal{C}$ est un code LCD (*Linear code with complementary dual*) si $\mathcal{C}$ et $\mathcal{D}$ sont supplémentaires.

On notera pour la suite G et H les matrices génératrices de $\mathcal{C}$ et $\mathcal{D}$ respectivement.

Propriété 13
Soient $\mathcal{C}$ un code linéaire de paramètres $[n,k,d_\mathcal{C}]$, et G sa matrice génératrice. $\mathcal{C}$ est un code LCD si et seulement si la matrice $GG^\top$ est inversible.

Preuve :
Pour démontrer cette propriété nous allons démontrer chaque implication. Dans le théorème 12 nous avons montré qu'un code $\mathcal{C}$ est LCD si et seulement si $\mathcal{C} \cap \mathcal{D} = \{0\}$, il suffit donc de démontrer les implications suivantes :

1. GG^T est inversible $\implies \mathcal{C} \cap \mathcal{D} = \{0\}$;
2. GG^T n'est pas inversible $\implies \mathcal{C} \cap \mathcal{D} \neq \{0\}$ (la contraposée de $\mathcal{C} \cap \mathcal{D} = \{0\} \implies GG^T$ est inversible).

1. Soit $x \in \mathbb{F}_2^k$ un mot source et $c = xG \in \mathcal{C}$ le mot de code associé à x, nous avons : $cG^\top(GG^\top)^{-1}G = xGG^\top(GG^\top)^{-1}G = xG = c$. Ainsi, si en plus $c \in \mathcal{D}$ (i.e. $cG^\top = 0$) alors : $cG^\top(GG^\top)^{-1}G = c = 0$.
2. Inversement, si nous supposons que $GG^\top$ n'est pas inversible, alors il existe un vecteur non-nul $x \in \mathbb{F}_2^k$ tel que $x(GG^\top) = 0$. Soit $c = xG \in \mathcal{C}$ le mot de code associé à x. Nous avons $\forall x' \in \mathbb{F}_2^k$:

$$\begin{aligned} \langle xG, x'G \rangle &= xG(x'G)^\top \\ &= xGG^\top x'^\top \\ &= 0 \ . \end{aligned}$$

Donc c est aussi un mot du dual, ce qui implique que dans ce cas $\mathcal{C} \cap \mathcal{D} \neq \{0\}$. ■

Soit $\mathcal{C}$ un code LCD de paramètres $[n, k, d_\mathcal{C}]$, il existe une projection orthogonale de l'espace vectoriel $\mathbb{F}_2^n$ vers $\mathbb{F}_2^k$ définie par :

$$\Pi_\mathcal{C} = G^\top(GG^\top)^{-1} \ . \tag{1.6}$$

En parallèle, puisque le dual de $\mathcal{C}$ est aussi un code LCD, il existe une projection $\Pi_\mathcal{D}$ définie par :

$$\Pi_\mathcal{D} = H^\top(HH^\top)^{-1} \ . \tag{1.7}$$

En effet, par définition, chaque élément de $\mathbb{F}_2^n$ s'écrit de manière unique comme somme de deux éléments de $\mathcal{C}$ et $\mathcal{D}$. i.e. $\forall z \in \mathbb{F}_2^n$, $\exists! c, d \in \mathcal{C} \times \mathcal{D}$ tels que $z = c + d = xG + yH$. Alors :

$$x = zG^\top(GG^\top)^{-1} \ , \tag{1.8}$$

$$y = zH^\top(HH^\top)^{-1} \ . \tag{1.9}$$

1.4.2 Le masquage par somme directe orthogonale (ODSM)

Le masquage ODSM (*Orthogonal Direct Sum Masking*) a été présenté pour la première fois en 2014 dans [BCC+14]. Il s'agit d'un LEMS qui protège contre les attaques par canaux cachés et les attaques par injection de fautes.
Le masquage consiste à choisir un code LCD $\mathcal{C}$ de paramètres $[n, k, d_\mathcal{C}]$ pour encoder l'information sensible $x \in \mathbb{F}_2^k$ (i.e. calculer $c = xG$), et ensuite la masquer par l'addition d'un mot aléatoire du dual. Le masque agira dans ce cas comme un bruit intentionnellement ajouté. La fonction de masquage est définie par :

$$\texttt{mask}(x) = xG + rH \ , \tag{1.10}$$

où G est la matrice génératrice de $\mathcal{C}$, H est la matrice génératrice de $\mathcal{C}^{\perp}$, et r un vecteur aléatoire dans $\mathbb{F}_2^{n-k}$.
Comme l'information et le bruit appartiennent à deux sous-espaces supplémentaires, il est toujours possible de récupérer les deux en utilisant la fonction de projection de l'équation 1.6. Cette projection sera utilisée principalement pour extraire le masque ou bien pour démasquer la donnée sensible à la fin du programme de (dé)chiffrement. Pour extraire la donnée sensible x à partir d'un masqué $z \in \mathbb{F}_2^n$ il suffit d'appliquer l'opération suivante :

$$\texttt{unmask}(z) = zG^{\top}(GG^{\top})^{-1} \ . \tag{1.11}$$

Pour appliquer ce masquage à l'AES, nous allons classer les transformations qui composent ce dernier en trois types d'opérations :

(*i*) L'addition : qui opère sur deux octets ;

(*ii*) Les transformations linéaires : qui opèrent sur un seul octet (utilisée dans la transformation `MixColumns` et en partie de `SubBytes`) ;

(*iii*) Les transformations non linéaires : qui opèrent sur un seul octet (comme la `SubBytes`).

1.4.2.1 L'addition

Comme dans la plupart des schémas de masquage, l'addition reste l'opération la plus simple à masquer. Puisque l'ODSM est une opération linéaire, il est donc évident de calculer l'addition de deux masqués. Soient x_1, x_2 deux mots sensibles dans $\mathbb{F}_2^k$, et $z_1 = \texttt{mask}(x_1)$, $z_2 = \texttt{mask}(x_2)$ les masqués associés à x_1 et x_2 respectivement. Nous avons :

$$\begin{aligned} z_1 \oplus z_2 &= (x_1 G \oplus r_1 H) \oplus (x_2 G \oplus r_2 H) \\ &= (x_1 \oplus x_2)G \oplus (r_1 \oplus r_2)H \\ &= \texttt{mask}(x_1 \oplus x_2) \ . \end{aligned}$$

Masquer l'addition consiste donc à additionner les deux masqués :

$$\texttt{Add}(z_1, z_2) = z_1 \oplus z_2 \ . \tag{1.12}$$

Il est également possible d'additionner un masqué z avec une donnée non masquée k dans le but d'obtenir un résultat qui garde le même masque r :

$$z \oplus kG = (x \oplus k)G \oplus rH \ .$$

1.4.2.2 Les transformations linéaires

L'opération `MixColumns` est une fonction linaire qui opère sur des blocs de quatre octets et renvoie un seul octet, On peut la résumer dans la fonction suivante :

$$f(x_1, x_2, x_3, x_4) = 02x_1 \oplus x_2 \oplus x_3 \oplus 03x_4 \ . \tag{1.13}$$

Les coefficients 02 et 03 appartiennent au corps $K = \mathbb{F}_2[X]/X^8 + X^4 + X^3 + X + 1$ et représentent les éléments suivants α et $\alpha + 1$ respectivement (voir l'exemple 6 page 33).

Proposition 2
il existe un isomorphisme Ψ entre le groupe additif $\mathbb{F}_{2^k}$ et l'espace vectoriel $\mathbb{F}_2^k$ défini par :

$$\forall\ x = \sum_{i=0}^{k-1} c_i \alpha^i \in \mathbb{F}_{2^k}, \quad \Psi(x) = (c_0, \ldots, c_{k-1}) \in \mathbb{F}_2^k \ .$$

Nous avons donc :

$$\Psi(x \oplus y) = \Psi(x) \oplus \Psi(y) \ .$$

Exemple 8
Soient $x = 1 + \alpha^3 + \alpha^6 + \alpha^7 \in K$ et $y = \alpha^3 + \alpha^4 + \alpha^6 \in K$ nous avons :

$$\begin{aligned} \Psi(x \oplus y) &= \Psi(1 + \alpha^3 + \alpha^6 + \alpha^7 + \alpha^3 + \alpha^4 + \alpha^6) \\ &= \Psi(1 + \alpha^4 + \alpha^7) \\ &= (10001001) \ . \end{aligned}$$

$$\begin{aligned} \Psi(x) \oplus \Psi(y) &= (10010011) \oplus (00011010) \\ &= (10001001) \ . \end{aligned}$$

∎

Soit l une opération linéaire définie par :

$$\begin{array}{rccc} l: & K & \rightarrow & K \\ & x & \mapsto & \lambda x \ , \end{array}$$

Nous pouvons représenter cette opération comme un produit dans l'espace vectoriel $\mathbb{F}_2^8$ (nous passons d'une multiplication dans le corps $\mathbb{F}_{2^8}$ vers une multiplication matricielle dans $\mathbb{F}_2^8$). En effet, puisque le groupe additif $\mathbb{F}_2^8$ et l'espace vectoriel $\mathbb{F}_2^8$ sont isomorphes, il existe pour chaque $\lambda \in K$ une matrice $L \in \mathbb{F}_2^{8\times 8}$, telle que $\lambda x \equiv \Psi(x)L$. La matrice L est conçue à partir de l'isomorphisme Ψ où la *ieme* ligne de la matrice correspond à $\Psi(\alpha^i \lambda)$.

Exemple 9
Calculons le produit $x\lambda$, avec $x = A2 = \alpha + \alpha^5 + \alpha^7$ et $\lambda = 02 = \alpha$:

$$\begin{aligned}(\alpha + \alpha^5 + \alpha^7)\alpha &= \alpha^2 + \alpha^6 + \alpha^8 \\ &= \alpha^2 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha + 1 \\ &= 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^6 \ .\end{aligned}$$

$$\underbrace{\begin{bmatrix}0&1&0&0&0&1&0&1\end{bmatrix}}_{\Psi(\alpha+\alpha^5+\alpha^7)} \underbrace{\begin{bmatrix}0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\\1&1&0&1&1&0&0&0\end{bmatrix}}_{L} = \begin{bmatrix}1&1&1&1&1&0&1&0\end{bmatrix}$$

$$\begin{aligned}&= \Psi(1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^6) \\ &= \Psi((\alpha + \alpha^5 + \alpha^7)\alpha) \ .\end{aligned}$$

∎

En général chaque opération linéaire de la forme : $l(x) = xL$ peut être transformée en une opération masquée : $l'(z) = zL'$, avec $z = \texttt{mask}(x)$. Dans le cas ODSM L' est une matrice constante définie par :

$$L' = G^\top (GG^\top)^{-1} LG \oplus H^\top (HH^\top)^{-1} H \ . \tag{1.14}$$

En effet, si : $z = \texttt{mask}(x) = xG + rH$, alors :

$$\begin{aligned}zL' &= \Big(xG \oplus rH\Big)\Big(G^\top (GG^\top)^{-1} LG \oplus H^\top (HH^\top)^{-1} H\Big) \\ &= \Big(x \underbrace{GG^\top (GG^\top)^{-1}}_{=I_n} LG\Big) \oplus \Big(x \underbrace{GH^\top}_{=0} (HH^\top)^{-1} H\Big) \\ &\quad \oplus \Big(r \underbrace{HG^\top}_{=0} (GG^\top)^{-1} LG\Big) \oplus \Big(r \underbrace{HH^\top (HH^\top)^{-1}}_{=I_n} H\Big) \\ &= xLG \oplus rH \\ &= \texttt{mask}(xL) \ .\end{aligned}$$

Il faut noter que les auteurs de ce masquage, proposent de garder intentionnellement le masque r identique après chaque opération, pour des raisons que nous allons évoquer ultérieurement. Mais cela n'empêche pas qu'il soit assez facile de changer le masque après chaque opération.

1.4.2.3 Les transformations non linéaires

Pour les opérations non-linéaires il existe deux approches, la première consiste à stocker en mémoire la table de substitution masquée (d'où l'intérêt de garder le

même masque tout au long du chiffrement ou déchiffrement), la seconde consiste à calculer cette transformation dans une zone sécurisée du composant.

$$\begin{aligned} S'(z) &= S(\texttt{unmask}(z))G \oplus zH^\top(HH^\top)^{-1}H \\ &= S(x)G \oplus rH \\ &= \texttt{mask}(S(x)) \ . \end{aligned}$$

Dans le chapitre 2 nous allons voir comment sécuriser la transformation `SubBytes` grâce au masquage DSM avec des opérations dans K. Cette solution est avantageuse pour les équipements ayant une mémoire de stockage limitée ou bien qui ne possèdent pas une zone de calcul sécurisée.

1.4.2.4 La détection des erreurs

Un grand avantage de ce schéma de masquage est qu'il permet de générer des masques aléatoires de longueur n à partir d'un aléa de taille beaucoup plus petite k. Ceci représente un gain important pour le temps d'exécution. Un autre point innovant à noter pour ce schéma ; c'est la détection des erreurs, ce qui permet de détecter certaines injections de fautes.
Soit $z = xG \oplus rH = \texttt{mask}(x)$ le masqué associé à x, et r le masque utilisé. Soit $e \in \mathbb{F}_2^n$ l'erreur injectée. D'après la définition 20 page 62, il existe un unique vecteur $u \in \mathbb{F}_2^k$ et un unique vecteur $v \in \mathbb{F}_2^{n-k}$ tels que $e = uG \oplus vH$. Notons : $z' = z \oplus e$ le mot obtenu après l'injection de faute. Nous avons :

$$\begin{aligned} z' &= z \oplus e \\ &= (x \oplus u)G \oplus (r \oplus v)H \ . \end{aligned}$$

Grâce à la projection de l'équation 1.9 page 64 nous pouvons extraire $r \oplus v$ et le comparer avec le masque utilisé r. Cependant, si l'erreur est tirée de manière aléatoire, alors il existe une probabilité de $\frac{1}{2^{n-k}}$ que v soit égal à zéro (et donc $v \oplus r = r$). Ainsi, la probabilité de détecter une attaque de type FIA vaut $1 - 2^{k-n}$. Bien que cette probabilité soit importante, les injections de fautes restent plus difficiles à réaliser en pratique. Expérimentalement, il est plus facile de produire des erreurs qui ont un poids de Hamming faible, donc pour contourner l'ODSM il faudra que l'erreur e appartienne au code $\mathcal{C}$, ce qui signifie que le nombre d'erreurs injectées doit être supérieur à la distance minimale de $\mathcal{C}$. Ceci explique l'intérêt de choisir un code dont la distance minimale est optimale.

1.4.2.5 Sécurité et performances

Pour ce schéma de masquage nous supposons que la donnée sensible dépend du message (ou la valeur intermédiaire de chaque tour) et de la clé (clé maître ou clé de session), l'attaquant doit donc recueillir une quantité suffisante de traces pour récupérer le secret.

Définition 23 (Transformée de Fourier)
Soit $f : \mathbb{F}_2^n \to \mathbb{R}$ une fonction "pseudo booléenne" (à valeurs réelles), la transformée de Fourier de f est définie par :

$$\widehat{f}(z) = \sum_{a \in \mathbb{F}_2^n} f(a)(-1)^{\langle a,z \rangle} .$$

Propriété 14
Soit $P : \mathbb{F}_2^n \to \mathbb{R}$ une fonction pseudo booléenne de degré numérique $d^\circ(P)$, alors :

$$\forall z \in \mathbb{F}_2^n, \quad \omega_H(z) > d^\circ(P) \implies \widehat{P}(z) = 0 .$$

Preuve :
Comme P est une fonction polynomiale, alors par définition elle s'écrit ainsi : $P(y_1, \ldots, y_n) = \sum_{I \in \mathbb{F}_2^n} a_I y^I$, avec $y^I = \prod_{i=1}^n y_i^{I_i}$ et $a_I \in \mathbb{R}$. Par linéarité de la transformée de Fourier, nous avons :

$$\widehat{P}(z) = \sum_{I \in \mathbb{F}_2^n} a_I \widehat{M_I}(z) ,$$

avec $M_I(y) = y^I$. Montrons que $\forall z \in \mathbb{F}_2^n, \omega_H(z) > \omega_H(I) \implies \widehat{M_I}(z) = 0$.
Soit $z \in \mathbb{F}_2^n$ tel que $\omega_H(z) > \omega_H(I)$, il existe au moins une position i telle que $z_i = 1$ et $I_i = 0$, nous pouvons donc supposer - sans perte de généralité - que cette position i égal à n (la dernière position). Notons $y = (y', y_n) \in \mathbb{F}_2^n$ avec $y' = (y_1, \ldots, y_{n-1}) \in \mathbb{F}_2^{n-1}$ et $y_n \in \mathbb{F}_2$. Nous avons :

$$\begin{aligned} \widehat{M_I}(z) &= \textstyle\sum_{y' \in \mathbb{F}_2^{n-1}} \sum_{y_n \in \mathbb{F}_2} y^I (-1)^{\langle y,z \rangle} \\ &= \textstyle\sum_{y' \in \mathbb{F}_2^{n-1}} (y', 0)^I (-1)^{\langle y', z_1, \ldots, z_{n-1} \rangle} (1 + (-1)) \\ &= 0 . \end{aligned}$$

Donc nous avons bien : $\widehat{P}(z) = \sum_{I \in \mathbb{F}_2^n} a_I \times 0 = 0$. ■

Théorème 2 (L'ordre du masque ODSM [BCC+14])
ODSM permet de protéger contre les attaques de type SCA sur une variable pour un ordre de masquage d tel que $d < d_{\mathcal{C}}$, où $d_{\mathcal{C}}$ est la distance minimale de $\mathcal{C}$.

Preuve :[BCC+14]
Soient :
— $\Psi : \mathbb{F}_2^k \to \mathbb{F}_2^n$ la fonction d'encodage de $\mathcal{C}$ (i.e. $\Psi(x) = xG$),

— $\Phi : \mathbb{F}_2^n \to \mathbb{R}$ une fonction pseudo-booléenne d'un degré numérique donné j. Ce degré représente le nombre de bits de z exploités par l'attaquant, en d'autres termes la fonction Φ peut modéliser le produit de j bits de z.
— et $f : \mathbb{F}_2^n \to \mathbb{F}_2$ la fonction indicatrice de $\mathcal{D}$:

$$f(d) = \begin{cases} 1 & \text{si } d \in \mathcal{D} \quad (\text{i.e. } \exists y \in \mathbb{F}_2^{n-k} \quad | \quad yH = d) \\ 0 & \text{sinon .} \end{cases}$$

Soient X et D deux variables aléatoires uniformément distribuées dans $\mathbb{F}_2^k$ et $\mathcal{D}$ respectivement.

Proposition 3 (Condition de sécurité d'ordre j sur l'encodage des masques)
Pour toute fonction pseudo-booléenne Φ de degré numérique inférieur ou égal à j, la fuite $\Phi(\Psi(X) \oplus D)$ résiste aux attaques SCA sur une variable pour un ordre j si $\mathcal{D}$ est un code de distance dual égal à $j+1$ ($d_{\mathcal{D}}^{\perp} = j+1$).

Cette proposition est une simple reformulation du théorème précédent, et pour la démontrer il suffit de montrer qu'il n'existe aucune dépendance entre $x \in \mathbb{F}_2^k$ et $\mathbb{E}[\Phi(\Psi(X) \oplus D)|X = x]$, en d'autres termes, il faut montrer que :

$$\mathtt{Var}\Big[\mathbb{E}\big[\Phi(\Psi(X) \oplus D)|X\big]\Big] = 0 \ .$$

Nous avons :

$$\begin{aligned} \mathbb{E}\big[\Phi(\Psi(X) + D) \mid X = x\big] &= \sum_{d \in \mathcal{D}} \tfrac{1}{|\mathcal{D}|} \Phi(\Psi(x) \oplus d) \\ &= 2^{-(n-k)} \sum_{d \in \mathbb{F}_2^n} f(d)\Phi(\Psi(x) \oplus d) \\ &= 2^{-(n-k)} (f \otimes \Phi)(\Psi(x)) \ . \end{aligned}$$

Ainsi, il faut montrer que la valeur $(f \otimes \Phi)(\Psi(x))$ ne dépend pas de x.
Soit $g : \mathbb{F}_2^n \to \mathbb{R}$ une fonction pseudo booléenne, et $\widehat{g}$ sa transformée de Fourier. Nous avons : g est constante $\iff \forall z \neq 0, \widehat{g}(z) = 0$. Notons : $g = f \otimes \Phi$, la transformée de Fourier transforme un produit de convolution en un produit dans $\mathbb{R}$, ainsi nous avons : $\widehat{g}(z) = \widehat{f \otimes \Phi}(z) = \widehat{f}(z)\widehat{\Phi}(z)$. Il reste donc à démontrer que :

$$\widehat{f}\widehat{\Phi} = 0. \tag{1.15}$$

Comme Φ dans la proposition 3 est de degré numérique j, alors d'après la propriété 14 nous avons : pour tout z dans $\mathbb{F}_2^n$ tel que $\omega_H(z) > j$ alors $\widehat{\Phi}(z) = 0$. En d'autres termes, ce masquage protège contre la SCA pour un ordre j si $\forall z \in \mathbb{F}_2^n, 0 < \omega_H(z) \leqslant j, \widehat{f}(z) = 0$, ce qui signifie également que la distance minimale du dual de $\mathcal{D}$ est $d_{\mathcal{D}^\perp} = d_{\mathcal{C}} = j+1$ (puisque $\mathcal{C}$ et $\mathcal{D}$ sont des codes LCD). ■

1.4.3 Exemple

Comme la plupart des composants électroniques opèrent sur des octets (par exemple les cartes à puces, ou les TPM), il est préférable pour ce genre d'architectures de prendre une longueur de code n multiple de $k = 8$. Dans [CDD+15] Carlet *et al.* ont présenté une méthode efficace pour construire des codes optimaux pour ce type de masquage.
Soit $\mathcal{C}$ un code LCD de paramètres $[16, 8, 5]$, tels que G est sa matrice génératrice et H sa matrice de parité (donc la matrice génératrice de son supplémentaire $\mathcal{D}$), avec :

$$G = \begin{bmatrix} 1&0&0&0&0&0&0&0 & 1&0&0&1&1&1&1&0 \\ 0&1&0&0&0&0&0&0 & 0&1&0&0&1&1&1&1 \\ 0&0&1&0&0&0&0&0 & 1&1&0&0&1&1&0&0 \\ 0&0&0&1&0&0&0&0 & 0&1&1&0&0&1&1&0 \\ 0&0&0&0&1&0&0&0 & 0&0&1&1&0&0&1&1 \\ 0&0&0&0&0&1&0&0 & 1&1&1&1&0&0&1&0 \\ 0&0&0&0&0&0&1&0 & 0&1&1&1&1&0&0&1 \\ 0&0&0&0&0&0&0&1 & 1&1&0&1&0&1&1&1 \end{bmatrix} , \qquad H = \begin{bmatrix} 1&0&1&0&0&1&0&1 & 1&0&0&0&0&0&0&0 \\ 0&1&1&1&0&1&1&1 & 0&1&0&0&0&0&0&0 \\ 0&0&0&1&1&1&1&0 & 0&0&1&0&0&0&0&0 \\ 1&0&0&0&1&1&1&1 & 0&0&0&1&0&0&0&0 \\ 1&1&1&0&0&0&1&0 & 0&0&0&0&1&0&0&0 \\ 1&1&1&1&0&0&0&1 & 0&0&0&0&0&1&0&0 \\ 1&1&0&1&1&1&0&1 & 0&0&0&0&0&0&1&0 \\ 0&1&0&0&1&0&1&1 & 0&0&0&0&0&0&0&1 \end{bmatrix} .$$

Soient $x = (1, 0, 0, 1, 1, 0, 1, 1) = \Psi(9B)$ une donnée sensible et $r = (0, 1, 0, 1, 1, 1, 0, 1) = \Psi(5D)$ le masque à utiliser. Après l'opération `mask`, le mot masqué associé à x sera donc :

$$z = xG \oplus rH = (0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0) = \Psi(3B38) .$$

Pour extraire x ou r à partir de z (les équations 1.8 et 1.9), nous avons :

$$\begin{aligned} zG^\top(GG^\top)^{-1} &= (1, 0, 0, 1, 1, 0, 1, 1) = \Psi(9B) , \\ zH^\top(HH^\top)^{-1} &= (0, 1, 0, 1, 1, 1, 0, 1) = \Psi(5D) . \end{aligned}$$

Il faut noter que les matrices $G^\top(GG^\top)^{-1}$ et $H^\top(HH^\top)^{-1}$ sont précalculées :

$$G^\top(GG^\top)^{-1} = \begin{bmatrix} 0&0&1&1&1&0&0&1 \\ 0&0&1&0&0&1&0&1 \\ 1&1&1&1&1&0&1&0 \\ 1&0&1&0&1&1&0&0 \\ 1&0&1&1&1&1&1&0 \\ 0&1&0&1&1&1&1&1 \\ 0&0&1&0&1&1&1&1 \\ 1&1&0&0&0&1&1&0 \\ \\ 0&1&0&1&1&0&1&0 \\ 1&1&0&0&0&1&0&1 \\ 0&1&1&0&0&0&1&0 \\ 0&0&1&1&0&0&0&1 \\ 1&1&0&0&1&0&0&1 \\ 1&0&0&0&1&1&0&0 \\ 1&0&0&1&0&1&1&1 \\ 0&1&1&1&0&0&1&0 \end{bmatrix} , \qquad H^\top(HH^\top)^{-1} = \begin{bmatrix} 0&1&0&0&1&1&1&0 \\ 1&1&1&0&1&0&0&1 \\ 0&0&1&1&0&0&0&1 \\ 1&0&0&1&0&0&1&1 \\ 1&0&0&0&1&1&0&0 \\ 0&1&0&0&0&1&1&0 \\ 1&0&1&0&0&0&1&1 \\ 0&1&0&1&1&0&1&0 \\ \\ 1&1&1&0&0&0&1&1 \\ 1&0&1&1&0&1&0&0 \\ 1&1&0&1&1&0&1&0 \\ 0&1&1&0&1&1&0&1 \\ 0&0&1&1&1&1&0&1 \\ 0&1&0&1&1&0&1&1 \\ 1&0&1&0&0&1&1&0 \\ 1&0&0&1&1&1&0&1 \end{bmatrix} .$$

Calculer MixColumns :

Dans cet exemple nous allons calculer la fonction f de l'équation 1.13 qui représente le fonctionnement de la transformation `MixColumns` :

$$f(x_1, x_2, x_3, x_4) = 02x_1 \oplus x_2 \oplus x_3 \oplus 03x_4 .$$

Comme nous l'avons vu dans la partie (opération (ii)) ; les fonctions : $x \mapsto 02x$ et $x \mapsto 03x$ équivalent respectivement : $\Psi(x) \mapsto \Psi(x)L$ et $\Psi(x) \mapsto \Psi(x)M$ telles que :

$$L = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&1&0&1&1&0&0&0 \end{bmatrix}, \qquad M = \begin{bmatrix} 1&1&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 0&0&0&1&1&0&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&1&1 \\ 1&1&0&1&1&0&0&1 \end{bmatrix}.$$

Soient z_1, z_2, z_3, z_4 les masqués associés à x_1, x_2, x_3, x_4 respectivement, et L', M' les matrices associées à L et M respectivement, et qui sont calculées grâce à l'équation 1.14. Nous pouvons construire la fonction alternative à f ainsi :

$$f(z_1, z_2, z_3, z_4) = z_1L' \oplus z_2 \oplus z_3 \oplus z_4M' \ .$$

Avec :

$$L' = \begin{bmatrix} 0&1&1&1&1&1&0&1 & 0&1&1&0&1&1&0&1 \\ 1&0&1&0&1&1&1&1 & 0&1&1&1&0&0&1&0 \\ 1&0&1&0&0&1&1&1 & 1&1&0&0&0&0&1&0 \\ 1&1&1&0&1&0&1&0 & 0&0&1&1&0&0&0&1 \\ 1&1&1&0&1&0&0&1 & 1&1&0&0&1&0&1&0 \\ 1&0&1&0&1&1&0&0 & 0&1&1&0&0&0&0&1 \\ 1&1&1&0&0&0&1&0 & 0&0&0&1&1&1&0&1 \\ 1&0&1&0&0&1&0&0 & 0&1&1&1&0&1&1&1 \\ \\ 0&1&1&1&0&1&1&1 & 0&0&1&1&1&0&0&1 \\ 0&1&1&1&1&1&1&1 & 1&1&0&0&1&0&1&0 \\ 0&1&0&1&0&0&1&1 & 1&0&1&0&0&1&1&1 \\ 1&1&1&1&0&0&0&1 & 1&0&1&1&1&1&0&0 \\ 0&1&1&1&0&1&0&1 & 1&1&0&0&1&0&0&0 \\ 1&1&0&0&1&0&1&0 & 1&0&0&1&1&1&1&1 \\ 0&0&0&0&0&1&0&0 & 1&1&1&1&0&0&0&0 \\ 0&1&0&0&1&0&1&1 & 1&1&0&1&0&0&1&1 \end{bmatrix}, \qquad M' = \begin{bmatrix} 0&1&0&0&0&1&0&0 & 0&0&1&0&0&0&1&1 \\ 1&0&0&0&1&0&1&0 & 1&0&0&1&1&0&1&1 \\ 0&1&0&1&1&1&0&1 & 1&1&1&1&0&0&1&1 \\ 0&1&0&0&0&1&1&0 & 1&0&1&0&0&0&1&0 \\ 0&1&0&1&0&1&1&1 & 0&1&0&0&0&1&1&0 \\ 1&1&1&1&0&0&1&1 & 0&0&1&0&0&1&1&1 \\ 1&1&0&0&1&1&0&1 & 1&0&1&1&1&1&1&0 \\ 0&1&1&0&0&0&1&0 & 0&0&1&0&1&1&0&1 \\ \\ 0&0&1&0&1&1&0&1 & 0&1&0&1&1&0&1&0 \\ 1&0&1&1&1&0&1&0 & 0&0&1&1&1&1&1&0 \\ 0&0&1&1&0&0&0&1 & 0&1&0&1&1&1&0&1 \\ 1&1&0&0&0&0&0&0 & 1&1&0&0&0&0&0&1 \\ 1&0&1&1&1&1&0&0 & 1&1&1&1&1&1&0&1 \\ 0&1&0&0&0&1&1&0 & 1&1&0&0&0&0&0&0 \\ 1&0&0&1&0&0&1&1 & 0&1&0&1&0&1&0&0 \\ 0&0&1&1&1&0&0&1 & 0&1&0&0&1&1&1&1 \end{bmatrix}.$$

1.5 Conclusion

Dans ce chapitre nous avons présenté une variété de schémas de masquage les plus connus. Nous avons essayé de présenter des méthodes qui sont basées sur différents concepts. Chaque schéma possède ses propres avantages et faiblesses, il y en a qui s'adaptent mieux à certains besoins mais pas d'autres et tout dépend du support qui va accueillir cette solution. Pour conclure, nous avons résumé un bon schéma de masquage en quatre points essentiels :

1. Indéterministe : Pour un niveau de sécurité plus élevé, il est important que le schéma de masquage soit indéterministe, c'est-à-dire que la donnée sensible soit combinée avec une valeur aléatoire dans le processus du masquage ;
2. Homomorphe : L'opération de masquage doit pouvoir masquer les deux opérations de base (addition et multiplication). En d'autres termes, il faudra construire des opérations alternatives à l'addition et à la multiplication qui coopèrent avec la nouvelle structure de la donnée traitée ;
3. Léger : La complexité algorithmique représente aussi un défi majeur pour le concepteur. Il est important de prendre en compte la mémoire très limitée des composants électroniques avant de construire une fonction de masquage ;
4. Structuré : Le mot masqué doit avoir une structure d'un code correcteur afin de protéger contre FIA.

2 | Le masquage par somme directe sur un corps fini

Sommaire

2.1 Introduction

2.1.1 Objectifs

Nous remarquons que les schémas de masquage que nous avons présentés dans le chapitre précédent ne protègent pas tous contre les deux attaques SCA et FIA simultanément. Le masquage booléen, le masquage basé sur le partage du secret et le masquage basé sur le produit scalaire permettent tous de masquer l'addition et la multiplication et protéger ainsi contre SCA. D'un autre côté, le masquage ODSM propose une solution pour masquer uniquement l'addition, et en même temps détecter certaines injections des erreurs.
Pour surmonter tout ça, nous allons présenter dans ce chapitre un schéma de masquage qui pourra accomplir les tâches suivantes [CDGT18] :

- Masquer l'information sensible de manière efficace, il faut que la fonction de masquage soit non-déterministe, et qu'elle permette de masquer les deux opérations d'addition et de multiplication ;
- Détecter les erreurs potentielles et éventuellement les corriger si le besoin l'impose ;
- Réduire le plus possible la complexité algorithmique en termes de temps de calcul (la génération de l'aléa, et la quantité d'opérations) et de l'espace mémoire (l'évolution de la longueur du masqué en termes de l'ordre désiré).

Comme nous l'avons constaté dans le chapitre précédent, pour masquer l'AES, il suffit de réussir à masquer l'addition et la multiplication dans le corps $\mathbb{F}_{256}$. Donc pour atteindre les trois objectifs ci-dessus il faudra construire les trois algorithmes suivantes :

1. $\texttt{mask} : x \mapsto z$ tel que z soit un mot de code ;
2. $\texttt{Add} : \texttt{mask}(x), \texttt{mask}(y) \mapsto \texttt{mask}(x + y)$;
3. $\texttt{Mult} : \texttt{mask}(x), \texttt{mask}(y) \mapsto \texttt{mask}(xy)$.

La solution que nous proposons dans ce chapitre et qui permet de réaliser ces objectifs est basée sur la même approche que ODSM. Cependant, dans ce schéma nous avons choisi de travailler dans le corps fini $\mathbb{F}_{256}$ qui est le corps de base de toutes les opérations qui composent l'AES. Ce choix nous permet d'avoir plus de flexibilité pour manipuler les polynômes, et donc plus de maîtrise pour atteindre l'objectif final.

2.1.2 Vocabulaire

Pour commencer, nous allons établir un vocabulaire qui sera utilisé tout au long de ce chapitre.

Soit $K = \mathbb{F}_{256} \equiv \mathbb{F}_2[\alpha]/(\alpha^8+\alpha^4+\alpha^3+\alpha+1)$ un corps fini de 256 éléments. Chaque élément de ce corps est une classe d'équivalence représentée par un polynôme de degré au plus 7. Soient $\mathcal{C}$ et $\mathcal{D}$ deux sous-espaces vectoriels de dimension 1 dans K^n (contrairement à la construction du chapitre précédent ; il s'agit ici de $\mathbb{F}_{256}^n$ et non pas $\mathbb{F}_2^n$). Pour faire la distinction entre les éléments de K et de K^n, nous allons exprimer les éléments de l'espace vectoriel K^n avec une flèche en dessus. Soient $\vec{g}, \vec{h} \in K^n \backslash \{0\}$ des générateurs de $\mathcal{C}$ et $\mathcal{D}$ respectivement tels que $\langle \vec{g}, \vec{h} \rangle = 0$, $\langle \vec{g}, \vec{g} \rangle \neq 0$ et $\langle \vec{h}, \vec{h} \rangle \neq 0$. Nous avons :

$$\mathcal{C} = \{x \cdot \vec{g} \mid x \in K\} \ ,$$

et :

$$\mathcal{D} = \{x \cdot \vec{h} \mid x \in K\} \ ,$$

où " $\cdot$ " désigne un produit entre un scalaire et un vecteur (exemple : $x \cdot \vec{g} = (xg_1, \ldots, xg_n) \in K^n$). Notons $\mathcal{K}$ le sous-espace vectoriel de K^n formé à partir des éléments de $\mathcal{C}$ et $\mathcal{D}$:

$$\mathcal{K} = \{c \oplus d \mid c \in \mathcal{C}, d \in \mathcal{D}\} \subseteq K^n \ .$$

Remarque 1
Pour $n = 2$, le code $\mathcal{C}$ est un code LCD, et dans ce cas $\mathcal{K} = K^n$.

Le masquage PDSM (*Polynomial Direct Sum Masking*) [CDGT18] consiste à additionner le mot de code associé à la donnée sensible avec un ou plusieurs mots aléatoires dans le dual :

$$\texttt{mask}(x) = x \cdot \vec{g} \oplus r \cdot \vec{h} \ \in \mathcal{K} \ . \tag{2.1}$$

Algorithm 20 $\texttt{mask}(x)$ — Complexité : $2n$ multiplications et n additions

1: **entrée :** $x \in K$
2: **sortie :** $\texttt{mask}(x) \in \mathcal{K}$
3: $r \xleftarrow{\$} K$ — le masque
4: $\vec{z} \leftarrow \vec{0} \in \mathcal{K}$
5: **pour** $i = 1$ **jusqu'à** n **faire :**
6: $\quad z_i \leftarrow xg_i \oplus rh_i$
7: **renvoyer** $\vec{z}$

Pour extraire l'information sensible x à partir d'un masqué $\vec{z} = \texttt{mask}(x)$, il suffit de multiplier ce dernier par $\vec{g}$ et par le scalaire $g' = \langle \vec{g}, \vec{g} \rangle^{-1}$:

$$\texttt{unmask}(\vec{z}) = \langle \vec{z}, \vec{g} \rangle \langle \vec{g}, \vec{g} \rangle^{-1} \ . \tag{2.2}$$

Preuve : Cette opération se déduit directement de l'équation 1.8 (par symétrie nous pouvons également extraire r grâce à l'équation 1.9). Nous avons :

$$\begin{aligned}\langle \vec{z}, \vec{g} \rangle \langle \vec{g}, \vec{g} \rangle^{-1} &= \langle x \cdot \vec{g} \oplus r \cdot \vec{h}, \vec{g} \rangle \langle \vec{g}, \vec{g} \rangle^{-1} \\ &= \Big(x\langle \vec{g}, \vec{g} \rangle \oplus \underbrace{r\langle \vec{h}, \vec{g} \rangle}_{=0} \Big) \langle \vec{g}, \vec{g} \rangle^{-1} \\ &= x\langle \vec{g}, \vec{g} \rangle \langle \vec{g}, \vec{g} \rangle^{-1} \\ &= x \ .\end{aligned}$$

■

Algorithm 21 `Unmask`$(\vec{z})$ Complexité : $(n+1)$ multiplications et n additions

1: **entrée :** $\vec{z} = \texttt{mask}(x) \in \mathcal{K}$
2: **sortie :** $x \in K$
3: $x \leftarrow 0 \in K$
4: **pour** $i = 1$ **jusqu'à** n **faire :**
5: $\quad x \leftarrow x \oplus z_i g_i$
6: $x \leftarrow x\langle \vec{g}, \vec{g} \rangle^{-1}$ Nous supposons que la valeur $\langle \vec{g}, \vec{g} \rangle^{-1}$ est précalculée
7: **renvoyer** x

La figure 2.1 résume l'ensemble des éléments qui sont utilisés pour ce schéma de masquage.

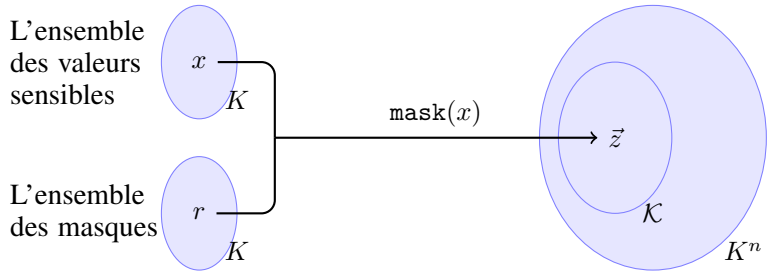


FIGURE 2.1 – Représentation graphique de l'ensemble des éléments du masquage DSM.

2.2 L'addition et la multiplication

2.2.1 L'addition

Cette méthode de masquage est une fonction linéaire par rapport à l'addition. Considérons `mask`(x) et `mask`(y) la valeur masquée d'une donnée sensible x et y

respectivement, la valeur masquée de $x \oplus y$ correspond à :

$$\texttt{mask}(x) \oplus \texttt{mask}(y) = \texttt{mask}(x \oplus y) \ .$$

Algorithm 22 $\texttt{Add}(\vec{z}, \vec{z}')$ — Complexité : n additions

1: **entrée :** $\vec{z} = \texttt{mask}(x), \vec{z}' = \texttt{mask}(y) \in \mathcal{K}$
2: **sortie :** $\texttt{mask}(x \oplus y) \in \mathcal{K}$
3: $\vec{u} \leftarrow \vec{0} \in \mathcal{K}$
4: **pour** $i = 1$ **jusqu'à** n **faire :**
5: $\quad u_i \leftarrow z_i \oplus z'_i$
6: **renvoyer** $\vec{u}$

En ce qui concerne la multiplication, nous allons distinguer deux types de multiplications.

2.2.2 La multiplication par une valeur publique

Le cas le plus simple est la multiplication entre une valeur masquée et une valeur non masquée. Ce cas permet de gagner en complexité en évitant de masquer les valeurs publiques inutilement (exemple : la matrice de `MixColumns` et les coefficients de la transformation `SubBytes`). Ainsi, pour masquer le produit λx à partir d'une donnée publique λ et le masqué de x, nous procédons ainsi :

$$\begin{aligned} \lambda \cdot \texttt{mask}(x) &= \lambda \cdot \left(x \cdot \vec{g} \oplus r \cdot h \right) \\ &= \lambda x \cdot \vec{g} \oplus \lambda r \cdot \vec{h} \\ &= \texttt{mask}(\lambda x) \ . \end{aligned}$$

Algorithm 23 $\texttt{mult1}(\lambda, \vec{z})$ — Complexité : n multiplications

1: **entrée :** $\lambda \in K$ et $\vec{z} = \texttt{mask}(x) \in \mathcal{K}$
2: **sortie :** $\texttt{mask}(\lambda x) \in \mathcal{K}$
3: $\vec{u} \leftarrow \vec{0} \in \mathcal{K}$
4: **pour** $i = 1$ **jusqu'à** n **faire :**
5: $\quad u_i \leftarrow \lambda z_i$
6: **renvoyer** $\vec{u}$

2.2.3 La multiplication entre deux valeurs sensibles

Le deuxième cas est la multiplication entre deux valeurs masquées, nous avons :

$$\begin{aligned}\texttt{mask}(xy) &= \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big[\texttt{mask}\Big(r_t + \langle \texttt{mask}(x), \texttt{mask}(y) \rangle \Big) \\ &\quad - \Big(r_t + \langle \vec{h}, \vec{h} \rangle^{-1} \langle \texttt{mask}(x), \vec{h} \rangle \langle \texttt{mask}(y), \vec{h} \rangle \Big) \cdot \vec{g} \Big] \ ,\end{aligned}$$

avec r_t un masque intermédiaire généré pendant le calcul pour sécuriser le produit scalaire $\langle \texttt{mask}(x), \texttt{mask}(y) \rangle$. En effet, ce produit (comme indiqué dans la preuve ci-dessous) permet d'extraire la valeur sensible xy.

Preuve : Nous avons :

$$\begin{aligned}r_t \oplus \langle \texttt{mask}(x), \texttt{mask}(y) \rangle &= r_t \oplus \langle x \cdot \vec{g} \oplus r \cdot \vec{h}, y \cdot \vec{g} \oplus r' \cdot \vec{h} \rangle \\ &= r_t \oplus \langle x \cdot \vec{g}, y \cdot \vec{g} \rangle \oplus \langle x \cdot \vec{g}, r' \cdot \vec{h} \rangle \oplus \langle r \cdot \vec{h}, y \cdot \vec{g} \rangle \oplus \langle r \cdot \vec{h}, r' \cdot \vec{h} \rangle \\ &= r_t \oplus xy \langle \vec{g}, \vec{g} \rangle \oplus \underbrace{xr' \langle \vec{g}, \vec{h} \rangle \oplus yr \langle \vec{h}, \vec{g} \rangle}_{=0} \oplus rr' \langle \vec{h}, \vec{h} \rangle \\ &= r_t \oplus xy \langle \vec{g}, \vec{g} \rangle \oplus rr' \langle \vec{h}, \vec{h} \rangle \ .\end{aligned}$$

$$\begin{aligned}\langle \texttt{mask}(x), \vec{h} \rangle &= \langle x \cdot \vec{g} \oplus r \cdot \vec{h}, \vec{h} \rangle \\ &= x \langle \vec{g}, \vec{h} \rangle \oplus r \langle \vec{h}, \vec{h} \rangle \\ &= r \langle \vec{h}, \vec{h} \rangle \ .\end{aligned}$$

Nous obtenons donc :

$$\langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big[\texttt{mask}\Big(r_t \oplus \langle \texttt{mask}(x), \texttt{mask}(y) \rangle \Big) - \Big(r_t \oplus \langle \vec{h}, \vec{h} \rangle^{-1} \langle \texttt{mask}(x), \vec{h} \rangle \langle \texttt{mask}(y), \vec{h} \rangle \Big) \cdot \vec{g} \Big]$$

$$\begin{aligned}&= \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big[\Big(r_t \oplus xy \langle \vec{g}, \vec{g} \rangle \oplus rr' \langle \vec{h}, \vec{h} \rangle \Big) \cdot \vec{g} \oplus r'' \cdot \vec{h} - \Big(r_t \oplus rr' \langle \vec{h}, \vec{h} \rangle \Big) \cdot \vec{g} \Big] \\ &= \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big[xy \langle \vec{g}, \vec{g} \rangle \cdot \vec{g} \oplus r'' \cdot \vec{h} \Big] \\ &= xy \cdot \vec{g} \oplus r'' \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \vec{h} \\ &= \texttt{mask}(xy) \ .\end{aligned}$$

■

Algorithm 24 $\texttt{Mult2}(\vec{z}, \vec{z}')$ Complexité : $7n + 2$ multiplications et $4n + 1$ additions

entrée : $\vec{z} = \texttt{mask}(x), \vec{z}' = \texttt{mask}(y) \in \mathcal{K}$
sortie : $\texttt{mask}(xy) \in \mathcal{K}$
$r_t \xleftarrow{\$} K^*$ un masque temporaire
$a \leftarrow r_t$
$b \leftarrow 0 \in K$
$c \leftarrow 0 \in K$
$\vec{u} \leftarrow \vec{0} \in \mathcal{K}$
pour $i = 1$ **jusqu'à** n **faire :**
    $a \leftarrow a \oplus z_i z'_i$
    $b \leftarrow b \oplus z_i h_i$
    $c \leftarrow c \oplus z'_i h_i$
$\vec{u} \leftarrow \langle \vec{g}, \vec{g} \rangle^{-1} \left(\texttt{mask}(a) \oplus \texttt{Mult1}(r_t \oplus bc \langle \vec{h}, \vec{h} \rangle^{-1}, \vec{g}) \right)$
renvoyer $\vec{u}$

Nous supposons que les valeurs $\langle \vec{g}, \vec{g} \rangle^{-1}$ et $\langle \vec{h}, \vec{h} \rangle^{-1}$ sont précalculées.

2.3 Sécurité et performances

Propriété 15
Soient $\vec{g}$ et $\vec{h}$ deux vecteurs dans K^n tels que $\langle \vec{g}, \vec{h} \rangle = 0$. Notons $\mathcal{C} = \{x \cdot \vec{g} \mid \forall x \in K\}$ et $\mathcal{D} = \{x \cdot \vec{h} \mid \forall x \in K\}$ les codes associés à $\vec{g}$ et $\vec{h}$ respectivement. Les deux conditions $\langle \vec{g}, \vec{g} \rangle \neq 0$ et $\langle \vec{h}, \vec{h} \rangle \neq 0$ impliquent :

— $\vec{g}$ et $\vec{h}$ sont linéairement indépendants (i.e. $\forall \lambda \in K \ \lambda \cdot \vec{g} \neq \vec{h}$) ;
— $\mathcal{C} \cap \mathcal{D} = \{\vec{0}\}$;
— $\forall \vec{z} \in \mathcal{K}, \ \exists ! (x, y) \in K^2$ tels que $\vec{z} = x \cdot \vec{g} \oplus y \cdot \vec{h}$.

Preuve :

— Pour la première propriété, supposons qu'il existe $\lambda \in K$ tel que : $\lambda \cdot \vec{g} = \vec{h}$, alors : $\lambda \langle \vec{g}, \vec{h} \rangle = \langle \lambda \cdot \vec{g}, \vec{h} \rangle = \langle \vec{h}, \vec{h} \rangle = 0$. La réciproque se démontre de la même façon.
— La deuxième propriété se déduit directement de la première, les deux générateurs $\vec{g}$ et $\vec{h}$ sont linéairement indépendants, alors le seul élément en commun entre les deux sous-espace vectoriels générés par $\vec{g}$ et $\vec{h}$ est l'élément neutre 0.
— La troisième propriété est une conséquence du théorème du rang : $dim(\mathcal{K}) = dim(\mathcal{C}) + dim(\mathcal{D})$.

■

Définition 24 (matrice génératrice)
Soit $\mathcal{C}$ le code décrit dans la propriété 15, la matrice génératrice de $\mathcal{C}$ est définie par : $G \in \mathbb{F}_2^{8\times 8n}$ telle que la i*eme* ligne de cette matrice correspond à la représentation binaire par Ψ (proposition 2) de $\alpha^i \cdot \vec{g}$ (i.e. $G_i = \Psi(\alpha^i \cdot \vec{g})$).

Exemple 10
Soient $\vec{g} = (1, 1, \alpha^3 + \alpha^4 + \alpha^7) \in K^3$ avec $K = \mathbb{F}_2[\alpha]/\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$, et $\mathcal{C}$ le code généré par $\vec{g}$. La matrice génératrice de $\mathcal{C}$ est définie par :

$$G = \begin{bmatrix}
1\,0\,0\,0\,0\,0\,0\,0 & 1\,0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,1\,1\,0\,0\,1 \\
0\,1\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,0\,0\,0 & 1\,1\,0\,1\,0\,1\,0\,0 \\
0\,0\,1\,0\,0\,0\,0\,0 & 0\,0\,1\,0\,0\,0\,0\,0 & 0\,1\,1\,0\,1\,0\,1\,0 \\
0\,0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,0\,0\,0 & 0\,0\,1\,1\,0\,1\,0\,1 \\
0\,0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0\,0 & 1\,1\,0\,0\,0\,0\,1\,0 \\
0\,0\,0\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,1\,0\,0 & 0\,1\,1\,0\,0\,0\,0\,1 \\
0\,0\,0\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,1\,0 & 1\,1\,1\,0\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,1 & 0\,0\,0\,0\,0\,0\,0\,1 & 0\,1\,1\,1\,0\,1\,0\,0
\end{bmatrix} .$$

Par exemple la troisième ligne de G correspond à :

$$\begin{aligned}
\Psi(\alpha^3 \cdot \vec{g}) &= \Psi(\alpha^3, \alpha^3, \alpha^6 + \alpha^7 + \alpha^{10}) \\
&= \Psi(\alpha^3, \alpha^3, \alpha^6 + \alpha^7 + \alpha^2\alpha^8) \\
&= \Psi(\alpha^3, \alpha^3, \alpha^6 + \alpha^7 + \alpha^2(\alpha^4 + \alpha^3 + \alpha + 1)) \\
&= \Psi(\alpha^3, \alpha^3, \alpha^6 + \alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + \alpha^2) \\
&= \Psi(\alpha^3, \alpha^3, \alpha^2 + \alpha^3 + \alpha^5 + \alpha^7) \\
&= (0\,0\,0\,1\,0\,0\,0\,0 \;\; 0\,0\,0\,1\,0\,0\,0\,0 \;\; 0\,0\,1\,1\,0\,1\,0\,1) .
\end{aligned}$$

▮

Définition 25 (La distance minimale)
Soit $\mathcal{C} = \{x \cdot \vec{g} \mid \forall x \in K\}$ un code généré par un vecteur non nul $\vec{g} \in K^n$. Nous définissons la distance minimale de $\mathcal{C}$ par :

$$d_{\mathcal{C}} = min\{\omega_H(\Psi(v)) \mid v \in \mathcal{C}^*\} ,$$

avec ω_H désignant le poids de Hamming.

Théorème 3
Le masquage de l'équation 2.1 permet de protéger contre les attaques de type SCA sur une variable jusqu'à l'ordre $d_{\mathcal{D}^\perp} - 1$, où $d_{\mathcal{D}^\perp} - 1$ est la distance minimale du dual de $\mathcal{D}$.

Preuve : Comme dans ce qui précède, soient $\vec{g}, \vec{h} \in K^n$ deux générateurs de $\mathcal{C}$ et $\mathcal{D}$ respectivement, tels que : $\langle \vec{g}, \vec{h} \rangle = 0$, $\langle \vec{g}, \vec{g} \rangle \neq 0$ et $\langle \vec{h}, \vec{h} \rangle \neq 0$. L'opération de masquage qui consiste à calculer $\vec{z} = x \cdot \vec{g} + r \cdot \vec{h}$, pour une donnée sensible x et

un masque aléatoire r, équivaut à $z' = \Psi(x)G + \Psi(r)H = \Psi(z)$, avec G et H deux matrices génératrices de $\mathcal{C}$ et $\mathcal{D}$ respectivement. On observe ainsi que cette opération de masquage fait partie de la famille DSM. Le théorème 2 page 69 prouve que la dernière opération de masquage ($z' = xG + rH$) peut être attaquée par SCA monovarié d'ordre supérieur seulement à l'ordre $j \geqslant d_{\mathcal{D}^\perp}$ (dans le cas d'ODSM $\mathcal{D}^\perp = \mathcal{C}$). ■

Le tableau 2.1 résume la complexité algorithmique de chaque algorithme présenté, cette complexité est calculée en fonction de la taille du masque et du nombre d'opérations (addition et multiplication) effectuées.

	nb. multiplications	nb. d'additions
`mask`	$2n$	n
`unmask`	$n+1$	n
`Add`	0	n
`Mult1`	n	0
`Mult2`	$7n+2$	$4n+1$

TABLE 2.1 – Une vision globale sur la complexité algorithmique de chaque fonction en terme de nombre d'opérations.

La complexité algorithmique du masquage dépend souvent de la complexité de l'algorithme de multiplication (`Mult2`). En effet, c'est toujours l'opération la plus gourmande en terme d'opérations, cette complexité est exprimée en fonction de la longueur de la valeur masquée et du nombre d'opérations dans le corps fini $\mathbb{F}_{2^8}$. Dans le tableau 2.2 nous présentons un comparatif des méthodes de masquage que nous avons présenté en terme de complexité, longueur, et de caractéristiques.

	BM	SSM	IPM	ODSM	PDSM
L'addition	Oui	Oui	Oui	Oui	Oui
La multiplication	Oui	Oui	Oui	Non	Oui
Le nombre de parts n	$d+1$	$2d+1$	$2d+1$	$d+1$	$d+1$
La complexité algorithmique	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	-	$\mathcal{O}(n)$
Détection des erreurs	Non	Oui	Non	Oui	Oui

TABLE 2.2 – Comparatif entre PDSM et l'état de l'art

Pour mesurer la résistance du PDSM face aux attaques du type SCA, nous avons calculé le taux de réussite d'une attaque d'ordre 2 en utilisant les même paramètres pour les différents schémas de masquage que nous avons présentés. Nous remarquons, que le schéma IPM a la meilleur résistance (à partir de 4200 traces), ensuite SSM (à partir de 3700 traces), puis PDSM (à partir de 3000 traces) et enfin BM

(à partir de 400 traces). La schéma IPM est une généralisation du schéma SSM, ce qui explique son score par rapport à SSM. Le schéma PDSM, utilise uniquement une graine de taille 8 bits (r) pour calculer le masque ($r \cdot \vec{h}$). Ainsi, pour obtenir une meilleur résistance face aux attaques d'ordres supérieurs, il sera intéressant d'étudier le comportement de ce schéma en utilisant plusieurs graines.

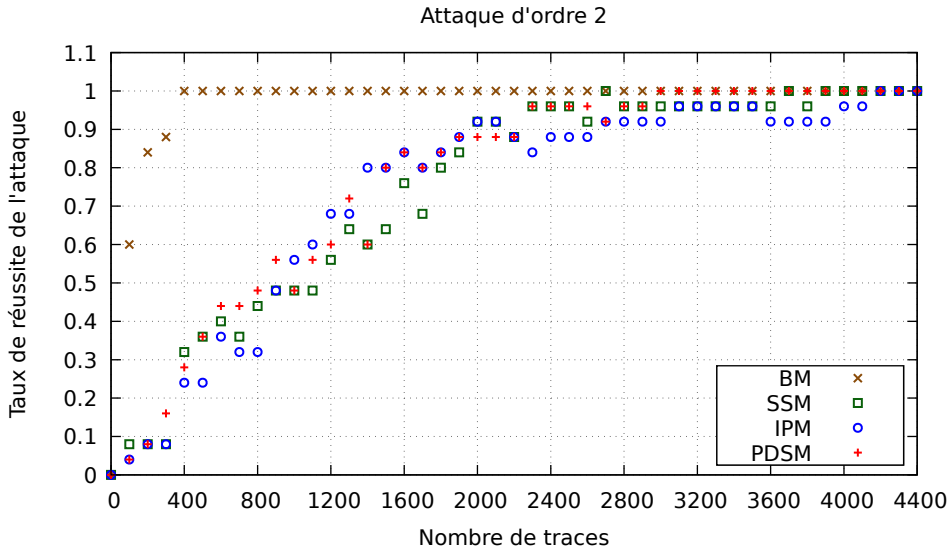


FIGURE 2.2 – Comparaison du taux de réussite d'une attaque SCA d'ordre 2 par rapport au nombre de traces entre le masquage PDSM et l'état de l'art.

2.4 Exemple général : La boîte de substitution (SubBytes)

Pour mieux observer le rôle de chacun des algorithmes que nous avons présentés dans ce chapitre, nous allons présenter une application de chacun d'entre eux dans une seule transformation qui les englobe tous. La transformation `SubBytes` est définie mathématiquement par la fonction suivante :

$$\begin{aligned}\texttt{SubByte}(x) \quad &= 63 \oplus 05x^{-1} \oplus 09x^{-2} \oplus F9x^{-4} \oplus 25x^{-8} \\ &\quad \oplus F4x^{-16} \oplus 01x^{-32} \oplus B5x^{-64} \oplus 8Fx^{-128} \in K \ .\end{aligned}$$

Cette fonction englobe les trois opérations que nous avons vues précédemment :

- `Add` : pour remplacer l'addition ($\oplus$) ;
- `Mult1` : pour remplacer la multiplication par un coefficient ;

— `Mult2` : pour remplacer la multiplication entre les données sensibles, dans ce cas il s'agit des puissances de x.

Avant de définir la version masquée de `SubBytes`, nous devons définir la fonction `Inverse` qui permettra de calculer l'inverse d'un élément dans K. Comme dans le section 1.1.2, nous allons nous baser sur le théorème d'Euler **??**. Pour le cas AES, $K = \mathbb{F}_2[\alpha]/(\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1)$. Nous avons : $\forall x \in K\backslash\{0\}, x^{255} = 1$, ainsi :

$$\forall x \in K\backslash\{0\}, \quad x^{-1} = x^{254} \ . \tag{2.3}$$

Pour calculer l'inverse, nous allons utiliser l'algorithme 2 dans [RP10], ce dernier permet de calculer cette exponentiation en utilisant le plus petit nombre possible d'opérations `Mult2`. Cependant, dans notre cas, il n'est pas nécessaire de rafraîchir le masque car cette opération est déjà incluse dans l'algorithme `Mult2`.

Algorithm 25 `Inverse`$(\vec{z})$ Complexité : $77n + 22$ multiplications et $44n + 11$ additions

entrée : $\vec{z} = \texttt{mask}(x) \in \mathcal{K}$
sortie : $\vec{y} = \texttt{mask}(x^{-1}) \in \mathcal{K}$

$\vec{x} \leftarrow \texttt{Mult2}(\vec{z}, \vec{z})$	$\vec{x} \leftarrow \vec{z}^2$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{x}, \vec{z})$	$\vec{y} \leftarrow \vec{z}^3$
$\vec{w} \leftarrow \texttt{Mult2}(\vec{y}, \vec{y})$	$\vec{w} \leftarrow \vec{z}^6$
$\vec{w} \leftarrow \texttt{Mult2}(\vec{w}, \vec{w})$	$\vec{w} \leftarrow \vec{z}^{12}$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{w}, \vec{y})$	$\vec{y} \leftarrow \vec{z}^{15}$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{y}, \vec{y})$	$\vec{y} \leftarrow \vec{z}^{30}$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{y}, \vec{y})$	$\vec{y} \leftarrow \vec{z}^{60}$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{y}, \vec{y})$	$\vec{y} \leftarrow \vec{z}^{120}$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{y}, \vec{y})$	$\vec{y} \leftarrow \vec{z}^{240}$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{y}, \vec{w})$	$\vec{y} \leftarrow \vec{z}^{252}$
$\vec{y} \leftarrow \texttt{Mult2}(\vec{y}, \vec{x})$	$\vec{y} \leftarrow \vec{z}^{254}$

renvoyer $\vec{y}$

Maintenant que nous avons conçu les fonctions de base qui composent l'AES, il est temps de les utiliser pour calculer la transformation `MSubBytes` :

Algorithm 26 $\texttt{MSubBytes}(\vec{z})$ Complexité : $136n + 36$ multiplications et $81n + 18$ additions

entrée : $\vec{z} = \texttt{mask}(x) \in \mathcal{K}$
sortie : $\texttt{mask}(\texttt{SubBvtes}(x)) \in \mathcal{K}$
$\vec{v} \leftarrow \texttt{Inverse}(\vec{z})$ $\vec{v} = \vec{z}^{-1}$
$\vec{u} \leftarrow \texttt{mask}(63)$
$\lambda = \{05, 09, F9, 25, F4, 01, B5\}$ Les coefficients de $\texttt{SubBvtes}$
pour $i = 1$ **jusqu'à** 7 **faire :**
    $\vec{u} \leftarrow \texttt{Add}(\vec{u}, \texttt{Mult1}(\lambda_i, \vec{v}))$ $\vec{u} = \vec{u} + \lambda \cdot \vec{v}$
    $\vec{v} \leftarrow \texttt{Mult2}(\vec{v}, \vec{v})$
$\vec{u} \leftarrow \texttt{Add}(\vec{u}, \texttt{Mult1}(8F, \vec{v}))$ $\vec{u} = \vec{u} + 8F \cdot \vec{v}$
renvoyer $\vec{u}$

2.5 Détection et correction des erreurs

Pour pouvoir détecter et corriger les erreurs, il est important que la valeur masquée soit un mot de code, pour $n = 2$ l'espace d'arrivée $\mathcal{K}$ correspond à l'espace vectoriel K^2, la distance minimale étant 1, il est donc impossible de détecter les erreurs. Pour surmonter cela, il faut que $\mathcal{K}$ soit un sous-espace vectoriel de K^n et donc un code correcteur d'erreur avec une distance minimale supérieure à 2. Nous proposons donc d'augmenter la taille de l'espace de départ à $n = 3$, et nous travaillons donc sur $\mathcal{K} \subsetneq K^3$.
Soient $\vec{g}, \vec{h}$ deux vecteurs dans $K^3 \backslash \{\vec{0}\}$ tels que : $\langle \vec{g}, \vec{h} \rangle = 0$, $\langle \vec{g}, \vec{g} \rangle \neq 0$ et $\langle \vec{h}, \vec{h} \rangle \neq 0$. Soient $\mathcal{C}$ et $\mathcal{D}$ les codes générés à partir de $\vec{g}$ et $\vec{h}$ respectivement. Le cardinal de $\mathcal{C}$ et $\mathcal{D}$ est $|\mathcal{C}| = |\mathcal{D}| = 2^8$. Soient $G, H \in \mathbb{F}_2^{8\times 24}$ la matrice génératrice de $\mathcal{C}$ et $\mathcal{D}$ respectivement. On rappelle que dans ce cas G et H ne sont pas toujours orthogonales ($GH^\top \neq 0$). Notons $\mathcal{K} = \mathcal{C} \oplus \mathcal{D}$ l'ensemble des valeurs masquées (i.e. $\mathcal{K} = \{\vec{z} = \vec{c} \oplus \vec{d} \mid \forall (\vec{c}, \vec{d}) \in \mathcal{C} \times \mathcal{D}\}$). La matrice génératrice de $\mathcal{K}$ est définie par :

$$J = \left[\frac{G}{H}\right] \in \mathbb{F}_2^{16\times 24} .$$

Propriété 16
$\mathcal{K}$ est un code linéaire, de paramètres $[24, 16, d_\mathcal{K}]$, avec une capacité de correction égale à :

$$\left\lfloor \frac{d_\mathcal{K}}{2} - 1 \right\rfloor .$$

Notons $L \in \mathbb{F}_2^{8\times 24}$ la matrice de parité de $\mathcal{K}$, i.e. $JL^\top = 0$. Si l'on considère $\vec{u} = \vec{z} + \vec{e}$ le mot masqué erroné, avec $\vec{e}$ le vecteur d'erreur. Le syndrome de décodage consiste

à calculer $\varepsilon = \Psi(\vec{u})L^\top = \Psi(\vec{z} \oplus \vec{e})L^\top = \Psi(\vec{z})L^\top \oplus \Psi(\vec{e})L^\top = \Psi(\vec{e})L^\top$. Si le syndrome n'est pas nul, ce qui signifie une injection d'erreur, alors la position de l'erreur correspond à la position de la colonne de L qui est égale à ε.

Exemple 11
Prenons $n = 3$, avec $\vec{g} = (1, 1, x^7 + x^4 + x^3)$ et $\vec{h} = (x^7 + x^4 + x^3 + 1, 1, 1)$ la distance minimale de $\mathcal{K}$ est $d_{\mathcal{K}} = 3$, donc le code peut corriger au plus une erreur, ce qui est suffisant pour protéger contre une attaque par injection de faute unique. La matrice génératrice et la matrice de parité de $\mathcal{K}$ sont définies respectivement par :

$$J = \left[\begin{array}{cccccccccccccccccccccccc}
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&0&1&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&0&1&0&0\\
\hline
1&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&1&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&1&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
1&1&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&1&1&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
1&1&1&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&1&1&1&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1
\end{array}\right] \begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ G \\ \\ \\ \\ \\ \\ \\ \\ H \end{array}$$

$$L = \left[\begin{array}{cccccccccccccccccccccccc}
1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&1&0\\
0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&1&1&1&1\\
0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&1&0&1&0&0&1&1&1\\
0&0&0&1&0&0&0&0&1&0&0&1&1&0&0&0&0&1&0&1&1&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&1&0&0&1&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&1&0&0&1&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&1&0&0&1\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0
\end{array}\right].$$

Soit $x = \alpha^7 + \alpha^3 + 1$ une valeur sensible, et $\vec{z} = \texttt{mask}(x) = (\alpha^5 + \alpha + 1, \alpha^6 + \alpha^4 + \alpha^3 + 1, \alpha^7 + \alpha^6 + \alpha^3 + \alpha)$, supposons que $\vec{z}$ a subi une erreur sur une position $i = 13$, celle-ci correspond au vecteur $\vec{e} = (0, x^4, 0)$. Le vecteur obtenu après l'injection de faute sera donc : $\vec{u} = (\alpha^5 + \alpha + 1, \alpha^6 + \alpha^3 + 1, \alpha^7 + \alpha^6 + \alpha^3 + \alpha) \in \mathcal{K}$.
Pour détecter la présence de l'erreur, nous calculons d'abord le syndrome :

$$\begin{aligned}
\varepsilon &= \Psi(\vec{u})L^\top \\
&= \left(\begin{array}{cccccccccccccccccccccccc} 1&1&0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&1&0&1&0&0&1&1 \end{array}\right)L^\top \\
&= \left[\begin{array}{c} 0\\0\\0\\1\\1\\0\\0\\0 \end{array}\right].
\end{aligned}$$

Le fait que le syndrome soit non nul prouve que le vecteur $\vec{u}$ est erroné. Pour détecter la position de l'erreur, il suffit de parcourir les colonnes de L et de voir laquelle est identique au syndrome. Dans cet exemple, on remarque bien que le syndrome ε équivaut la 13ème colonne de L, et il s'agit bien de la position de l'erreur. ∎

2.6 Conclusion

Dans ce chapitre, nous avons proposé une solution de masquage qui fait partie de la famille DSM. Cette solution, en plus de permettre de masquer les deux opérations d'addition et de multiplication, permet également de détecter et de corriger les erreurs. En terme de performance algorithmique, il s'agit d'un masquage de complexité linéaire, donc compatible avec les composants à faible capacité, en particulier les objets connectés ou les IoTs (*Internet of Things*). La génération d'un masque aléatoire est souvent une opération coûteuse, dans ce schéma le masque de taille $8n$ est généré à partir d'une graine de taille 8 bits.
Pour mesurer la résistance de ce schéma, nous lui avons fait subir l'attaque de Bruneau *et al.* [BGH+16] qui consiste à chercher une corrélation entre le poids de Hamming du masque choisi et toutes les valeurs possibles. D'après les résultats (Figure 2.2), bien que le masque PDSM soit généré à partir d'une graine de 8 bits (r), il reste plus résistant qu'un masquage classique d'ordre 2, et légèrement moins résistant par rapport à IPM (le masquage basé sur le produit scalaire) et SSM (le masquage basé sur le partage du secret). Cependant, il sera intéressant pour la suite d'étudier une version généralisée de ce schéma, en ajoutant plusieurs masques afin d'obtenir une meilleur résistance face aux attaques à l'ordre supérieur dans le modèle *probing*.

Deuxième partie

LA CRYPTOGRAPHIE BASÉE SUR L'IDENTITÉ

1 | Introduction

Sommaire

Nous allons consacrer ce chapitre à l'étude du schéma IBE de Boneh et Franklin, premier schéma efficace de chiffrement basé sur l'identité. Pour commencer, nous allons d'abord rappeler quelques notions de base sur la théorie des groupes, et en particulier sur les courbes elliptiques. Ensuite, nous allons utiliser ces outils pour expliquer le fonctionnement du schéma BF-IBE (en anglais : *Boneh and Franklin Identity Based Encryption*). Nous allons également présenter quelques améliorations qui ont été proposées par la communauté pour balayer les critiques qu'a subi le schéma.

1.1 Rappels sur la théorie des groupes

La construction des cryptosystèmes à clé publique tels que IBE ou RSA est souvent liée aux propriétés que l'on trouve dans la théorie des groupes. Dans cette section, nous allons faire quelques rappels sur les bases de la théorie des groupes.

Définition 26 (Un groupe)
Un groupe $(G, *)$ est un ensemble G auquel est associée une opération $*$ (appelée aussi "la loi de composition") vérifiant les quatre propriétés suivantes :

- $\forall x, y \in G, \quad x * y \in G$;
- $\forall x, y, z \in G, \quad x * (y * z) = (x * y) * z$;
- Il existe un élément neutre $e \in G$ tel que $\forall x \in G, \quad x * e = x$;
- $\forall x \in G$, il existe un élément inverse x^{-1} tel que $x * x^{-1} = e$.

Si en plus l'opération $*$ est commutative ($x * y = y * x$), on dit que G est un groupe *abélien*.

Propriété 17
Voici quelques propriétés de base sur les groupes :

- L'élément neutre e est unique. En effet, s'il existe un autre élément neutre e', alors : $e' * e = e = e'$;
- L'inverse de l'élément neutre est lui-même ;
- Chaque élément de G possède un seul inverse. En effet, s'il existe deux inverses x_1 et x_2 pour un seul élément x, alors : $x_1 * (x * x_2) = x_1 * e = x_1 = (x_1 * x) * x_2 = e * x_2 = x_2$.

Définition 27 (Un sous-groupe)
Une partie $H \subset G$ est un sous-groupe de G si :

- $e \in H$;
- pour tout $x, y \in H$, on a $x * y \in H$;
- pour tout $x \in H$, on a $x^{-1} \in H$.

Propriété 18
Soit H un sous-groupe de G, et r, s deux éléments de G. Nous avons : $Hr = Hs$

(respectivement $rH = sH$) si et seulement si : $r * s^{-1} \in H$ (respectivement $s^{-1} * r \in H$). Sinon, Hr et Hs (respectivement rH et sH) n'ont aucun élément en commun.

Preuve :
Nous avons :

$$\begin{aligned} Hr = Hs &\implies r \in Hs \\ &\implies \exists h \in H \quad | \quad h * s = r \\ &\implies h = r * s^{-1} \in H \ . \end{aligned}$$

$$\begin{aligned} r * s^{-1} \in H &\implies H(r * s^{-1}) = H \\ &\implies (Hr)s^{-1} = H \\ &\implies Hr = Hs \ . \end{aligned}$$

Supposons qu'il existe un élément commun $h \in Hr \cap Hs$, alors, il existe $h_1, h_2 \in H$ tels que : $h_1 * r = h_2 * s = h$ ce qui implique que : $r = h_1^{-1} * h_2 * s$, donc $r * s^{-1} = h_1^{-1} * h_2 \in H$ ce qui signifie que $Hr = Hs$ par le premier résultat. ■

Définition 28 (L'ordre du groupe)
Un groupe est dit fini si le nombre de ses éléments est fini. Dans ce cas, son cardinal est appelé l'ordre du groupe, il est noté $|G|$.

Définition 29 (L'ordre d'un élément)
L'ordre d'un élément $x \in G$ est le plus petit entier n tel que $\underbrace{x * \cdots * x}_{n \text{ fois}} = x^n = e$.

Théorème 4 (Théorème de Lagrange)
L'ordre d'un élément $x \in G$ divise toujours l'ordre de G.

Preuve :
Il suffit de démontrer que l'ordre du sous-groupe généré par x divise l'ordre de G. Notons H le sous-groupe généré par x (i.e. $H = \{x^i \quad | \quad 1 \leqslant i \leqslant ordre(x)\}$). Soit $r_1 \in G$, nous avons $|Hr_1| = |H|$, et soit $r_2 \in G \backslash Hr_1$, d'après la propriété 18, Hr_1 et Hr_2 dont disjoints, i.e. $|Hr_1 \cup Hr_2| = 2|H|$. De la même manière, nous pouvons continuer à choisir un $r_i \in G \backslash Hr_1 \cup Hr_2 \cup \cdots \cup Hr_{i-1}$ jusqu'à un certaine entier $i = n$ tel que : $|G| = n|H|$ avec $G = Hr_1 \cup \cdots \cup Hr_n$.

■

Définition 30 (Un groupe cyclique)
Un groupe G est dit cyclique s'il existe un élément $g \in G$ tel que : $\forall x \in G$, il existe un entier n vérifiant $x = g^n$. Dans ce cas, g est appelé générateur du groupe G.

Définition 31 (Un anneau)
Un anneau est un ensemble non vide A muni de deux lois de composition internes, l'une notée comme une addition et l'autre comme une multiplication, vérifiant les propriétés suivantes :

- A est un groupe abélien pour l'addition (on note 0 son élément neutre) ;
- La multiplication est associative : $\forall x, y, z \in A, \quad (xy)z = x(yz)$;
- La multiplication est distributive par rapport à l'addition (à gauche et à droite) : $\forall x, y, z \in A, \quad x(y+z) = xy + xz$ et $(x+y)z = xz + yz$;
- Si en plus la multiplication est commutative ($xy = yx$) alors on dit que A est *commutatif* ;
- On dit que A est *unitaire* si de plus la multiplication admet un élément neutre (que l'on note 1).

Définition 32 (Un sous-anneau)
Soit A un anneau. On appelle sous-anneau de A toute partie non-vide B de A qui vérifie les deux conditions suivantes :

- B est un sous-groupe du groupe additif de A ;
- Pour tout $x, y \in B$ on a $xy \in B$.

Définition 33 (Un corps)
On appelle corps commutatif (ou plus simplement corps) tout anneau commutatif unitaire dans lequel tout élément non-nul est inversible.

1.2 Les courbes elliptiques

1.2.1 Définition

La première utilisation des courbes elliptiques en cryptographie remonte au milieu des années 80, lorsque Miller [Mil85] puis Koblitz [Kob87] ont proposé indépendamment les courbes elliptiques comme outils pour la cryptographie.

Définition 34 (Une courbe elliptique)
Une courbe elliptique E, définie sur un corps fini $\mathbb{F}$ dont la caractéristique est différente de 2 et 3, est un ensemble "de points" défini par :

$$E = \{(x, y) \in \mathbb{F}^2 \quad | \quad y^2 = x^3 + ax^2 + b\} \cup \{P_\infty\} \ ,$$

tel que : $\Delta = 4a^3 + 27b^2 \neq 0$.
P_∞ est un élément supplémentaire appelé *point à l'infini* de la courbe.

Les courbes elliptiques ont une structure de groupe qui constitue un remplacement idéal pour les groupes traditionnels basés sur la théorie des nombres, tels que Diffie-Hellman ou ElGamal. Une courbe elliptique E sur un corps fini $\mathbb{F}$ est munie d'une

structure de groupe abélien dont l'élément neutre est le point à l'infini. On peut définir une opération d'addition de points (notée +) de la façon suivante :

— $\forall P \in E, \quad P + P_\infty = P$;
— L'opposé de $P = (x, y) \in E$ est défini par $-P = (x, -y)$;
— Soient P et Q deux points de la courbe E, la droite passant par P et Q (ou la tangente sur P si $P = Q$) recoupe la courbe E en un troisième point $R \in E$, ce dernier est par définition l'opposé de $P + Q$ i.e. $P + Q = -R$.

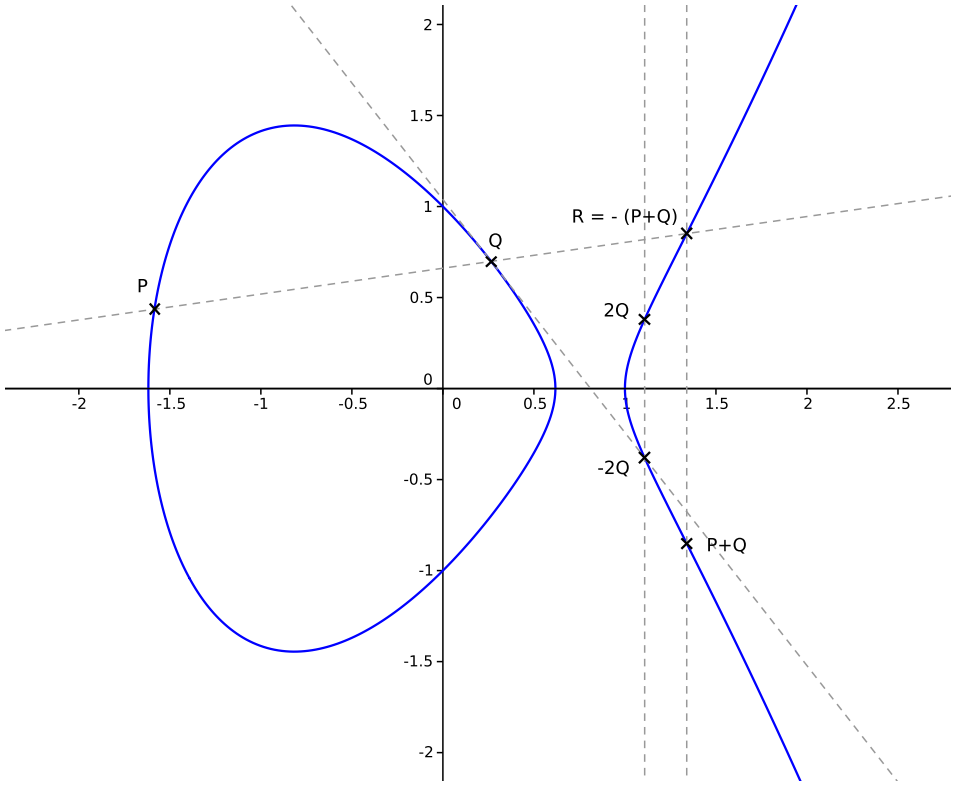


FIGURE 1.1 – Représentation graphique de la courbe : $y^2 = x^3 - 2x + 1$ sur $\mathbb{R}$

1.2.2 Les formules d'addition et doublement

Comme le point à l'infini est un élément neutre, nous allons présenter dans cette partie la formule qui permet de calculer la somme de deux points finis d'une courbe. La figure 1.1 montre que la somme de deux points différents $P = (x_P, y_P)$ et $Q = (x_Q, y_Q)$, tels que $x_P \neq x_Q$, est l'opposé du point d'intersection de la droite (PQ) avec la courbe. Ainsi, pour calculer les coordonnés de $P + Q$, nous allons commencer par calculer l'équation la droite (PQ) en fonction des coordonnées de

P et Q. Le coefficient directeur de cette droite est :

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P} \ . \tag{1.1}$$

La droite (PQ) a donc pour équation $y = \lambda x + \mu$ où $\mu = \lambda x_P + y_P$. Elle coupe la courbe E en un troisième point R dont les coordonnées (x_R, y_R) vérifient simultanément :

$$\begin{cases} y_R = \lambda x_R + \mu & \text{car } R \text{ appartient à la droite } (PQ) \\ y_R^2 = x_R^3 + a x_R + b & \text{car } R \text{ appartient à la courbe } E \end{cases}$$

En résolvant ce système d'équations, on trouve les formules suivantes :

$$\begin{cases} x_S &= \lambda^2 - x_P - x_Q \\ y_S &= -\lambda(x_S - x_P) - y_P \end{cases} \tag{1.2}$$

avec $(x_S, y_S) = (x_R, -y_R) = P + Q$.
Pour calculer le double d'un point P, c'est-à-dire $2P$ (i.e. lorsque $P = Q = (x_P, y_P)$), le raisonnement précédent s'applique également, en considérant la tangente en P sur la courbe. Les formules de l'équation 1.2 restent valables, seule la valeur de λ est différente. Dans ce cas, il faudra considérer le coefficient directeur de la tangente qui est donné par l'équation :

$$\lambda = \frac{3x_P^2 + a}{2y_P} \ . \tag{1.3}$$

Le dernier cas qui reste, c'est quand nous avons deux points P et Q qui sont symétriques par rapport à l'axe des abscisses (i.e. $x_P = x_Q$ et $y_P = -y_Q$), dans ce cas nous avons : $P = -Q$ et donc $P + Q = P - P = P_\infty$.

1.2.3 Le groupe de torsion

Définition 35 (Multiplication par un entier)
Pour tout entier relatif n, la multiplication par n, notée $[n]$ et définie par : $[n] : P \mapsto nP$, est un morphisme de E vers E (avec $nP = \underbrace{P + \cdots + P}_{n \text{ fois}}$).

Définition 36 (Groupe de n-torsion)
Pour tout entier n, le groupe de n-torsion d'une courbe elliptique E est le noyau de l'isogénie de multiplication par n, on le note $E[n]$:

$$E[n] = \{P \in E \quad | \quad nP = P_\infty\} \ .$$

Le corps $\mathbb{F}$ étant supposé algébriquement clos, le polynôme $x^3 + ax + b$ a trois racines dans $\mathbb{F}$. La courbe étant lisse ($\Delta \neq 0$), ces trois racines sont distinctes. Donc il existe toujours trois points distincts (*les points spéciaux*) dans E, qui sont $P_\alpha = (\alpha, 0)$, $P_\beta = (\beta, 0)$, $P_\gamma = (\gamma, 0)$ tels que : $x^3 + ax + b = (x - \alpha)(x - \beta)(x - \gamma)$. Les points spéciaux se distinguent des autres points de la courbe par le fait d'être les seuls points d'ordre 2.

1.2.4 Le couplage sur les courbes elliptiques

La cryptographie basée sur les courbes elliptique est souvent liée aux fonctions bilinéaires admissibles.

Définition 37 (Fonction bilinéaire admissible)
Soient q un nombre premier, $(G_1, +)$ un groupe fini d'ordre q noté additivement, et $(G_2, \times)$ un groupe fini d'ordre q également mais noté multiplicativement. Comme q est premier, ces groupes sont cycliques. On note P un générateur de G_1.

Une application $\beta : G_1 \times G_1 \rightarrow G_2$ est dite admissible pour la cryptographie bilinéaire si elle possède les quatre propriétés suivantes :

- Elle est bilinéaire : $\forall P, Q \in G_1, \forall a, b \in \mathbb{Z}, \quad \beta(aP, bQ) = \beta(P, Q)^{ab}$;
- Elle est non dégénérée : $\beta(P, P) \neq 1$;
- Elle est efficacement calculable : il existe un algorithme de complexité polynomiale qui, à partir de P et Q dans G_1, calcule la valeur de $\beta(P, Q)$;
- Étant donnés P, aP, bP et Q dans G_1, il doit être en pratique impossible de trouver la valeur de $\beta(P, Q)^{ab}$. Cela implique en particulier que le problème du logarithme discret doit être difficile dans G_1.

Les deux fonctions bilinéaires admissibles souvent utilisées en cryptographie basée sur les courbes elliptiques sont le couplage de Tate et le couplage de Weil. La forme originale de ces deux fonctions nécessite des opérations mathématiques complexes. Afin de gagner en performance, beaucoup de schémas cryptographiques ont été conçus en utilisant une notation abstraite de ces fonctions. La sécurité de ces schémas cryptographiques est basée sur le problème BDH (en anglais : *Bilinear Diffie-Hellman Problem*).

La bilinéarité des couplages a permis la construction de protocoles originaux et l'amélioration des protocoles existants :

- L'échange de Diffie-Hellman entre trois parties (Joux 2001) ;
- Les schémas de signature courte (Boneh, Lynn, Shacham 2001) ;
- La cryptographie basée sur l'identité (Boneh et Franklin 2001).

1.2.5 L'échange des clés à trois

Nous commençons par le principe classique d'échange de Diffie-Hellman à trois. Soient Alice, Bob et Charlie trois utilisateurs qui veulent échanger une clé secrète commune pour l'utiliser dans un algorithme de chiffrement symétrique. Ils commencent par se mettre d'accord sur le choix d'un groupe de points $(G, +)$ d'ordre p et d'un point générateur $P \in G$. Chaque utilisateur va choisir une clé secrète dans $\mathbb{Z}_p$: a pour Alice, b pour Bob, et c pour Charlie.

— Alice calcule aP et l'envoie à Bob,
— Bob calcule bP et l'envoie à Charlie,
— Charlie calcule cP et l'envoie à Alice.
— Alice obtient cP. Elle calcule acP et l'envoie à Bob,
— Bob obtient aP. Il calcule abP et l'envoie à Charlie,
— Charlie obtient bP. Elle calcule bcP et l'envoie à Alice.

Enfin chaque utilisateur va multiplier la clé reçue par sa clé secrète pour obtenir la clé commune $abcP$ comme illustré dans la figure 1.2.

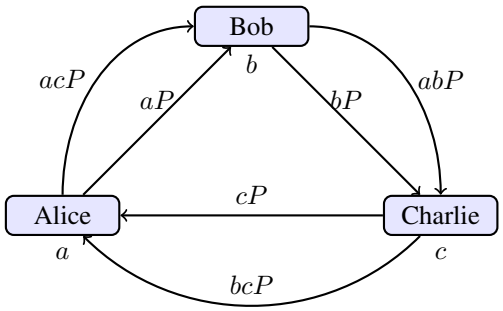


FIGURE 1.2 – Le schéma classique d'échange de clé à trois

JOUX en 2000 a décrit l'échange de DIFFIE-HELLMAN à trois avec un seul tour d'échange [Jou00] en utilisant une fonction bilinéaire admissible $e : G_1 \times G_1 \to G_2$:

— Alice envoie aP à Bob et Charlie,
— Bob envoie bP à Alice et Charlie,
— Charlie envoie cP à Alice et Bob.

Chaque utilisateur calculera ensuite la clé commune : $e(bP, cP)^a = e(aP, cP)^b = e(aP, bP)^c = e(P, P)^{abc}$.

1.2.6 La signature courte

Pour assurer l'intégrité et l'authentification des messages, des schémas de signatures ont été mis en place. BONEH, LYNN, et SHACHAM [BLS01] ont proposé en 2001 un protocole de signature courte à base de couplage dont les signatures sont assez courtes. Les paramètres du schéma sont :

— Une fonction bilinéaire admissible $e : G_1 \times G_1 \to G_2$;
— Un générateur $P \in G_1$;
— Une fonction de hachage : $\mathcal{H} : \{0,1\}^* \to G_1$.

Pour signer un message M, Alice choisit un secret $a \in \mathbb{Z}_p$ ensuite :

— Elle calcule sa clé publique aP ;
— Elle calcule $H = \mathcal{H}(M) \in G_1$;
— Elle envoie à Bob le couple : M, aH.

Pour vérifier la signature, il suffit de tester l'égalité $e(\mathcal{H}(M), aP) \stackrel{?}{=} e(aH, P)$.

1.3 Le schéma de Boneh et Franklin

Dans une infrastructure à clé publique classique, l'existence d'une autorité de certification (CA) est essentielle. Les utilisateurs d'une clé publique doivent s'assurer que la clé secrète associée est possédée par l'entité correcte. Cette assurance peut être fournie en ayant une autorité de certification de confiance qui lie la clé publique d'une entité et son identité sous la forme d'un certificat signé. Si cette signature est valide, alors les données intégrées dans le certificat sont correctes. Comme nous l'avons détaillé dans la partie 1.5.3, la gestion de ces certificats est souvent compliquée à mettre en œuvre et présente plusieurs défauts.

En 2001, Boneh et Franklin ont introduit leur système de chiffrement basé sur l'identité, aussi connu sous le nom IBE (en anglais : *Identity Based Encryption*) [BF01]. Sa mise en œuvre repose sur un groupe de points sur une courbe elliptique (souvent un groupe de n-torsion) et la fonction bilinéaire correspondante pour effectuer les calculs nécessaires.

Une architecture IBE est composée de quatre algorithmes principaux :

— `Initialisation` : Prend un paramètre de sécurité k et renvoie les paramètres publics du système et la clé maître. Les paramètres du système comprennent une description d'un espace de messages fini $\mathcal{M}$, une description d'un espace de texte chiffré fini $\mathcal{C}$, les fonctions mathématiques utilisées (la courbe elliptique, la fonction de couplage et la fonction de hachage). Les paramètres du système seront connus publiquement, tandis que la clé maître sera connue uniquement par le *générateur de clé privée* appelé le PKG (en anglais : *Public Key Generator*).
— `Extraction` : Prend comme paramètres d'entrée la clé maître et une identité $id \in \{0,1\}^*$, et renvoie une clé secrète associée S. L'identité est une chaîne arbitraire qui sera utilisée comme clé publique grâce à une fonction de hachage et S correspond à la clé de déchiffrement privée correspondante. L'algorithme Extraction permet donc d'extraire une clé privée à partir d'une clé publique grâce à la clé maître, ce qui signifie que seul le PKG est capable d'exécuter cet algorithme.

— `Chiffrement` : Prend comme paramètres d'entrée l'identité du destinataire id et le message clair $M \in \mathcal{M}$. Il retourne un texte chiffré $C \in \mathcal{C}$.
— `Déchiffrement` : Prend comme paramètres d'entrée un texte chiffré $C \in \mathcal{C}$ et la clé secrète du destinataire légitime S. Il retourne le texte clair $M \in \mathcal{M}$.

Ces algorithmes doivent satisfaire la contrainte de cohérence standard, à savoir quand S est la clé secrète générée par l'algorithme `Extract` quand on lui donne id comme clé publique, alors :

$$\forall M \in \mathcal{M}, \quad \texttt{Dec}(\texttt{Enc}(M, id), S) = M \ , \tag{1.4}$$

avec `Enc` et `Dec` désignant les algorithmes de chiffrement et déchiffrement respectivement.

1.3.1 Les avantages du schéma de Boneh et Franklin (BF-IBE)

Le principal avantage d'un IBE est qu'il permet d'extraire une clé publique à partir d'une chaîne de caractères arbitraire. Cette caractéristique nous offre plusieurs options :

— La révocation des clés : Pour gérer la révocation des clés, plus précisément l'expiration des clés, Boneh et Franklin ont proposé de concaténer l'identité de l'utilisateur avec une date de validité. Grâce à ce mécanisme, il est possible de chiffrer des messages valables à partir d'une date précise.
— L'usurpation d'identité : Les algorithmes de chiffrements basés sur les problèmes d'arithmétique nécessitent l'intervention d'une autorité de certification afin d'approuver l'identité du possesseur d'une clé publique. Tandis que dans un schéma IBE, il n'est nul besoin de faire cette vérification, puisque la clé publique est générée par n'importe quel utilisateur.
— La sauvegarde des clés : Les clés secrètes des utilisateurs sont générées par le PKG, il est donc toujours possible de les régénérer en cas de perte.

1.3.2 Le schéma BF-IBE

Pour bien distinguer les données publiques et les données sensibles, nous avons choisi de mettre toute valeur sensible en couleur rouge.

`Initialisation`
— Entrée : Le paramètre de sécurité k ;
— Choisir un nombre premier p de longueur k ;
— Choisir la clé maître $s \in \mathbb{Z}_p$;

— Choisir une fonction bilinéaire admissible $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$;
— Choisir une fonction de hachage $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{G}$;
— Choisir une fonction de hachage $\mathcal{H}_2 : \mathbb{G}_T \to \mathcal{M}$, avec $\mathcal{M}$ l'ensemble des messages ;
— Choisir un élément générateur du système : $P \in \mathbb{G}$;
— Calculer la clé publique du PKG : $Y = sP$;
— Sortie : $PP = (\mathbb{G}, \mathbb{G}_T, e, P, Y, \mathcal{H}_1, \mathcal{H}_2)$, $MSK = s$.

Le premier algorithme renvoie les paramètres publiques du système (PP), et la clé maître MSK (en anglais : *Master Secret Key*) possédée par le PKG, cette clé sera utilisée par l'algorithme `Extraction` pour générer les clés secrètes des utilisateurs.

`Extraction`
— Entrée : Une identité $id \in \{0,1\}^*$;
— Calculer la clé publique de l'utilisateur : $Q_{id} = \mathcal{H}_1(id) \in \mathbb{G}$;
— Calculer la clé secrète de l'utilisateur : $S_{id} = sQ_{id} \in \mathbb{G}$;
— Sortie : Q_{id}, S_{id}.

Comme son nom l'indique, l'algorithme `Extraction` permet d'extraire une clé secrète S_{id} à partir d'une clé publique Q_{id}. La clé publique est calculée grâce à une fonction de hachage à partir d'une chaîne de caractères arbitraire appelée id (l'identité). Il peut s'agir du nom de l'utilisateur, son adresse mail, un identifiant ou autre. Il est également possible de concaténer cette chaîne avec une date précise dans le futur, afin de s'assurer que le possesseur ne peut pas déchiffrer avant cette date.

`Chiffrement`
— Entrée : Un message $M \in \mathcal{M}$, l'identité du destinataire $id \in \{0,1\}^*$;
— Choisir une valeur aléatoire $r \in \mathbb{Z}_p^*$;
— Calculer $g_{id} = e(\mathcal{H}_1(id), Y) \in \mathbb{G}_T$;
— Calculer $U = rP \in \mathbb{G}$;
— Calculer $V = M \oplus \mathcal{H}_2(g_{id}^r) \in \mathcal{M}$;
— Sortie : $C = (U, V) \in \mathbb{G} \times \mathcal{M}$.

Grâce à l'aléa r qui intervient dans le processus de chiffrement, cet algorithme peut renvoyer des cryptogrammes différents pour un même message. Il s'agit bien d'un chiffrement non-déterministe. Cette propriété ajoute une nouvelle couche de sécurité pour ce chiffrement.

`Déchiffrement`
— Entrée : Un cryptogramme $C = (U, V) \in \mathbb{G} \times \mathcal{M}$;
— Calculer $M = V \oplus \mathcal{H}_2(e(S_{id}, U))$;
— Sortie : $M \in \mathcal{M}$.

Pour vérifier la propriété 1.4 pour l'algorithme `Déchiffrement` il suffit de montrer que : $g_{id}^r = e(S_{id}, U)$, nous avons :

$$\begin{aligned} e(S_{id}, U) &= e(sQ_{id}, rP) \\ &= e(Q_{id}, sP)^r \\ &= e(\mathcal{H}_1(id), Y)^r \\ &= g_{id}^r \ . \end{aligned}$$

1.3.3 La sécurité du schéma BF-IBE

La sécurité du système de chiffrement BF-IBE repose sur la difficulté du problème Diffie-Hellman bilinéaire (aussi appelé BDH : en anglais *Bilinear Diffie–Hellman*).

Définition 38 (Le problème BDH)
Soient $\mathbb{G}, \mathbb{G}_T$ deux groupes d'ordre premier p, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ une fonction bilinéaire admissible, et P un élément générateur de $\mathbb{G}$. Le problème BDH consiste à calculer $e(P, P)^{abc} \in \mathbb{G}_T$ étant donné P, aP, bP, cP pour $a, b, c, \in \mathbb{Z}_p$.

On dit qu'un algorithme $\mathcal{A}$ a l'avantage ϵ pour résoudre le problème BDH si :

$$Pr\left[\mathcal{A}(P, aP, bP, cP) = e(P, P)^{abc}\right] \geqslant \epsilon \ . \tag{1.5}$$

Il existe aussi une version décisionnelle du problème (DBDH), qui consiste à trouver un distingueur entre $e(P, P)^{abc}$ et une valeur aléatoire dans $\mathbb{G}_T$.

Définition 39 (Le problème DBDH)
Soient trois entiers $a, b, c \in \mathbb{Z}_q^*$, $P \in \mathbb{G} \backslash \{P_\infty\}$ un point de la courbe, et $T \in \mathbb{G}_T$. Le problème DBDH (en anglais : *Decisional Bilinear Diffie-Hellman*) consiste à déterminer si $T = e(P, P)^{abc}$ ou un élément aléatoire dans $\mathbb{G}_T$, avec $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ une fonction bilinéaire admissible.

Pour analyser la sécurité de certaines constructions cryptographiques, Bellare et Rogaway ont introduit un modèle de sécurité appelé modèle d'oracle aléatoire [BR93].

Définition 40 (Oracle aléatoire)
Un oracle aléatoire est une fonction $\mathcal{H} : X \to Y$ choisie uniformément au hasard dans l'ensemble de toutes les fonctions $h : X \to Y$ (on suppose que Y est un ensemble fini). Un algorithme peut interroger l'oracle aléatoire pour n'importe quelle valeur d'entrée $x \in X$ et recevoir la valeur $\mathcal{H}(x)$ en réponse.

Les oracles aléatoires servent à modéliser les fonctions de hachage utilisées en cryptographie telles que SHA-2 (voir le tableau 1.1). Notez que la sécurité dans le modèle oracle aléatoire n'implique pas la sécurité dans le monde réel, néanmoins le modèle oracle aléatoire est un outil très utile pour la validation des constructions cryptographiques. Pour l'adapter aux schémas IBE, Boneh et Franklin ont renforcé le modèle IND-CCA pour faire face à un adversaire qui possède des clés privées correspondant aux identités de son choix. Les schémas IBE sont prouvés sûrs selon des scénarios d'attaques très proches des scénarios des schémas de chiffrement à clé publique, avec en plus quelques spécificités pour la fonction `Extraction`. Dans le cas d'un IBE, un adversaire est autorisé à accéder à un oracle d'extraction qui permet de calculer la clé privée à partir d'une identité id.

Les modèles d'attaques : Avant d'introduire les modèles de sécurité dans le cadre d'IBE, rappelons d'abord la liste des modèles d'attaques connues en cryptanalyse :

KPA : L'attaque à clair connu (en anglais : *Known Plaintext Attack*) est un modèle d'attaque où l'attaquant a accès à la fois au texte en clair, et à son cryptogramme.

CPA : L'attaque à clair choisi (en anglais : *Chosen Plaintext Attack*) est un modèle d'attaque qui suppose que l'attaquant peut obtenir les textes chiffrés pour des messages en clair choisis. Le but de l'attaque est d'obtenir des informations qui réduisent la sécurité du schéma de chiffrement. Cette attaque est plus puissante que KPA car le cryptanalyste peut choisir des textes en clair spécifiques qui donneront plus d'informations sur la clé secrète.

CCA : L'attaque à cryptogramme choisi (en anglais : *Chosen Ciphertext Attack*) est un modèle d'attaque où l'attaquant peut recueillir des informations en obtenant les clairs associés à des cryptogrammes choisis. À partir de ces informations, l'adversaire peut tenter de récupérer la clé secrète utilisée pour le déchiffrement.

CPA2 : Il s'agit d'une version modifiée de CPA où l'attaquant peut demander les textes chiffrés des textes en clair supplémentaires après avoir vu les textes chiffrés pour certains textes en clair.

Le modèle IND-ID-CPA : Attaques adaptatives à clairs choisis.
Premièrement, considérons le concept plus faible de sécurité sémantique qui suppose un attaquant interne capable d'émettre des requêtes à un oracle d'extraction (algorithme `Extraction`), qui lui permet d'extraire la clé privée correspondant à des identités de son choix. Le succès de l'attaquant IND-ID-CPA est formalisé par son avantage dans le jeu suivant :

Initialisation : Le challenger choisit un paramètre de sécurité k et exécute l'algorithme `Initialisation`. Il retourne à l'adversaire $\mathcal{A}$ les paramètres publics, et garde la clé maître s.

Phase 1 : L'adversaire peut effectuer adaptativement des requêtes à l'oracle d'extraction $\mathcal{O}(id_i)$ qui lui retourne la clé privée S_{id}. À la fin de cette phase, l'adversaire retourne deux messages de même longueur m_0 et m_1 et une identité id' avec une restriction sur id', pour laquelle $\mathcal{A}$ ne doit pas faire de requête d'extraction de clé privée dans les phase 1 et 2.

Challenge : Le challenger tire un bit aléatoire $b \in \{0,1\}$ et calcule le cryptogramme $c = \texttt{Chiffrement}(S_{id'}, id', m_b)$, et l'envoie à l'adversaire.

Phase 2 : Comme dans la première phase, l'adversaire peut effectuer adaptativement des requêtes à l'oracle d'extraction $\texttt{Extraction}(id_i)$ qui lui retourne la clé privée S_{id} , où $id_i \neq id'$.

Réponse : À la fin de son attaque, l'adversaire retourne son estimation $b' \in \{0,1\}$ de b et il gagne le défi si $b' = b$. L'avantage de $\mathcal{A}$ et donné par : $Adv_{\mathcal{E},\mathcal{A}}(k) = |Pr[b = b'] - 1/2|$.

Définition 41
Un schéma IBE $\mathcal{E}$ est sémantiquement sûr contre les attaques adaptatives à clairs choisis si pour tout adversaire IND-ID-CPA en temps polynomial $\mathcal{A}$ la fonction $Adv_{\mathcal{E},\mathcal{A}}(k)$ est négligeable.

Le modèle IND-ID-CCA : Attaques adaptatives à chiffrés choisis.
Dans ce modèle l'attaquant $\mathcal{A}$ est supposé pouvoir accéder de manière adaptative à un oracle de déchiffrement, mais également à un oracle lui donnant les clés privées correspondant à des identités de son choix. Le succès de l'attaquant IND-ID-CCA est formalisé par son avantage dans le jeu suivant :

Initialisation : Le challenger choisit un paramètre de sécurité k et exécute l'algorithme `Initialisation`. Il retourne à l'adversaire $\mathcal{A}$ les paramètres publics et garde la clé secrète maıtre s.

Phase 1 : L'adversaire peut effectuer de manière adaptative les requêtes à :
- L'oracle d'extraction $\mathcal{O}_{\texttt{Extract}}(id_i)$: qui retourne la clé privée S_{id_i}.
- L'oracle de déchiffrement $\mathcal{O}_{\texttt{Dec}}(c_i, id_i)$: qui renvoie le clair (s'il existe) qui correspond au chiffré c_i pour l'identité id_i.

A la fin de cette phase, l'adversaire retourne deux messages (m_0, m_1) de même longueur et une identité id' avec une restriction sur id' , pour laquelle $\mathcal{A}$ ne doit pas faire de requête d'extraction de clé privée dans les phase 1 et 2.

Challenge : Le challenger tire un bit aléatoire $b \in \{0,1\}$ et calcul le chiffré $c = \texttt{Chiffrement}(id', m_b)$, et l'envoie à l'adversaire.

Phase 2 : A la manière de la première phase, l'adversaire peut effectuer adaptativement les requêtes à :

— L'oracle d'extraction $\mathcal{O}_{\texttt{Extract}}(id_i)$: qui retourne la clé privée S_{id_i} où $id_i \neq id'$.
— L'oracle de déchiffrement $\mathcal{O}_{\texttt{Dec}}(c_i, id_i)$: qui renvoie le clair (s'il existe) qui correspond au chiffré c_i pour l'identité id_i où $c_i \neq c$.

Réponse : À la fin de son attaque, l'adversaire retourne son estimation $b' \in \{0, 1\}$ de b. Il gagne le jeu si $b' = b$ et l'avantage de $\mathcal{A}$ et donné par : $Adv_{\mathcal{E},\mathcal{A}}(k) = |Pr[b = b'] - 1/2|$.

Définition 42
Un schéma IBE $\mathcal{E}$ est sémantiquement sûr contre les attaques adaptatives à chiffrés choisis si pour tout adversaire IND-ID-CCA en temps polynomial $\mathcal{A}$, la fonction $Adv_{\mathcal{E},\mathcal{A}}(k)$ est négligeable.

Boneh et Franklin ont prouvé que dans un modèle d'oracle aléatoire, le protocole était sémantiquement sécurisé selon l'hypothèse BDH. Cela signifie qu'un attaquant opérant dans un certain cadre et capable de casser le système IBE pourrait tout aussi bien être utilisé pour résoudre BDH.

1.3.4 L'enrôlement et la mise à jour des clés

La politique de distribution des clés est un point important dans un SGC (*Schéma de Gestion des Clés*). En effet, en cryptographie asymétrique, il est décisif de distinguer l'envoi d'une clé publique sur un canal non sécurisé et l'envoi d'une clé privée sur un canal sécurisé. La distribution des clés privées se fait généralement pendant la phase d'enrôlement et la phase de mise à jour des clés (*rekeying*).
Pour enrôler un utilisateur, ce dernier doit envoyer une requête au PKG qui assurera l'authenticité de l'identité de cet utilisateur. Le PKG exécutera ensuite l'algorithme `Extraction` pour générer la clé de l'utilisateur et la lui transmettra accompagnée des paramètres publics du système (PP) sur un canal sécurisé (par exemple : sur un support physique comme une clé usb, ou une carte à puce). Il est toujours possible pour l'utilisateur de vérifier l'exactitude de ses clés en testant :

$$e(Q_{id}, Y) \stackrel{?}{=} e(S_{id}, P) \ . \tag{1.6}$$

En effet, ces deux valeurs valent $e(Q_{id}, P)^s$ qui dépend de la clé maître du PKG. Pour mettre à jour les clés dans le schéma BF-IBE, il suffit de générer une nouvelle clé maître MSK, tout en gardant la précédente dans un lieu sûr (un coffre) pour les besoins judiciaires que nous avons évoqués dans la partie 1.5.3.

1.4 Critiques et améliorations

Depuis son apparition, BF-IBE est visé par deux critiques principales :

1. **Le problème de centralisation** : Dans ce schéma, le PKG détient tout le pouvoir, il est capable d'extraire à lui seul les clé secrètes de tous les utilisateurs, ce qui lui permet de déchiffrer toutes les communications, les falsifier, et même usurper les identités.
2. **Le problème de révocation** : Le cas de révocation d'une entité est rarement traité dans les schémas de gestion de clés basés sur l'identité. Dans le schéma BF-IBE, le destinataire doit interroger le PKG à chaque communication pour obtenir sa clé de déchiffrement. Bien que cette option empêche les utilisateurs révoqués de déchiffrer, elle oblige le PKG à rester en écoute pour valider chaque échange entre utilisateurs, et établir à chaque fois un canal sécurisé pour communiquer la clé de déchiffrement.

Au fur et à mesure, plusieurs auteurs ont proposé des solutions pour résoudre le problème de centralisation. Al-Riyami et Paterson [AP03] ont construit CL-PKC (en anglais : *Certificatless Public Key Cryptography*) qui permet à toute entité valide de construire ses propres clés privées et publiques. Bien que ce système résolve le problème de centralisation, il ne résout pas complètement celui de l'autorité de séquestre (*Key Escrow Problem*) [XZQ05]. De plus, la mise à jour des clés doit se faire sur un canal privé.
Lee *et al.* [LBD+04] ont présenté un schéma IBE non centralisé. Le schéma définit plusieurs autorités de protection des clés appelées KPA (en anglais : *Key Privacy Authoritie*) en plus de l'autorité standard qui est le PKG. Dans ce schéma, chaque autorité possède sa propre clé maître (MSK) et participe à l'exécution de l'algorithme `Extraction`.

1.4.1 Émission sécurisée des clés

Comme nous l'avons mentionné, le schéma ESC [LBD+04] (*Émission Sécurisée des Clés*) est un IBE qui possède plusieurs autorités. Soient $s_0 \in \mathbb{Z}_p$ la clé maître du PKG, et $s_1, \ldots, s_n$ la clé maître de KPA_1, ..., KPA_n respectivement. La clé publique du système dans ce cas est : $Y = s_0 s_1 \ldots s_n P$; de plus, chaque autorité i possède sa propre clé publique $P_i = s_i P$ pour $0 \leqslant i \leqslant n$. Pour calculer Y, chaque autorité calcule $Y_i = s_i Y_{i-1}$ et l'envoie à l'autorité $i+1$, tel que $Y_0 = s_0 P = P_0$ (voir la figure 1.3). Une fois le calcul terminé et publié, chaque autorité est capable de vérifier l'exactitude des Y_i en vérifiant l'égalité suivante :

$$e(Y_i, P) \stackrel{?}{=} e(Y_{i-1}, P_i) \ .$$

En effet, cette valeur équivaut : $e(Y_i, P) = e(s_i Y_{i-1}, P) = e(Y_{i-1}, s_i P) = e(Y_{i-1}, P_i) = e(Y_{i-1}, P)^{s_i}$, et seul KPA_i est capable de générer Y_i qui satisfait ce test.
Après la phase `Initialisation`, il faudra redéfinir la nouvelle version de l'algorithme `Extraction`. Contrairement à BF-IBE, dans cette version l'utilisateur et

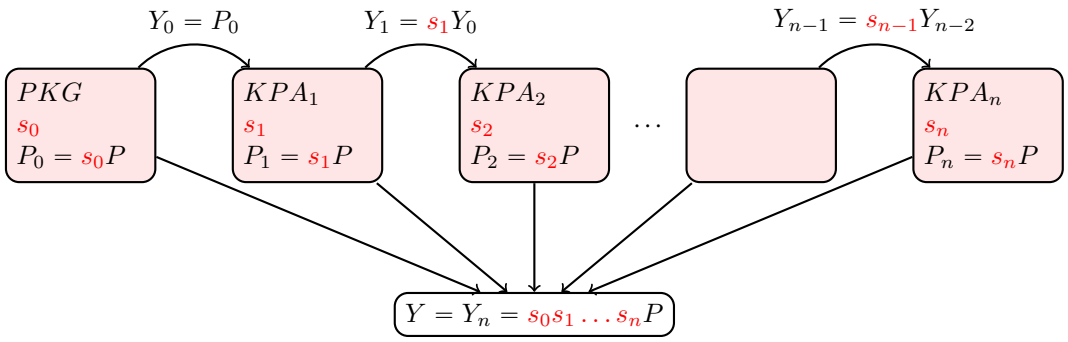


FIGURE 1.3 – Génération de la clé publique du système dans le schéma d'ESC

les KPA participent aussi à l'exécution de cet algorithme. Soit $\mathcal{H}_3 : \mathbb{G}_T \to \mathbb{Z}_p$ une fonction de hachage qui s'ajoute désormais aux paramètres publics, l'algorithme `Extraction` est défini ainsi :

`Extraction`

- Par l'utilisateur :
  - Générer une clé temporaire $x \xleftarrow{\$} \mathbb{Z}_p$;
  - Calculer $X = xP$;
  - Envoyer id, X au PKG.
- Par le PKG : Il valide l'authenticité de l'utilisateur puis calcule :
  - La clé publique : $Q_{id} = \mathcal{H}(id, PKG, KPA_1, \dots, KPA_n)$;
  - La clé partielle masquée de l'utilisateur :

$$Q_0 = \mathcal{H}_3(e(s_0 X, P_0)) s_0 Q_{id} \; ; \tag{1.7}$$

  - Signature pour vérifier l'intégrité : $Sign(Q_0) = s_0 Q_0$;
  - Envoie $(Q_0, Sign(Q_0))$ au KPA_1.
- Par KPA_i :
  - Vérifie la signature : $e(Sign(Q_{i-1}), P) \stackrel{?}{=} e(Q_{i-1}, P_{i-1})$;
  - Calcule :

$$Q_i = \mathcal{H}_3(e(s_i X, P_i)) s_i Q_{i-1} \; ; \tag{1.8}$$

  - Signe : $Sign(Q_i) = s_i Q_i$;
  - Envoie $(Q_i, Sign(Q_i))$ au KPA_{i+1} sauf pour le cas $i = n$, dans ce cas le KAP_n envoie les clés à l'utilisateur.
- Par l'utilisateur : L'utilisateur reçoit $Q_n, Sign(Q_n)$ puis :
  - Vérifie la signature : $e(Sign(Q_n), P) \stackrel{?}{=} e(Q_n, P_n)$;
  - Calcule la clé privée :

$$S_{id} = \frac{Q_n}{\prod_{i=0}^{n} \mathcal{H}_3(e(P_i, P_i)^x)} \; ; \tag{1.9}$$

— Vérifie la clé privée : $e(S_{id}, P) \stackrel{?}{=} e(Q_{id}, Y)$.

L'utilisateur commence par choisir une clé secrète temporaire x, cette clé va servir à la fin de l'algorithme pour extraire (démasquer) la clé secrète Q_n. Chaque autorité i génère sa part de la clé secrète de l'utilisateur notée Q_i, cette clé (Q_i) est protégée par un masque $\mathcal{H}_3(e(s_i X, P_i))$ que seul l'utilisateur ou l'autorité i peut calculer. En effet, nous avons : $e(s_i X, P_i) = e(s_i x P, P_i) = e(s_i P, P_i)^x = e(P_i, P_i)^x$. Ainsi, par les équations 1.8 et 1.7, nous obtiendrons :

$$Q_n = \underbrace{\mathcal{H}_3(e(s_0 X, P_0)) \dots \mathcal{H}_3(e(s_n X, P_n))}_{\text{le masque}} \underbrace{s_0 \dots s_n Q_{id}}_{S_{id}} . \tag{1.10}$$

Par conséquent, seul l'utilisateur légitime (ou la réunion de toutes les autorités) est capable de démasquer Q_n et d'en extraire sa clé privée S_{id} (équation 1.9).

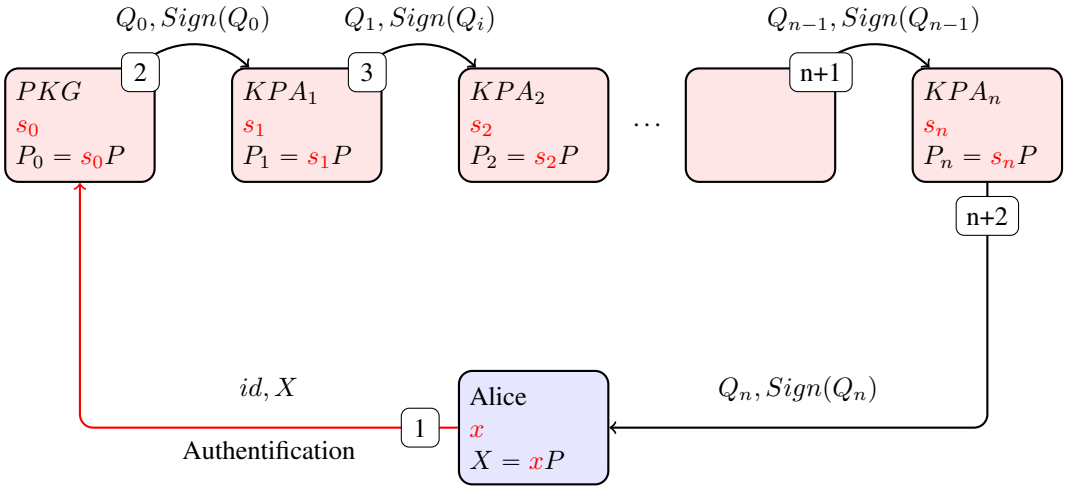


FIGURE 1.4 – L'algorithme `Extraction` dans un schéma d'ESC

Étant donné que les échanges se font sur un canal public (voir la figure 1.4), il est important de protéger les communications contre les attaques de *l'homme du milieu*. Pour ce faire, chaque autorité signe son envoi envers sa voisine pour authentifier la donnée envoyée.
Enfin, l'utilisateur calcule sa clé privée S_{id} et le dernier test permet de vérifier l'exactitude de l'ensemble des calculs, et que la clé privée est bien composée de $s_0 \dots, s_n$. En effet, sans ce test, il est possible qu'une des autorités corrompues utilise une fausse clé maître pour perturber le système.

1.4.2 Conclusion

Dans ce schéma, nous distinguons le rôle du PKG, qui s'occupe d'authentifier les futurs utilisateurs et qui participe à la génération de leur clé, et le rôle des KPA, qui s'occupent uniquement de la protection de la clé privée de l'utilisateur.
Bien que ce schéma résolve le problème de centralisation, ainsi que celui de l'autorité de séquestre, nous remarquons que le problème de révocation reste ouvert ; de plus, comme l'expliquent Xu *et al.* [XZQ05] dans ce schéma, il ne suffira pas de s'authentifier uniquement auprès du PKG. En effet, un PKG corrompu est capable de générer une fausse valeur x' et $X' = x'P$ et lancer l'exécution de l'algorithme `Extraction` avec ces valeurs au lieu de x et X de l'utilisateur légitime.
Dans le chapitre suivant, nous allons présenter une version améliorée de ce schéma, dans laquelle nous traitons le cas du PKG corrompu, la révocation, la mise à jour des clés, ainsi que la simplification des calculs.

2 | Amélioration d'IBE

Sommaire

2.1 Introduction

Dans le chapitre précédent, nous avons présenté le schéma IBE de Boneh et Franklin, ainsi que le schéma ESC pour la protection des clés. La combinaison de ces deux technologies donne naissance à un schéma décentralisé où, seul l'utilisateur est capable de construire sa clé. En même temps, dans le cadre de la procédure "e-discovery", il est toujours possible à une autorité judiciaire d'obtenir la clé privée d'une entité en combinant le pouvoir du PKG et celui des KPAs.

Bien que le schéma que nous avons présenté (ainsi que d'autres) résolve le problème de centralisation, sont souvent négligés le problème de révocation et la mise à jour des clés (*rekeying*). En effet, dans le schéma précédent, pour éviter l'usurpation d'identité par le PKG, il est nécessaire d'authentifier chaque utilisateur auprès de toutes les autorités (PKG et KPAs). Étant donné que l'authentification est généralement une procédure longue, car elle nécessite un contact direct avec l'autorité concernée, il est préférable de minimiser de manière optimale ce besoin.

Dans ce chapitre, nous présentons une nouvelle architecture de gestion des clés qui combine les avantages d'un schéma PKI et IBE [DT16]. Rappelons qu'une infrastructure de gestion de clés décrit trois phases principales :

- **l'enrôlement** : correspond à l'inscription des utilisateurs ;
- **la mise à jour des clés** : correspond à la réinitialisation de toutes les clés privées après la fin de leurs dates de validité ;
- **la révocation** : vise à annuler l'enregistrement d'un utilisateur, et donc l'empêcher de déchiffrer les futurs messages chiffrés.

2.2 Les paramètres du système

Dans l'architecture que nous proposons, nous distinguons deux types d'autorités :

- Le PKG statique (PKG_0) : il s'agit d'une autorité gérée par un officier de sécurité, qui participe uniquement à la phase d'enrôlement. Son rôle est d'authentifier les utilisateurs, et de leur fournir une clé privée partielle.
- Le PKG dynamique (PKG_1) : il s'agit d'une autorité qui participe à la protection de la clé privée de l'utilisateur, et s'occupe de la mise à jour des clés. Pour cette raison, le PKG dynamique met à jour sa clé maître plus régulièrement que le PKG statique, ainsi sans avoir un impacte sur toutes les clés des utilisateurs.

Sans perte de généralité, nous allons considérer un seul PKG dynamique. L'algorithme d'initialisation prend en entrée un paramètre de sécurité k, et renvoie les paramètres publics PP et les deux clés maîtres s_0 du PKG_0 et s_1 du PKG_1.

`Initialisation`
- Entrée : Le paramètre de sécurité k ;
- Choisir un nombre premier p de longueur k ;
- Choisir la clé maître du PKG statique : $s_0 \in \mathbb{Z}_p$;
- Choisir la clé maître du PKG dynamique : $s_1 \in \mathbb{Z}_p$;
- Choisir une fonction bilinéaire admissible $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$;
- Choisir une fonction de hachage $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{G}_1$;
- Choisir une fonction de hachage $\mathcal{H}_2 : \mathbb{G}_T \to \mathcal{M}$, où $\mathcal{M}$ est l'ensemble des messages ;
- Choisir deux générateurs : $P \in \mathbb{G}_0$ et $P' \in \mathbb{G}_1$;
- Calculer la clé publique du PKG statique : $P_0 = s_0 P$;
- Calculer les clés publiques du PKG dynamique : $P_1 = s_1 P$ et $P'_1 = s_1 P'$;
- Calculer la clé publique du système : $Y = s_0 s_1 P$;
- Sortie : $PP = (\mathbb{G}, \mathbb{G}_T, e, P, Y, \mathcal{H}_1, \mathcal{H}_2)$, $MSK = s_0, s_1$.

La clé publique Y est calculée par le PKG dynamique à partir de la clé P_0, puisque $Y = s_0 s_1 P = s_1 P_0$. Bien entendu, il est toujours possible à quiconque de vérifier ce calcul en testant :

$$e(Y, P') \stackrel{?}{=} e(P_0, P'_1) \ . \tag{2.1}$$

En effet, toujours grâce à la bilinéarité de e, nous avons : $e(Y, P') = e(s_0 s_1 P, P') = e(s_0 P, s_1 P') = e(P_0, P'_1)$.

2.3 L'extraction de la clé privée

Pour l'algorithme `Extraction`, nous distinguons la phase d'enrôlement où l'utilisateur obtient sa première clé, et la phase de mise à jour, où la crypto-période de la clé est achevée.

2.3.1 L'enrôlement

La phase d'enrôlement commence par une authentification auprès du PKG statique, qui valide l'identité de l'entité concernée, et génère sa clé partielle S_p. Cette clé (privée) sera utilisée uniquement pour authentifier l'utilisateur auprès du PKG dynamique, et générer la clé privée.

`Étape 1`. Par le PKG statique :
- Authentification et validation de l'identité de l'utilisateur ;

— Calculer la clé publique de l'utilisateur : $Q_{id} = \mathcal{H}_1(id) \in \mathbb{G}_1$;
— Calculer la clé partielle de l'utilisateur : $S_p = s_0 Q_{id} \in \mathbb{G}_1$;
— Sortie : Q_{id}, S_p.

L'utilisateur reçoit sa clé partielle en passant par un canal sécurisé, et vérifie son exactitude grâce au test 2.2. Ce test permet de valider que le PKG statique a bien utilisé sa clé maître s_0 pour calculer la clé partielle. Une fois le test validé, l'utilisateur génère un masque t qu'il va garder jusqu'à la fin de la phase d'enrôlement. Ce masque permet de protéger la clé partielle S_p. Enfin, l'utilisateur envoie la clé partielle masquée $S_{p,t}$, ainsi que les clés publiques P_t, Q_t pour valider les tests 2.3 et 2.4 sur un canal public.

`Étape 2`. Par l'utilisateur :
— L'utilisateur valide si ce test est satisfait :

$$e(P, S_p) \stackrel{?}{=} e(P_0, Q_{id}) \; ; \tag{2.2}$$

— L'utilisateur choisit un masque temporaire $t \in \mathbb{Z}_p$;
— Calcule la clé partielle masquée : $S_{p,t} = tS_p \in \mathbb{G}_1$;
— Calcule la clé publique masquée : $Q_t = tQ_{id} \in \mathbb{G}_1$;
— Calcule : $P_t = tP \in \mathbb{G}_0$;
— Envoie : $id, S_{p,t}, Q_t, P_t$ au PKG dynamique.

Le PKG statique reçoit la requête de l'utilisateur. D'abord, il vérifie que $Q_t = tQ_{id}$ grâce au test 2.3, ceci confirme que Q_t porte l'identité id. Ensuite, il vérifie que $S_{p,t} = s_0 Q_t$ grâce au test 2.4 qui confirme que l'utilisateur a été validé par le PKG statique, et qu'il a obtenu sa clé partielle. Après la validation de ces deux tests, le PKG statique calcule la clé privée masquée : S_t et l'envoie à l'utilisateur sur un canal public.

`Étape 3`. Par le PKG dynamique :
— Le PKG dynamique valide si ces deux tests sont satisfaits :

$$e(P, Q_t) \stackrel{?}{=} e(P_t, Q_{id}) \; ; \tag{2.3}$$

$$e(P, S_{p,t}) \stackrel{?}{=} e(P_0, Q_t) \; ; \tag{2.4}$$

— Calcule la clé privée masquée : $S_t = s_1 S_{p,t} = ts_0 s_1 Q_{id}$;
— Envoie : S_t à l'utilisateur.

L'utilisateur reçoit sa clé masquée, il vérifie grâce au test 2.5 que la clé reçue vaut bien $ts_0s_1Q_{id}$. Après la validation du test, l'utilisateur extrait sa clé privée S_{id} en démasquant la clé reçue.

`Étape 4`. Par l'utilisateur :

— L'utilisateur valide si ce test est satisfait :

$$e(tY, Q_{id}) \stackrel{?}{=} e(P, S_t) \ ; \tag{2.5}$$

— Calcule la clé secrète : $S_{id} = \frac{1}{t} S_t$.

À la fin de la phase d'enrôlement l'utilisateur possède : la clé publique Q_{id}, la clé privée S_{id} et la clé partielle S_p. La figure 2.1 résume les échanges entre l'utilisateur et les deux PKG.

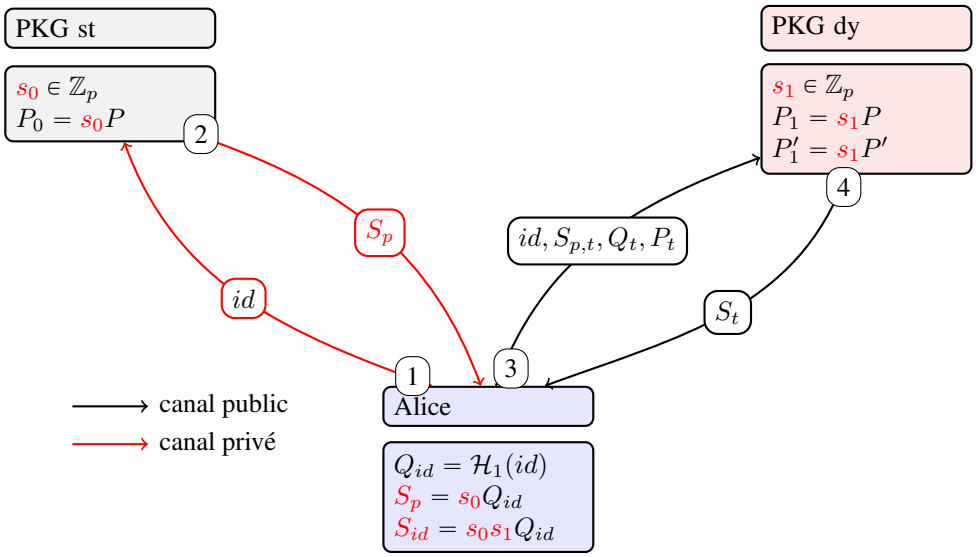


FIGURE 2.1 – Récapitulatif de la phase d'enrôlement

2.3.2 La mise à jour des clés

La phase de mise à jour des clés, ou "*rekeying*", est un composant indispensable dans un schéma de gestion des clés. En effet, pour des mesures de sécurité, les clés cryptographiques possèdent souvent une date d'expiration, ce que nous appelons une *crypto-période*. À la fin de la crypto-période, l'utilisateur doit lancer le programme de *rekeying* afin d'obtenir une nouvelle clé. Souvent, dans une PKI ou

dans les schémas IBE, cette phase est considérée identique à la phase d'enrôlement. Cependant, dans le schéma que nous présentons dans ce chapitre, nous avons essayé de distinguer ces deux étapes. Étant donné que les communications passent sur un canal public avec le PKG dynamique, nous proposons que ce dernier soit le seul à intervenir pour mettre à jour les clés des utilisateurs. Pour ce faire, le PKG dynamique génère une nouvelle clé maître s'_1 et lance l'algorithme de mise à jour. Les utilisateurs vont ensuite se servir de la clé partielle S_p et exécuter les étapes 3 et 4 de la phase d'enrôlement pour obtenir la nouvelle clé secrète S_{id}.

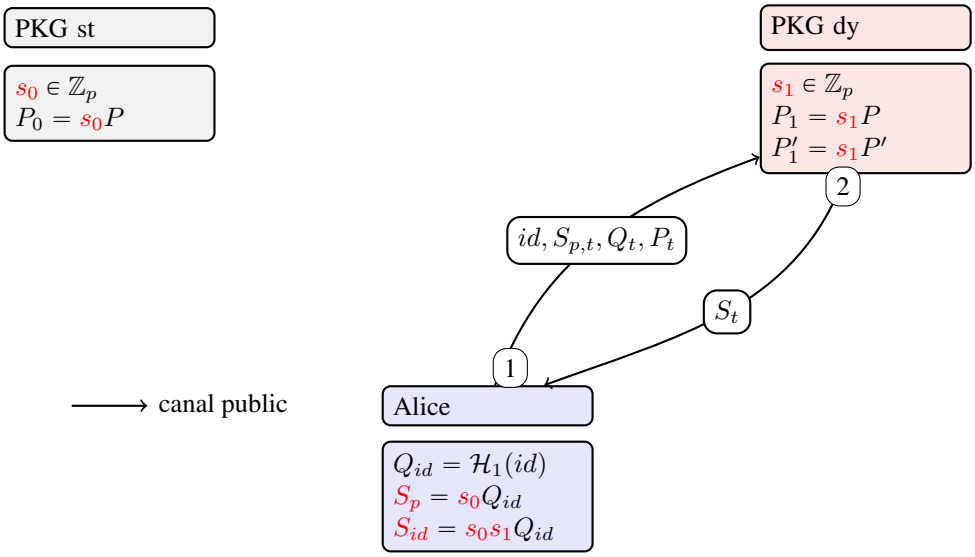


FIGURE 2.2 – Récapitulatif de la phase de mise à jour des clés

La mise à jour de la clé s_0 impacte tous les paramètres publics qui en dépendent, en particulier la clé publique du PKG dynamique ($P_1 = s_1 P$), et la clé publique du système $Y = s_0 s_1 P$. Pour calculer la nouvelle clé Y' nous avons :

$$Y' = \frac{s'_1}{s_1} Y = s_0 s'_1 P \ .$$

Il est toujours possible à quiconque de vérifier ce calcul grâce au test de l'équation 2.1.

2.4 Analyse de sécurité

La clé privée d'un utilisateur étant calculée de manière coopérative par plusieurs PKGs (statiques et dynamiques), la confidentialité de la clé privée de l'utilisateur est préservée si au moins un seul PKG statique reste honnête. Seul l'utilisateur légitime connaissant la clé de masquage t peut extraire et calculer sa clé privée.

Dans le cas de plusieurs autorité statiques, l'utilisateur sera amené à s'authentifier auprès de chacune de celles-ci pour obtenir sa clé partielle finale, cette condition est obligatoire afin de se protéger contre l'usurpation d'identité [XZQ05].
Le schéma proposé émet une véritable clé privée basée sur l'identité. Il peut donc être combiné avec d'autres schémas basés sur l'identité comme la signature basée sur l'identité, le chiffrement basé sur les attributs, etc.
Nous définissons IBE2 comme étant une extension des schémas IBE ordinaires. Ainsi les modèles de sécurité IND-ID-CPA et IND-ID-CCA (voir la page 102) restent similaires saufs à la phase 1 et 2 où l'attaquant peut accéder à la fois à la clé privée S_{id} et à la clé partielle S_p. Il suffit de remarquer qu' IBE classique est un cas particulier d'IBE2 dans ces deux modèles. En effet, dans le cadre d'IBE classique $S_p = Q_{id}$ avec $s_0 = 1$. Ainsi, si l'attaquant IND-ID-CPA ou IND-ID-CCA réussit son attaque sur IBE2 alors, il est toujours possible d'utiliser la même attaque sur IBE classique.

2.5 La révocation des clés

2.5.1 La problématique

Entre deux crypto-périodes T_i et T_{i+1} il est toujours possible qu'un utilisateur quitte le système pour plusieurs raisons : fin de contrat, licenciement, perte de clé, matériel défectueux ou compromis, etc. Il est donc important de définir une stratégie de révocation des clés. Dans une PKI, la révocation est gérée par le service d'annuaire, qui s'occupe d'ajouter les certificats révoqués dans la CRL (la liste des certificats révoqués), cette liste est signée par l'autorité de certification afin de prouver son intégrité. Au fur et à mesure, la CRL ne cesse d'augmenter, ce qui provoque une latence pendant la validation des certificats. À cause de ce problème, une grande partie des clients PKI effectuent leur validation de manière incomplète.
Dans le schéma BF-IBE, il n'existe aucun moyen de communiquer aux expéditeurs qu'une identité a été révoquée, car seule la clé publique du PKG et l'identité du destinataire sont nécessaires pour chiffrer un message. Avec un tel mécanisme, la mise à jour régulière des clés privées semble être la seule solution au problème de révocation. Cela signifie que tous les utilisateurs, que leurs clés aient été révoquées ou non, doivent régulièrement entrer en contact avec le PKG, prouver leur identité et obtenir de nouvelle clé privée. Le PKG doit rester en ligne pour toutes ces transactions, et un canal sécurisé doit être établi entre le PKG et chaque utilisateur pour transmettre la clé privée. Prenant en compte l'évolutivité du déploiement d'IBE, nous pouvons déduire que, pour un très grand nombre d'utilisateurs, cela peut vite devenir une impasse.

2.5.2 Les solutions existantes

Le problème de révocation est un problème bien étudié dans le cadre d'une PKI traditionnelle, et plusieurs solutions ont été proposées [Mic08, ALO98, Gen03, Goy07]. Cependant, les articles sur IBE négligent souvent les mécanismes de révocation. Hanaoka *et al.* [HHSI05] proposent un moyen pour les utilisateurs de renouveler périodiquement leurs clés privées sans interagir avec le PKG. Le PKG affiche publiquement les informations de mise à jour des clés, ce qui est beaucoup plus pratique. Cependant, chaque utilisateur doit posséder un dispositif matériel inviolable.
L'autre solution proposée consiste à combiner IBE avec un schéma de partage de secret [BDTW01, LQ03]. Dans ce cadre, il existe un tiers de confiance, appelé *médiateur*, qui détient les partages des clés privées de tous les utilisateurs et qui les aide à déchiffrer chaque cryptogramme. Ainsi, si une identité est révoquée, le médiateur est invité à cesser d'aider l'utilisateur.

2.5.3 La solution proposée

Pour surmonter le problème de révocation, nous proposons de combiner la solution utilisée dans une PKI qui est la CRL, et les avantages qu'offre le schéma IBE. L'idée consiste à concevoir une liste des identités révoquées que nous appelons IRL (en anglais : *Identity Revocation List*). Cette liste doit respecter les conditions suivantes :

- effacer la liste après chaque crypto-période. Ainsi, nous pouvons révoquer les entités en temps réel sans augmenter constamment l'IRL. Étant donné que les PKGs sont capables de restituer les clés, il n'est donc nul besoin de garder une sauvegarde des clés révoquées ;
- modifier l'IRL n'est accessible qu'à l'officier (ou les officiers) de sécurité, et elle est signée par les deux PKG afin de prouver son intégrité ;
- stocker uniquement les identités révoquées ;
- choisir la crypto-période de façon à rendre l'IRL vide la plupart du temps, ainsi elle n'affectera pas la fluidité du trafic réseau.

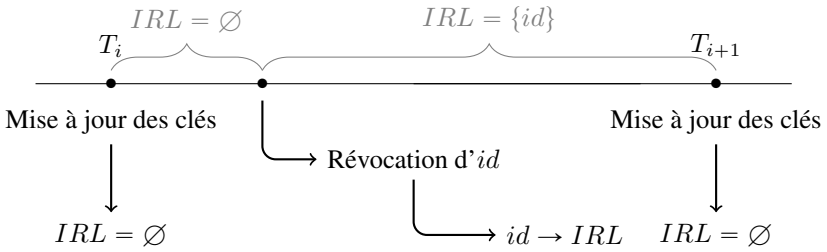


FIGURE 2.3 – La politique de révocation

2.6 Récapitulatif

Dans ce chapitre, nous avons présenté une architecture IBE qui apporte des solutions sur les problématiques qui touchent ce genre de schémas, en particulier : la décentralisation, l'autorité de séquestre, et la révocation. Nous avons choisi d'améliorer la solution proposée par Byoungcheon *et al.* sur la sécurisation des clés. La valeur ajoutée consiste à simplifier la procédure d'enrôlement en minimisant les calculs.
Le masque utilisé par Byoungcheon *et al.* pour protéger la clé privée est composé de plusieurs calculs de fonctions de hachage (équation 1.10), tandis que le masque dans le schéma présenté dans ce chapitre consiste en une seule valeur t générée par l'utilisateur.
Nous avons également démontré que la signature des échanges pour la phase d'enrôlement n'est pas nécessaire. En effet, grâce au différents tests que nous avons présentés, nous constatons que les clés sont auto-signées.
Il faut noter également la différence entre les rôles du PKG statique, qui est une autorité isolée (hors-ligne), et du PKG dynamique, qui est quant à lui connecté au réseau. Cette distinction a trois avantages :

- la protection du système contre les différentes attaques informatiques, étant donné que le PKG statique est toujours hors-ligne ;
- l'authentification de l'utilisateur auprès du PKG statique est simplifiée, il suffit de démontrer qu'il possède la clé partielle générée par le PKG statique ;
- la simplification de la mise à jour des clés avec l'intervention du PKG statique uniquement.

Enfin, pour résoudre le problème de révocation, nous avons intégré dans le système une liste des certificats révoqués (IRL) ; cette solution permet d'obtenir une révocation en temps réel sans attendre la fin de la crypto-période.

3 | Conclusion et perspectives

Sommaire

Dans ce manuscrit nous avons exposé deux problématiques qui touchaient la cryptographie symétrique et asymétrique. L'objectif de ce travail était d'analyser ces problématiques, montrer une partie des travaux réalisés par la communauté, et enfin présenter notre valeur ajoutée pour traiter ces problèmes. Dans ce chapitre de conclusion, nous allons rappeler brièvement les deux sujets :

— Le masquage pour lutter contre les attaques par canaux auxiliaires et les attaques par injection de fautes ;
— Les infrastructures de gestion des clés.

Nous mettons l'accent sur les lignes importantes que nous avons traitées, et les améliorations possibles que nous prévoyons pour après la thèse.

3.1 La protection contre les attaques par canaux auxiliaires et les attaques par injection de faute

3.1.1 Rappel

Le premier sujet concerne la cryptographie symétrique, nous avons traité la vulnérabilité des systèmes de chiffrement, en particulier du standard du chiffrement symétrique, l'AES, face aux attaques par canaux auxiliaires (SCA [1]) et les attaques par injection de faute (FIA [2]), et nous avons montré qu'il est primordial de masquer les données sensibles durant le chiffrement pour lutter contre ces attaques.
Nous avons vu que l'implémentation d'un système de chiffrement sur un composant électronique, (par exemple : une carte à puce ou une carte FPGA), est souvent plus sensible aux attaques comparée à une implémentation sur un ordinateur classique. La simple mesure de la consommation électrique du composant pendant l'exécution d'une opération de chiffrement (ou déchiffrement) peut révéler des informations sur les données traitées. D'un autre côté, par le principe de KERCKHOFFS, il est fortement recommandé de publier les algorithmes de chiffrement et les protocoles utilisés. Un attaquant ayant un accès physique au composant électronique, connaissant son architecture et les algorithmes qu'il exécute, est capable de prédire la clé secrète contenue dans ce composant.

3.1.2 Le masquage

L'implémentation des algorithmes cryptographiques nécessite un soin particulier afin d'assurer une bonne protection contre les attaques de type SCA et FIA. En

1. *Side Channel Attacks*
2. *Fault Injection Attack*

effet, **la complexité des opérations ne doit pas dépendre des données sensibles**. Pour éviter ce genre de situations, il est conseillé de masquer les valeurs sensibles avant de les traiter.
La conception d'une fonction de masquage implique la modification de la structure du système de chiffrement ciblé. Pour chaque transformation f qui compose ce système de chiffrement, il faut construire une transformation alternative f' telle que : $f'(\texttt{mask}(x)) = \texttt{mask}(f(x))$.
Nous avons vu dans la première partie plusieurs schémas de masquage, et nous avons remarqué qu'ils ne protègent pas tous contre les deux attaques SCA et FIA simultanément. Le masquage classique, le masquage basé sur le partage du secret et le masquage basé sur le produit scalaire permettent tous de masquer l'addition et la multiplication et protéger ainsi contre SCA. D'un autre côté, le masquage de la multiplication dans DSM est un problème ouvert. Nous avons remarqué également que la complexité algorithmique des opérations (notamment le masquage de la multiplication) est quadratique, ceci entraîne un coût non négligeable pour les composants à faibles ressources.
Pour surmonter ces obstacles, et construire un nouveau schéma de masquage, nous avons défini les objectifs suivants :

— masquer les deux opérations, addition et multiplication ;
— détecter (et corriger si besoin) les erreurs potentielles pendant l'exécution de l'algorithme de chiffrement ou déchiffrement ;
— minimiser le coût des opérations.

Pour atteindre ces objectifs nous avons utilisé la même structure que celle du schéma ODSM. Ce dernier, étant basé sur les codes LCD, sa conception possède une structure proche de celle des codes correcteurs. De plus, il permet de masquer et rafraîchir le masque de manière simple et rapide. Cependant, nous avons modifié la représentation des données traitées afin d'augmenter le nombre de manipulations possibles. Dans le schéma original les opérations se font sur un espace vectoriel, tandis que le nouveau schéma traite les données dans un corps fini. Étant donné que la multiplication est commutative dans un corps fini, cette propriété nous avait permis de concevoir un masquage de complexité linéaire pour la multiplication. De plus, avec cette nouvelle représentation, il est possible également de détecter les erreurs potentielles de manière plus efficace. En effet, il n'est plus besoin de garder en mémoire le masque utilisé pour détecter les erreurs. Dans la nouvelle structure, selon le choix des paramètres $\vec{g}$ et $\vec{h}$ il est possible de détecter et même corriger les erreurs sans connaître le masque utilisé r.
Les paramètres $\vec{g}$ et $\vec{h}$ que nous avons utilisés dans l'exemple 11 de la page 87 permettent de concevoir un code correcteur $\mathcal{K}$ de distance minimale 3 :

$$\mathcal{K} = \left\{ \vec{z} = \vec{c} \oplus \vec{d} \mid \forall (\vec{c}, \vec{d}) \in \mathcal{C} \times \mathcal{D} \right\} \subset \mathbb{F}_{2^8}^n \ .$$

Ainsi, pour $n = 3$ il est possible de détecter jusqu'à 2 erreurs. En général, pour une

longueur n petite, il est possible de faire une recherche exhaustive des paramètres $\vec{g}$ et $\vec{h}$ pour obtenir un code $\mathcal{K}$ d'une capacité de détection optimale.

3.1.3 Perspectives

Il sera intéressant pour la suite d'étudier le choix des paramètres $\vec{g}$ et $\vec{h}$ pour des longueurs plus grandes, trouver des propriétés capables de générer facilement les codes $\mathcal{C}$ et $\mathcal{D}$ optimaux de ce masquage. L'objectif sera de trouver, pour un paramètre de sécurité d, deux vecteurs $\vec{g}$ et $\vec{h}$ qui satisferont les critères de ce schéma de masquage, et qui forment un code $\mathcal{K}$ dont la capacité de détection d'erreurs est maximale pour une longueur n minimale.
Nous avions démontré que ce masquage protège bien contre les attaques monovariées. Cependant, nous n'avons pas de preuve en ce qui concerne les attaques multivariées, en particulier pour une longueur n assez grande. Il sera donc intéressant d'étudier le comportement de ce masquage face à ces attaques, et voir s'il existe un seuil auquel il peut protéger.
Une autre piste à explorer, concerne l'utilisation de ce masquage dans d'autres systèmes de chiffrement symétrique. La fonction de substitution du 3DES par exemple utilise deux représentations différentes ($\mathbb{F}_2^6$ en entrée, et $\mathbb{F}_2^4$ en sortie). Il sera donc intéressant de trouver une généralisation de ce masquage pour les systèmes de chiffrement qui utilisent des représentations multiples.

3.2 Les infrastructures de gestion des clés

3.2.1 Rappel

Le deuxième sujet que nous avons traité concerne la cryptographie asymétrique. Pour sécuriser les communications entre un ensemble d'utilisateurs, il est préférable d'utiliser des systèmes de chiffrement symétriques. Ces derniers, sont souvent plus rapides et efficaces pour chiffrer une grande quantité de données. D'un autre côté, il est nécessaire également d'utiliser des systèmes de chiffrement asymétrique pour assurer l'échange de clés. Dans la deuxième partie de ce manuscrit, nous avons présenté les problématiques liées à la gestion des clés dans un système asymétrique. La mise en place d'une architecture de gestion de clés est souvent confrontée à plusieurs obstacles tels que : l'usurpation d'identité, le problème de révocation, la centralisation, la sauvegarde des clés privées ...
Ces problématiques remontent au début des années quatre-vingt. Il a fallu trouver un moyen pour authentifier les clés afin de se protéger contre l'attaque de l'homme au milieu (voir la section 1.4.2 page 13). Shamir a proposé alors le concept de la cryptographie basée sur l'identité (IBE). Ce concept consiste principalement

à intégrer l'identité dans la clé publique. La conception d'un tel crypto-système étant un problème ouvert, les constructeurs ont adopté la solution PKI. Celle-ci consiste à ajouter une autorité de certification (CA) qui assurera l'authenticité des clés.

Plus tard, en 2001, Boneh et Franklin ont proposé le premier système de chiffrement efficace de type IBE, ce dernier, basé sur le problème *Diffie-Hellman* bilinéaire, a résolu le problème d'usurpation d'identité. Cette solution utilise l'accouplement bilinéaire sur un groupe de points d'une courbe elliptique. Le schéma est composé d'une seule autorité (PKG) qui assure la génération des clés privées à partir d'une chaîne de caractères arbitraires. Le PKG possède une clé maître qui permet de faire cette extraction de clé secrète à partir d'une clé publique, ce qui signifie que la sécurité du système repose entièrement sur cette clé maître.

Le schéma initial de Boneh et Franklin (BF-IBE) avait également plusieurs inconvénients :

1. Le problème de centralisation : Dans ce schéma, le PKG détient tout le pouvoir, il est capable de produire à lui seul les clés secrètes de tous les utilisateurs, ce qui lui permet de déchiffrer toutes les communications, les falsifier, et même usurper les identités.
2. Le problème de révocation : Dans le schéma BF-IBE, le destinataire doit interroger le PKG à chaque communication pour obtenir sa clé de déchiffrement. Bien que cette option empêche les utilisateurs révoqués de déchiffrer, elle oblige le PKG de rester en écoute pour valider chaque échange entre utilisateurs, et établir à chaque fois un canal sécurisé pour communiquer la clé de déchiffrement.

Plusieurs auteurs ont proposé des solutions pour résoudre le problème de centralisation. Al-Riyami et Paterson [AP03] ont construit CL-PKC (*Certificatless Public Key Cryptography*) qui permet à une entité valide de construire ses propres clés privées et publiques.

Lee *et al.* [LBD+04] ont présenté un schéma IBE non centralisé. Le schéma définit plusieurs autorités de protection des clés appelées KPA (*Key Privacy Authority*) en plus de l'autorité standard qui est le PKG. Dans ce schéma, chaque autorité possède sa propre clé maître (MSK) et participe à l'exécution de l'algorithme d'extraction de la clé privée des utilisateurs.

Bien que ces systèmes, et d'autres ont résolu le problème de centralisation, le problème de l'autorité de séquestre (*Key Escrow Problem*) [XZQ05] qui consiste en d'autres termes à appliquer la procédure *e-discovery* (voir l'exigence 5 de la section 1.5.2) persiste encore.

3.2.2 Les solutions proposées et les perspectives

Pour résoudre les problématiques que nous avons citées, nous avons présenté dans le chapitre 2 un schéma qui combine les caractéristiques suivantes :

- un schéma décentralisé où seul l'utilisateur est capable de reconstruire sa clé privée ;
- il est possible de reconstruire la clé privée d'un utilisateur en combinant les clés maîtres des autorités, ainsi respecte la procédure "e-discovery" sans utiliser un matériel de stockage sécurisé comme un HSM ;
- la mise à jour des clés se fait sur un canal public ;
- la révocation des utilisateurs moins coûteuse grâce à l'IRL que l'on vide à chaque mise à jour des clés.

Cette solution optimise également les calculs effectués durant la phase d'enrôlement, et assure l'intégrité des échanges entre l'utilisateur et les différentes autorités. IBE-2 est un schéma capable de s'adapter avec d'autres systèmes basés sur l'identité comme le chiffrement/signature basé(e) sur les attributs. Il sera intéressant de combiner IBE2 avec les solutions existantes afin d'apporter une meilleur gestion de clés.

Table des figures

Liste des tableaux

Index alphabétique

Publications et conférences :

[CDGT18] Claude Carlet and Abderrahman Daif and Sylvain Guilley and Cédric Tavernier. Polynomial direct sum masking to protect against both SCA and FIA. *Journal of Cryptographic Engineering*, Aug 2018.

[DT16] Abderrahman Daif and Cédric Tavernier. An efficient certificateless key management architecture to solve IBE and PKI issues. In *Fifteenth International Workshop on Algebraic and Combinatorial Coding Theory*, pages 111-116, 2016.

[CDD+15] Claude Carlet and Abderrahman Daif and Jean-Luc Danger and Sylvain Guilley and Zakaria Najm and Xuan Thuy Ngo and Thibault Porteboeuf and Cédric Tavernier. Optimized linear complementary codes implementation for hardware trojan prevention. In *European Conference on Circuit Theory and Design, ECCTD 2015, Trondheim, Norway*, pages 1-4, 2015.

[ABB+15] Mickael Avril and Laurie Basta and Laurent Bouillet and Abderrahman Daif and Gregory Landais and Cédric Tavernier. Identity based cryptography for smart-grid protection. In *Proceedings of 9th international conference on computer engineering and applications*, pages 1-10, 2015.

Bibliographie

[ABB+15] Mickael Avril, Laurie Basta, Laurent Bouillet, Abderrahman Daif, Gregory Landais, and Cédric Tavernier. Identity based cryptography for smart-grid protection. In *Proceedings of 9th international conference on computer engineering and applications*, pages 1–10, 2015. Cited on pages 19 and 133.

[AK97] Ross J. Anderson and Markus G. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, pages 125–136, 1997. Cited on page 40.

[ALO98] William Aiello, Sachin Lodha, and Rafail Ostrovsky. Fast Digital Identity Revocation (Extended Abstract). In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 137–152, 1998. Cited on page 118.

[ANS16] ANSSI. Recommandations de sécurité relatives à TLS, 2016. https://www.ssi.gouv.fr/uploads/2016/09/guide_tls_v1.1.pdf. Cited on page 6.

[AP03] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless Public Key Cryptography. In *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, pages 452–473, 2003. Cited on pages 106 and 125.

[BCC+14] Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. Orthogonal Direct Sum Masking - A Smartcard Friendly Computation Paradigm in a Code, with Builtin Protection against Side-Channel and Fault Attacks. In *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece,*

June 30 - July 2, 2014. Proceedings, pages 40–56, 2014. Cited on pages 64 and 69.

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *J. Cryptology*, 14(2) :101–119, 2001. Cited on page 40.

[BDTW01] Dan Boneh, Xuhua Ding, Gene Tsudik, and Chi-Ming Wong. A Method for Fast Revocation of Public Key Certificates and Security Capabilities. In *10th USENIX Security Symposium, August 13-17, 2001, Washington, D.C., USA*, 2001. Cited on page 118.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 213–229, 2001. Cited on pages 19 and 99.

[BFG15] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner Product Masking Revisited. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 486–510, 2015. Cited on page 59.

[BFGV12] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and Practice of a Leakage Resilient Masking Scheme. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 758–775, 2012. Cited on page 59.

[BGH+16] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, Olivier Rioul, François-Xavier Standaert, and Yannick Teglia. Taylor Expansion of Maximum Likelihood Attacks for Masked and Shuffled Implementations. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 573–601, 2016. Cited on page 88.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10, 1988. Cited on pages 53 and 57.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In *Advances in Cryptology - ASIACRYPT 2001, 7th*

International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings, pages 514–532, 2001. Cited on page 98.

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical : A Paradigm for Designing Efficient Protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993. Cited on page 102.

[BS03] Johannes Blömer and Jean-Pierre Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*, pages 162–181, 2003. Cited on page 40.

[Car13] Claude Carlet. *Correlation-Immune Boolean Functions for Leakage Squeezing and Rotating S-Box Masking against Side Channel Attacks.* 2013. Cited on page 62.

[CDD+15] Claude Carlet, Abderrahman Daif, Jean-Luc Danger, Sylvain Guilley, Zakaria Najm, Xuan Thuy Ngo, Thibault Porteboeuf, and Cédric Tavernier. Optimized linear complementary codes implementation for hardware trojan prevention. In *European Conference on Circuit Theory and Design, ECCTD 2015, Trondheim, Norway, August 24-26, 2015*, pages 1–4, 2015. Cited on pages 71 and 133.

[CDG+14] Claude Carlet, Jean-Luc Danger, Sylvain Guilley, Houssem Maghrebi, and Emmanuel Prouff. Achieving side-channel high-order correlation immunity with leakage squeezing. *J. Cryptographic Engineering*, 4(2) :107–121, 2014. Cited on page 62.

[CDGM12] Claude Carlet, Jean-Luc Danger, Sylvain Guilley, and Houssem Maghrebi. Leakage Squeezing of Order Two. *IACR Cryptology ePrint Archive*, 2012 :567, 2012. Cited on page 62.

[CDGT18] Claude Carlet, Abderrahman Daif, Sylvain Guilley, and Cédric Tavernier. Polynomial direct sum masking to protect against both SCA and FIA. *Journal of Cryptographic Engineering*, Aug 2018. Cited on pages 76, 77, and 133.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 398–412, 1999. Cited on page 40.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6) :644–654, 1976. Cited on pages 4 and 13.

[DQ86] Yvo Desmedt and Jean-Jacques Quisquater. Public-Key Systems Based on the Difficulty of Tampering (Is There a Difference Between DES and RSA ?). In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 111–117, 1986. Cited on pages 6 and 19.

[DR00] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *AES Candidate Conference*, pages 343–348, 2000. Cited on pages 4 and 32.

[DT16] Abderrahman Daif and Cédric Tavernier. An efficient certificateless key management architecture to solve IBE and PKI issues. In *Fifteenth International Workshop on Algebraic and Combinatorial Coding Theory*, pages 111–116, 2016. Cited on pages 112 and 133.

[FK92] David Ferraiolo and Richard Kuhn. Role-Based Access Controls. *15th National Computer Security Conference (NCSC)*, 1992.

[GCMZ12] Aijun Ge, Cheng Chen, Chuangui Ma, and Zhenfeng Zhang. Short and Efficient Expressive Attribute-Based Signature in the Standard Model. *IACR Cryptology ePrint Archive*, 2012 :125, 2012.

[Gen03] Craig Gentry. Certificate-Based Encryption and the Certificate Revocation Problem. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 272–293, 2003. Cited on page 118.

[GM11] Louis Goubin and Ange Martinelli. Protecting AES with Shamir's Secret Sharing Scheme. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 79–94, 2011. Cited on pages 53 and 55.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis : Concrete Results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, number Generators, pages 251–261, 2001. Cited on page 40.

[Goy07] Vipul Goyal. Certificate Revocation Using Fine Grained Certificate Space Partitioning. In *Financial Cryptography and Data Security, 11th International Conference, FC 2007, and 1st International Workshop on Usable Security, USEC 2007, Scarborough, Trinidad and Tobago,*

February 12-16, 2007. Revised Selected Papers, pages 247–259, 2007. Cited on page 118.

[GP99] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis (The "Duplication" Method). In *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, pages 158–172, 1999. Cited on page 40.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98, 2006.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111, 1998. Cited on page 57.

[Ham50] Richard Wesley Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2) :147–160, April 1950. Cited on page 23.

[HHSI05] Yumiko Hanaoka, Goichiro Hanaoka, Junji Shikata, and Hideki Imai. Identity-Based Hierarchical Strongly Key-Insulated Encryption and Its Application. In *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, pages 495–514, 2005. Cited on page 118.

[HJW00] Detlef Hühnlein, Michael J. Jacobson Jr., and Damian Weber. Towards Practical Non-interactive Public Key Cryptosystems Using Non-maximal Imaginary Quadratic Orders. In *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings*, pages 275–287, 2000. Cited on pages 6 and 19.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits : Securing Hardware against Probing Attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 463–481, 2003. Cited on page 42.

[JLQ99] Marc Joye, Arjen K. Lenstra, and Jean-Jacques Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *J. Cryptology*, 12(4) :241–245, 1999. Cited on page 40.

[Jou00] Antoine Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Algorithmic Number Theory, 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings*, pages 385–394, 2000. Cited on page 98.

[Kah74] David Kahn. *The codebreakers*. Weidenfeld and Nicolson, 1974. Cited on page 40.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999. Cited on page 40.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177) :203–209, 1987. Cited on page 94.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996. Cited on page 40.

[LBD+04] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Secure Key Issuing in ID-based Cryptography. In *ACSW Frontiers 2004, 2004 ACSW Workshops - the Australasian Information Security Workshop (AISW2004), the Australasian Workshop on Data Mining and Web Intelligence (DMWI2004), and the Australasian Workshop on Software Internationalisation (AWSI2004) . Dunedin, New Zealand, January 2004*, pages 69–74, 2004. Cited on pages 106 and 125.

[Liu68] Chung Laung Liu. *Introduction to combinatorial mathematics*, volume 181. McGraw-Hill New York, 1968. Cited on page 53.

[LQ03] Benoît Libert and Jean-Jacques Quisquater. Efficient revocation and threshold pairing based cryptosystems. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003*, pages 163–171, 2003. Cited on page 118.

[Mas92] James L. Massey. Linear codes with complementary duals. *Discrete Mathematics*, 106-107 :337–342, 1992. Cited on page 63.

[Mes00] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, pages 238–251, 2000. Cited on pages 42 and 55.

[Mic08] Silvio Micali. Efficient certificate revocation, february 2008. US Patent 7,337,315. Cited on page 118.

[Mil85] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, pages 417–426, 1985. Cited on page 94.

[Mul54] David E. Muller. Application of Boolean algebra to switching circuit design and to error detection. *Trans. I.R.E. Prof. Group on Electronic Computers*, 3(3) :6–12, 1954. Cited on page 28.

[MY91] Ueli M. Maurer and Yacov Yacobi. Non-interactive Public-Key Cryptography. In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, pages 498–507, 1991. Cited on pages 6 and 19.

[NSGD12] Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. RSM : A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1173–1178, 2012. Cited on page 62.

[PR07] Emmanuel Prouff and Matthieu Rivain. A Generic Method for Secure SBox Implementation. In *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, pages 227–244, 2007. Cited on page 62.

[PR11] Emmanuel Prouff and Thomas Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 63–78, 2011. Cited on pages 53 and 55.

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA) : Measures and Counter-Measures for Smart Cards. In *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, pages 200–210, 2001. Cited on page 40.

[QS02] Jean-Jacques Quisquater and David Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of Esmart*, volume 2002, 2002. Cited on page 40.

[Ree54] Irving Stoy Reed. A class of multiple-error-correcting codes and the decoding scheme. *Transactions of the IRE Professional Group on Information Theory*, 4(4) :38–49, September 1954. Cited on page 28.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 413–427, 2010. Cited on pages 44 and 85.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 171–188, 2009. Cited on pages 41 and 42.

[RR01] Josyula R. Rao and Pankaj Rohatgi. EMpowering Side-Channel Attacks. *IACR Cryptology ePrint Archive*, 2001 :37, 2001. Cited on page 40.

[RS60] I. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2) :300–304, 1960. Cited on page 55.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978. Cited on page 4.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 2–12, 2002. Cited on page 40.

[Sha49] Claude Elwood Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4) :656–715, 1949. Cited on page 32.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11) :612–613, 1979. Cited on page 53.

[Sha84] Adi Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 47–53, 1984. Cited on pages 6 and 18.

[SP06] Kai Schramm and Christof Paar. Higher Order Masking of the AES. In *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, pages 208–225, 2006. Cited on page 42.

[SW63] Claude Elwood Shannon and Warren Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, USA, 1963. Cited on page 23.

[SW05] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 457–473, 2005.

[Tan87] Hatsukazu Tanaka. A Realization Scheme for the Identity-Based Cryptosystem. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 340–349, 1987. Cited on pages 6 and 19.

[TI89] Shigeo Tsujii and Toshiya Itoh. An ID-based cryptosystem based on the discrete logarithm problem. *IEEE Journal on Selected Areas in Communications*, 7(4) :467–473, 1989. Cited on pages 6 and 19.

[XZQ05] Chunxiang Xu, Junhui Zhou, and Zhiguang Qin. A Note on Secure Key Issuing in ID-based Cryptography. *IACR Cryptology ePrint Archive*, 2005 :180, 2005. Cited on pages 106, 109, 117, and 125.