Contribution à la cryptographie interactive

Baptiste Cottier

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Ecole Normale Supérieure

Contribution to interactive cryptography

Soutenue par
Baptiste COTTIER
Le 06 janvier 2023

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Cristina Onete Université de Limoges	*Rapporteur*
Sébastien Canard Orange Labs	*Rapporteur*
Damien Vergnaud Sorbonne Université, CNRS, LIP6	PRÉSIDENT DU JURY *Examinateur*
Maryline Laurent Télécom SudParis	*Examinateur*
David Pointcheval CNRS, Ecole Normale Supérieure	*Directeur de thèse*
Olivier Blazy Ecole Polytechnique	*Codirecteur de thèse*
Slim Bettaieb Worldline	*Invité*
Loïc Bidoux Worldline Technology Innovation Institute	*Invité*

1794
ENS | PSL

Remerciements

Dans un premier temps, je tiens naturellement à remercier David Pointcheval, mon directeur de thèse, pour sa confiance, son soutien ainsi que sa patience, que j'ai probablement parfois dû mettre à rude épreuve. Merci également à Olivier Blazy, co-directeur de cette thèse, pour sa disponibilité et sa pédagogie.

Mes remerciements vont aussi à Slim Bettaieb et Loïc Bidoux, mes superviseurs, pour m'avoir initié à l'univers de la recherche. J'espère que votre départ à l'autre bout du monde pendant ma thèse n'est qu'une coïncidence. Merci à Jean-Claude Barbezange sans qui cette thèse n'aurait pas eu lieu. Merci à Benjamin, Céline, Olivier et de manière plus générale à mes collègues de chez Worldline de m'avoir accepté avec mes pieds nus et leurs chaussons, mon humour et mes bavardages répétitifs.

Merci aux autres doctorants du laboratoire pour nos échanges, que ce soit autour d'un café ou d'une table de ping-pong. Merci également à Lise-Marie, Linda et Tiffany pour leur bonne humeur permanente et nos longues discussions.

Finalement, de manière plus pudique, merci à ma famille et mes amis, qui me supportent (dans tous les sens du terme) depuis de nombreuses années et n'ont cessé de croire en moi. Un merci tout particulier à celle dont notre aventure a commencé en même temps que cette thèse, mais qui elle ne se terminera pas.

N'étant pas de nature à exposer mes sentiments, je sais que cette section peut paraître succinte à prime abord. Cependant, je suis conscient que sans l'entourage que j'ai pu avoir pendant cette thèse, de tout point de vue, je n'en serai pas venu à bout, et je suis reconnaissant envers chacun d'entre vous.

Contents

List of Figures

List of Tables

Notations & acronyms

We define here some global notations. A more precise list of notations is given at the beginning of each chapter.

Symbols	
$a \| b$	a concatenated with b
$a \leftarrow\!\!\$\ A$	a is uniformly sampled from A
$[A]$	$[1\ ..\ A]$

Encryption	
sk	Secret key
pk	Public Key
$\mathcal{M}$	Space of Messages
$\mathcal{K}$	Space of Keys
IV	Initialization Vector
AHE	Additive Homomorphic Encryption
AEAD	Authenticated Encryption with Additional Data
OTP	One-Time Pad

Security	
1^κ	Security Parameter
$\mathcal{A}$	Adversary
$\mathcal{A}^F$	Adversary with access to an oracle computing $F(\cdot)$
$\mathfrak{Q}$	List of queries made by $\mathcal{A}$ to an oracle
Exp	Experiment
Adv	Advantage of the adversary
$\mathcal{F}$	Functionality
CDH	Computational Diffie-Hellman
DDH	Decisional Diffie-Hellman
GDH	Gap Diffie-Hellman
IND-CPA	INDistinguishability under Chosen Plaintext Attack
IND-CCA	INDistinguishability under Chosen Ciphertext Attack
EUF-CMA	Existential UnForgeability under Chosen Messages Attack
RO	Random Oracle
SO	Simulation Oracle

Primitives	
MPC	Multiparty Secure Computation
SPHF	Smooth Projective Hash Function(s)
SPHFwGZ	Smooth Projective Hash Function(s) with Gray Zone
OT	Oblivious Transfer
NIZK	Non-Interactive Zero Knowledge Proofs
MAC	Message Authentication Code

Chapter 1a

Introduction – Version française

Dans cette section, nous présentons les motivations qui ont conduit aux divers projets de cette thèse. Dans un second temps, nous présenterons les divers objectifs, ainsi que les contributions en ayant découléés dans une dernière section.

1.1 Motivations

Au quotidien, nos données personnelles sont de plus en plus exploitées. Avec l'arrivée du Règlement général sur la protection des données (RGPD) en 2016, la confidentialité a gagné en intérêt auprès des utilisateurs tout autant qu'auprès des entreprises. Afin de protéger ses données, une solution consiste à faire appel à la cryptographie. Depuis les petits messages codés échangés à l'école jusqu'aux communications inter-présidentielles, la cryptographie trouve facilement sa place dans la communication entre humains. Il va de soi qu'aujourd'hui, le spectre de la cryptographie ne se limite plus aux interactions humaines, mais atteint aussi nos téléphones, nos ordinateurs et de façon générale, le moindre objet connecté qui nous entoure, entre autres les serveurs. Bien qu'au premier abord, la confidentialité semble ne présenter que des avantages, elle peut constituer un frein dans certains cas. Prenons l'exemple d'une messagerie. Il est important quand nous utilisons une messagerie que seuls les deux contacts aient accès aux messages et médias échangés. D'un autre point de vue, garantir une telle confidentialité peut attirer des utilisateurs aux intentions malhonnêtes. Il serait donc tout aussi appréciable que, si des messages à contenu illégal sont échangés, cette même messagerie puisse les détecter et agir en conséquence. Se pose alors la question suivante : Comment trouver le parfait, ou à défaut le meilleur, équilibre entre la confidentialité et l'exploitation des données ? La branche de la cryptographie qui étudie ce genre de problématiques est appelée *Calcul Multipartite Sécurisé*. Le Calcul Multipartite Sécurisé peut se définir comme une famille de protocoles permettant à plusieurs acteurs, appelés parties, de collaborer afin d'évaluer une fonction de leurs données. Ces valeurs données en entrée du protocole, et de manière générale toute valeur donnée en entrée d'un protocole, seront désignées *input*, tandis que les valeurs retournées par le protocole seront appelées *output*. À la fin de l'exécution d'un protocole, chaque partie ne doit rien connaître de plus que ce qu'elles ne peuvent déduire à partir de leurs *input* et *output* respectifs. Pour reprendre l'exemple de la messagerie, cela reviendrait à pouvoir détecter un message à contenu illégal sans pour autant connaître le contenu du message.

Il n'est pas rare que lorsque toutes ces données sont collectées, elles soient utilisées comme *input* d'un modèle d'apprentissage automatique (*Machine Learning* en anglais). L'apprentissage automatique est un outil qui permet de classer, prédire ou estimer une valeur, à partir d'échantillons préalablement utilisés afin d'établir un modèle du comportement de ces échantillons. C'est un outil massivement utilisé de nos jours, que ce soit pour la détection de pourriel, la prédiction de la météo, ou bien, et c'est ce dernier cas d'usage qui va nous intéresser, pour pouvoir authentifier de manière continue un utilisateur. Authentifier de manière continue un utilisateur consiste à

l'authentifier sans que cela n'impacte son utilisation, de manière transparente. Dans le cadre de l'authentification continue sur mobile, cela peut se faire en analysant la vitesse à laquelle il utilise son clavier, la pression et la trajectoire de son doigt quand il utilise son navigateur, etc. C'est son comportement qui va lui permettre de s'authentifier. Du fait que ces données servent à l'authentification d'un utilisateur, ce sont des données sensibles, qui demandent un minimum de confidentialité. Il en va de même pour le modèle enregistré sur un serveur, qui, s'il était dévoilé, pourrait permettre à un utilisateur de fausser ses données pour pouvoir usurper l'identité d'un autre utilisateur. C'est pourquoi il faut donc aussi que le modèle utilisé soit confidentiel. Cela nous amène à une problématique proche de celle décrite précédemment :

Comment évaluer un modèle d'apprentissage automatique pour de l'authentification continue de manière confidentielle ?

Nous répondons à cette question dans le troisième chapitre de ce manuscrit. Notre réponse contient deux protocoles ; l'un est prouvé sûr contre un adversaire honnête-mais-curieux et le second, contre un adversaire malveillant. L'adversaire honnête-mais-curieux représente une partie qui suit le protocole, mais qui va essayer d'extraire autant d'informations que possible sur les autres parties à partir de ce qu'elle voit. L'adversaire malveillant, lui, est libre de ne pas suivre le protocole. Il peut le modifier pour en apprendre davantage sur les autres parties, mais il peut aussi essayer d'orienter le protocole afin d'obtenir le résultat qu'il désire. Dans ce second protocole, nous faisons appel à plusieurs primitives de Calcul Multipartite Sécurisé, dont les Transferts Inconscients (*Oblivious Transfer*s en anglais). Ces derniers permettent à un récepteur de récupérer un message précis parmi plusieurs, détenus par l'expéditeur, sans que ce dernier ne sache quel message a été récupéré par le récepteur et sans que celui-ci n'apprenne d'information sur les messages qu'il n'était pas censé récupérer. Il existe de nombreuses instanciations de Transferts Inconscients. Celles qui vont nous intéresser ici sont celles basées sur les *Smooth Projective Hash Functions* (SPHF). Les SPHF reposent sur un ensemble, appelé langage, et une paire de clés chacune associée à une fonction de hachage. Ces clés sont générées de telle sorte que si le mot donné en *input* dudit SPHF est un mot du langage associé à la SPHF, les deux fonctions de hachage retournent la même valeur. Cette propriété est appelée *correctness*. De plus, quand le mot donné en *input* n'est pas un mot du langage, les valeurs obtenues par les deux fonctions de hachage doivent être différentes et indiscernables. Cette propriété est appelée *smoothness*. Dans le cas où les Transferts Inconscients sont basés sur des SPHF faisant appel à de la cryptographie classique, les choses se passent bien. Cependant, quand ils sont basés sur des SPHF faisant appel à de la cryptographie post-quantique, une zone grise fait son apparition. Quand un mot est dans cette zone grise, npus ne pouvons plus prétendre à la propriété de *correctness* ni de *smoothness*. Cette incertitude laisse donc place à de potentielles attaques. Bien que des travaux aient déjà étudié la question, leur solution, faisant appel à des preuves à divulgation nulle de connaissance, alourdissent le protcole. Nous nous sommes donc posé la question suivante :

Comment exploiter sans surcoût ces Smooth Projective Hash Functions *avec une zone grise ?*

De plus, les Transferts Inconscients sont essentiellement composés avec d'autres protocoles. De ce fait, il peut être important de s'assurer que cette composition avec d'autres protocoles n'amène pas de nouvelles attaques. Un protocole qui fournit une telle sécurité est dit universellement composable. Ainsi, nous nous attendons à ce que notre futur construction soit démontrée sécurisée dans le modèle de la composabilité universelle.

Comme dit dans les premières lignes de cette section, la cryptographie n'implique plus seulement des humains, mais aussi des machines telles que des serveurs, des smartphones, etc. Nous sommes de plus en plus entourés d'objets du quotidien connectés à Internet. Caméras, ampoules, appareils électroménagers tels que frigidaires et fours, et même des capuchons de réservoir d'essence pour camion sont connectés à Internet. Les raisons d'une telle connectivité sont diverses : facilitation

de la vie, divertissement ou encore sécurité. De nombreux capteurs sont aussi reliés à Internet pour avoir accès de manière continue aux données mesurées.

Lorsque cette connectivité a lieu au sein d'un domicile nous parlons de domotique, mais nous parlerons ici plutôt de sa généralisation, dénommant l'inter-connectivité des objets reliés à Internet : l'Internet des Objets (ou *Internet of Things* en anglais). En 2021, alors que la population mondiale est estimée à un peu moins de 8 milliards d'habitants, le nombre d'appareils faisant partie de l'Internet des Objets est estimé à plus de 12 milliards ; chiffre n'incluant pas les appareils permettant à un utilisateur d'accéder à Internet tels que smartphones, ordinateurs, télévision. L'Internet des Objets se limite donc aux objets qui fonctionnent grâce à Internet. Ces objets sont souvent des appareils restreints, que ce soit en termes de consommation énergétique, de puissance de calcul ou encore de mémoire disponible. En effet, nous ne pouvons pas attendre d'un bouchon de réservoir d'avoir les même performances qu'un ordinateur de bureau. Ainsi, toutes ces restrictions impliquent de devoir adapter les primitives cryptographiques utilisées.

Le domaine de la cryptographie qui étudie et adapte la cryptographie aux appareils restreints est appelée Cryptographie à bas-coût (*Lightweight Cryptography* en anglais). Plus précisément, l'objectif de la cryptographie à bas-coût est de trouver le meilleur compromis entre faibles performances, faibles coûts et sécurité. Cela concerne aussi bien évidemment les protocoles d'échange de clés, permettant à deux appareils de partager une même clé privée. Certains de ces protocoles permettent aussi d'authentifier formellement l'appareil avec lequel ils communiquent. Parmi eux, EDHOC, pour *Ephemeral Diffie-Hellman Over COSE*, est en cours d'études et de standardisation. Le protocole EDHOC est un protocole d'échange de clés authentifié développé par le groupe de travail *Lightweight Authenticated Key Exchange* de l'Internet Engineering Task Force (IETF). Une étape cruciale avant un déploiement à grande échelle est l'analyse de sécurité d'un tel protocole. L'analyse formelle ayant déjà été réalisée, il nous a été suggéré d'en réaliser l'analyse dans le modèle calculatoire et de répondre à la question suivante :

Le protocole d'échange de clés EDHOC fournit-il une sécurité suffisante ?

1.2 Objectifs

Apprentissage automatique préservant la confidentialité. Avant d'aborder la question des objectifs, nous décrivons ici nos contraintes. La première de celles-ci est la localisation de l'évaluation du modèle sur les données. Il est assez fréquent d'envoyer ses données à un serveur pour qu'il puisse évaluer son modèle sur ces données. Cependant, dans le cas de l'authentification continue, pour des raisons de coût de communication, l'évaluation doit avoir lieu sur le mobile de l'utilisateur. Pour les mêmes raisons, il faut réduire au maximum le nombre de communications entre le serveur et le mobile. Étant donné que l'évaluation se fait sur le mobile de l'utilisateur, le serveur doit pouvoir lui envoyer le modèle qu'il stocke, en conservant un certain niveau de confidentialité. Se pose donc la question de savoir quelle technique d'apprentissage utiliser.

Il existe de multiples techniques d'apprentissage automatique. Cependant, la plupart d'entre elles utilisent des fonctions mathématiques que nous ne savons pas évaluer efficacement de manière cryptographique. La première étape est donc de déterminer à quelle technique d'apprentissage automatisé nous souhaitions ajouter une couche de cryptographie.

Ensuite, l'objectif sera de concevoir un protocole qui permet à un utilisateur d'évaluer ses données sur un modèle encodé (au sens où pas – ou peu – d'information en découle). Nous nous limiterons aux modèles à résultat binaire, c'est-à-dire ceux dont le résultat peut se résumer à accept ou reject. Dans un premier temps, nous nous concentrerons sur le modèle Honnête-mais-Curieux, puis nous essaierons d'étendre le protocole obtenu aux adversaires malveillants. Bien évidemment, les coûts, que ce soit de calcul et de communication, doivent être raisonnables et adaptés aux appareils concernés afin que l'expérience de l'utilisateur ne se retrouve pas impactée.

Après l'exécution desdits protocoles, le serveur ne doit rien avoir appris de plus sur les données de l'utilisateur que le résultat, tandis que le client ne doit rien avoir appris de confidentiel sur le modèle.

Un protocole de Transfert Inconscient Post-Quantique Universellement Composable basé sur les SPHFs. Comme il a été dit dans la section précédente, utiliser des SPHF basés sur la cryptographie post-quantique implique l'apparition d'une zone grise, dont nous ne pouvons pas prédire la correctness ou la smoothness. Récemment, Bettaieb *et al.* [BBB+22a] ont construit une Smooth Projective Hash Function post-quantique basée sur les codes correcteurs d'erreurs, en annihilant les problématiques liées à la zone grise en utilisant des preuves à divulgation nulle de connaissance. Bien que leur travail apporte une contre-mesure à cette problématique, l'utilisation de preuves à divulgation nulle de connaissances impacte les performances des SPHF. Notre objectif était donc de résoudre cette problématique sous un autre angle, moins impactant. Être capable de contrecarrer cette zone grise éliminera les risques qu'elle soit exploitée pour de potentielles attaques. Néanmoins, cela n'engage en rien la composabilité de la SPHF qui en résulte au sein d'un protocole de Transfert Inconscient. Ainsi, il nous faudra aussi fournir une preuve de sécurité du caractère universellement composable de notre Transfert Inconscient basé sur ces SPHF.

Analyse de Sécurité Calculatoire du protocole EDHOC. L'objectif de cette analyse est de venir compléter les travaux déjà réalisés sur les analyses formelles du protcole EDHOC. Ce dernier est un protocole authentifié d'échange de clés dont l'authentification peut se faire par deux méthodes : en utilisant une signature ou bien en utilisant une clé Diffie-Hellman statique. C'est sur cette dernière méthode qu'il nous a été suggéré d'étudier la sécurité du protocole, notamment avec des paramètres agressifs. Pour chaque exécution du protocole, une clé de session est générée en dérivant ces clés Diffie-Hellman statiques ainsi que des clés éphémères, à usage unique. L'analyse de sécurité du protocole se divise en trois points :

- Confidentialité de la clé ou Authentification Implicite : Au plus les deux participants connaissent la clé de session finale. Cette propriété doit rester valable même si les clés statiques sont révélées (*Forward Secrecy*).
- Authentification Mutuelle ou Authentification Explicite : Exactement les deux participants peuvent calculer la clé de session finale
- Protection de l'Identité : Au plus les deux participants connaissent l'identité des deux parties impliquées.

1.3 Contributions

Apprentissage automatique préservant la confidentialité. Nous avons réalisé deux protocoles pour évaluer de manière sécurisée une forêt de décision avec des classes de sorties binaires. Dans notre contexte, un serveur cherche à connaître le résultat de l'évaluation d'un modèle $\mathcal{M}$ par rapport à des données $\boldsymbol{x}$ d'un client, puis accept ou reject ce dernier selon ce que retourne l'évaluation. Pour nos deux protocoles, nous considèrons un serveur honnête-mais-curieux. Dans le premier protocole le client est, lui aussi, considéré honnête-mais-curieux puis dans le second protocole, elle est considéré malveillanre. Sachant que nous visons des applications où les interactions entre le client et le serveur doivent être minimales, nos protocoles se divisent en deux phases. Une phase préliminaire, hors-ligne, permet d'encoder le modèle du serveur, et de l'envoyer au client, indépendemment des données du client.

Dans le protocole conçu où le client est honnête-mais-curieux, l'évaluation ne nécessite qu'un flux du client vers le serveur dans la phase en-ligne pour que le protocole soit pleinement réalisé. Ce résultat surpasse les performances des autres protocoles de la littérature.

Dans le cas où le client est malveillant, notre protocole peut s'interpréter comme un compromis entre des constructions déjà existantes en ce qui concerne le nombre de flux et le coût en termes de bande passante.

Cependant, nos protocoles révèlent au serveur le nombre d'arbres (mais pas lesquels) ayant retourné accept. Cela peut se voir comme une fonctionnalité plus qu'un inconvénient sachant que nous désirons que le serveur apprenne le résultat de l'évaluation dans notre contexte. En effet, bien que la valeur retournée par les arbres soit binaire, le serveur peut adapter son comportement en fonction de la proportion d'arbres qui ont été évalués avec succès. Cela rejoint la notion d'indice de confiance que nous pouvons régulièrement retrouver dans certains cas d'usage, tel que l'authentification continue.

Dans notre contexte de sécurité malveillante, nous utilisons des *Garbled Circuits* qui permettent d'évaluer de manière sécurisée des circuits booléens. Bien qu'évalués dans un cadre malveillant, nous ne nécessitons aucune technique supplémentaire par rapport au cadre honnête-mais-curieux. Ainsi, nous n'avons pas besoin d'utiliser des solutions telles que *Cut & Choose*. Même si cette solution a été bien étudiée et optimisée [MF06, LP07, Lin13, LR14, AMPR14, WMK17], il s'agit toujours d'une solution coûteuse nécessitant ℓ *Garbled Circuits* pour atteindre une sécurité statistique de $2^{-\ell}$ bits. Dans une étude, Dupin et al. [DPB18] ont montré qu'un générateur malveillant peut corrompre un *Garbled Circuits* uniquement en ajoutant des portes logiques NOT ou en modifiant les labels utilisés de telle sorte que le circuit échoue dans certaines conditions (attaque par échec). Cependant, nous ne sommes pas sujets à ces attaques. En effet, en cas d'attaques par échec, l'adversaire sera probablement rejeté, n'apprenant aucune information sur les seuils utilisés dans les arbres. Dans le cas d'un circuit erroné (avec des portes NOT supplémentaires), nous avons introduit la notion de polarité, qui consiste en des inversions aléatoires des résultats des chemins, et donc une attaque se réduirait à deviner toutes (ou la plupart) des inversions, conduisant probablement à un rejet, sans fuite d'information.

Ces contributions ont abouti à une publication :

★ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Secure decision forest evaluation. In *The 16th International Conference on Availability, Reliability and Security*, pages 1–12, 2021

Une version de travail mise à jour est disponible sur les archives HAL:

► Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Secure Decision Forest Evaluation. hal-03321368, 2022. https://hal.archives-ouvertes.fr/hal-03321368

Un protocole de Transfert Inconscient Post-Quantique Universellement Composable basé sur les SPHFs. Plutôt qu'annihiler la zone grise propre à l'utilisation de la cryptographie post-quantique, comme l'ont fait Bettaieb *et al.* dans [BBB+22a], nous avons préféré étudier les conditions requises dans l'objectif de dompter cette zone et de mieux contrôler son comportement. De cette étude a résulté la notion d'*Insolubilité de Décomposition*. Par conséquent, nous avons introduit la notion de *Smooth Projective Hash Functions with Gray Zone* (SPHFwGZ) qui est une SPHF avec la propriété supplémentaire d'Insolubilité de Décomposition. Cette propriété affirme que, étant donné une valeur témoin aléatoire, il est hautement compliqué de générer deux valeurs soit dans le langage du SPHF, soit dans la zone grise, qui soient le complémentaire l'une de l'autre, vis-à-vis de la valeur témoin.

Nous avons par la suite montré que toute *Smooth Projective Hash Functions with Gray Zone* dont le langage se base sur des chiffrés d'un schéma de chiffrement homomorphe peut servir de base à la construction d'un *Oblivious Transfer*. Cette preuve a été faite dans le modèle de Composition Universelle et repose sur la sécurité sémantique du schéma de chiffrement.

Ces contributions sont accompagnées d'instanciations de telles SPHFwGZ. La première d'entre elles s'appuie sur le Problème de Diffie-Hellman et le cryptosystème d'ElGamal. Étant donné que le déchiffrement n'échoue jamais avec le cryptosystème d'ElGamal, la zone grise est vide et l'Insolubilité de Décomposition est évidente. Nous pouvons noter qu'en réalité, une telle SPHFwGZ est *de facto* une SPHF. L'idée derrière cette instanciation est d'une part, de familiariser le lecteur avec notre construction, et d'une autre part, de mettre en avant le fait que cette construction fonctionne avec toute SPHF. Une seconde instanciation se base sur les réseaux euclidiens, et plus précisément sur le problème *Learning with Errors*. Cette instanciation permet de souligner la généricité de notre construction.

Ces contributions ont conduit à la publication suivante :

★ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Post-Quantum and UC-secure Oblivious Transfer from SPHF with Grey Zone. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022

Une version de travail mise à jour est disponible sur les archives HAL:

▶ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Post-Quantum Oblivious Transfer from Smooth Projective Hash Functions with grey[1]Zone. hal-03772089, 2022. https://hal.archives-ouvertes.fr/hal-03772089

Analyse de sécurité calculatoire du protocole EDHOC. Nous avons réalisé la preuve de sécurité du protocole d'échange de clé authentifié EDHOC instancié avec la méthode d'authentification par clé Diffie-Hellman statique. De plus, la preuve a été réalisée avec une configuration agressive, dans le sens où la longueur de la valeur retournée par le MAC est de 64 bits.

Il s'en est conclu que la confidentialité de la clé approche les 128 bits de sécurité, ainsi que la Protection de l'Identité des deux parties. Néanmoins, l'Authentification Mutuelle atteint une sécurité de 64 bits pour l'Initiateur et le Répondant. Bien que cela soit assumé pour le Répondant – la documentation suggère un message supplémentaire si une plus forte sécurité est désirée –, dans le cas de l'Initiateur, c'est une réelle attaque que nous avons mis en exergue. Cependant, nous avons aussi prouvé qu'en apportant des modifications au protocole, sans coût calculatoire et de communication supplémentaires comparés aux performances du protocole initial, le protocole EDHOC peut atteindre une sécurité de 128 bits pour l'Authentification Mutuelle de l'Initiateur.

Ces contributions ont conduit à la publication suivante, ayant obtenu le "Best Paper Award" :

★ Baptiste Cottier and David Pointcheval. Security Analysis of Improved EDHOC Protocol. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022

Une version de travail mise à jour est disponible sur les archives HAL:

▶ Baptiste Cottier and David Pointcheval. Security analysis of the EDHOC protocol. hal-03772082v2, 2022. https://hal.archives-ouvertes.fr/hal-03772082

1.4 Organisation

Ce chapitre donne le contexte de la thèse au lecteur, et sera le seul du manuscrit rédigé en français. Cette introduction est suivie d'un résumé long de la thèse en français. Le chapitre 2 présente toutes les notions nécessaires à une compréhension complète des chapitres 3, 4 and 5. Le chapitre 3 décrit les protocoles permettant d'évaluer de manière confidentielle un modèle sur le mobile d'un client, ainsi que leurs coûts théoriques et performances pratiques sur deux cas d'usage. Dans le chapitre 4, nous détaillons la construction des Smooth Projective Hash Functions avec Zone Grise ainsi

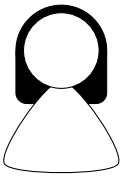
que la construction du protocole de Transfert Inconscient qui en découle. Nous y prouvons que ce dernier protocole est bien universellement composable, puis nous terminons ce chapitre par deux constructions, l'une basée sur de la cryptographie classique, l'autre sur de la cryptographie post-quantique. Le chapitre 5 décrit le protocole EDHOC tel qu'il était défini au moment de notre analyse. Cette dernière suit immédiatement cette description, puis nous concluons en décrivant des améliorations. Ce manuscrit se termine par une conclusion, dans laquelle nous émettons de potentiels futurs projets de recherche ainsi que des questions ouvertes.

1.5 Résumé long

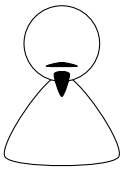
Le but de cette section est de présenter de manière simple et concise le contenu de cette thèse. Sa structure suivra celle de la thèse. Nous débuterons par une introduction pédagogique des diverses primitives et notions utilisées, puis développerons les contributions et résultats obtenus.

1.5.1 Préliminaires

Dans un premier temps, il est de coutume d'employer des prénoms pour expliquer le fonctionnement des diverses primitives, afin d'en faciliter la compréhension. L'un des acteurs est genré au féminin et le second est genré au masculin, exploitant les pronoms de ces derniers pour faciliter la compréhension. Ainsi, laissez-nous vous présenter Alice et Bob :



Prénoms: Alice, Renée, Christelle
Rôles: Chiffreuse, Réceptrice, Cliente
Pronoms: Elle, la



Prénoms: Bob, Edouard, Selim
Rôles: Déchiffreur, Expéditeur, Serveur
Pronoms: Il, lui

Leurs rôles seront nécessaires dans la suite de ce résumé.

Chiffrement. En cryptographie, nous distinguons deux familles de schéma de chiffrement. Dans la première famille, la clé utilisée par Alice pour chiffrer est la même que celle utilisée par Bob pour déchiffrer. Ce sont les schémas les plus simples à mettre en place, car le déchiffrement consiste essentiellement à "rembobiner" le chiffrement. Cette famille de schémas est appelée *Cryptographie Symétrique* ou *Cryptographie à clé secrète*. Dans cette section, nous serons amenés à parler d'un de ces schémas de chiffrement, le One-Time Pad.

> **One-Time Pad.** Étant donné un message m de longueur k bits, le One-Time Pad génère une clé secrète sk de longueur, elle aussi, k bits. Le chiffré c de m avec la clé sk consiste en un xor ($\oplus$) entre le message et ma clé. Formellement, nous avons $c \leftarrow \mathsf{Enc}(\mathsf{sk}, m) = m \oplus \mathsf{sk}$. Pour déchiffrer, Bob calcule $c \oplus \mathsf{sk} = m \oplus \mathsf{sk} \oplus \mathsf{sk} = m$.

Cependant, pour qu'un schéma de chiffrement à clé symétrique puisse être mis en place, il faut aussi transmettre de manière sécurisée ladite clé secrète à la personne avec laquelle nous souhaitons communiquer, ce qui peut parfois s'avérer compliqué. C'est pour cette raison qu'à la fin des années 70, des schémas où la notion d'une clé différente pour le chiffrement et le déchiffrement a fait son apparition. Dans ce contexte, la clé servant à chiffrer pouvait être publique, tandis que celle servant à déchiffrer devait logiquement rester secrète. Ainsi, si Alice souhaite communiquer avec Bob, elle récupère sa clé publique et lui envoie son message chiffré, que lui seul pourra déchiffrer, car il est le seul à connaître la clé de déchiffrement. De plus, étant donné que la clé de chiffrement est publique, tout le monde peut utiliser cette même clé pour communiquer avec Bob, là où dans le contexte de la cryptographie à clé secrète, une clé différente devait être associée à chaque personne voulant communiquer avec cette dernière. Cette famille de schémas, au fonctionnement asymétrique, est appelée *Cryptographie Asymétrique* ou *Cryptographie à clé publique*.

Chiffrement homomorphe. Parmi les schémas de chiffrement à clé publique, certains ont une propriété homomorphe. Cette propriété permet de faire des opérations sur les valeurs claires (non chiffrées), à partir de leurs valeurs chiffrées. Considérons deux valeurs v_1 et v_2 chiffrées avec un schéma de chiffrement à clé publique dont la clé publique est pk (pour *public key*, en anglais).

Nous obtenons donc $c_1 = \mathsf{Enc}(\mathsf{pk}, v_1)$ et $c_2 = \mathsf{Enc}(\mathsf{pk}, v_2)$. Un schéma de chiffrement homomorphe permet de calculer un chiffré de $v_1 + v_2$ ou de $v_1 \times v_2$ sans connaître ni v_1, ni v_2. Par exemple, avec le cryptosystème de ElGamal, multiplier les chiffrés permet d'obtenir un produit des valeurs claires. Autrement dit, en multipliant c_1 par c_2, nous obtenons un chiffré de $v_1 \cdot v_2$ sans connaître ni v_1, ni v_2.

Code d'authentification de message. Un Code d'Authentification de Message – *Message Authentication Code* (MAC) en anglais – permet à Bob de s'assurer que le message reçu n'a pas été altéré. Dans ce cas, conjointement avec le message chiffré, Alice calcule un *tag*, une valeur calculée en fonction de la clé secrète partagée avec Bob et du message chiffré. Ensuite, Bob peut s'assurer à l'aide du *tag* et de la clé secrète que le message n'a pas été altéré. En effet, une propriété des Codes d'Authentification de Message est qu'ils sont difficilement falsifiables.

Chiffrement authentifié. Un chiffrement authentifié est un schéma de chiffrement qui inclut un Code d'Authentification de Message. Cette authentification se fait (dans ce que nous verrons dans cette thèse) en générant une valeur, appelée *tag* à l'aide d'une clé secrète, et du message à chiffrer. Par la suite, quand Bob déchiffre le message reçu par Alice, le processus de déchiffrement inclus une vérification du *tag*, et ne poursuit le déchiffrement que si la vérification réussit. De plus, similairement aux Codes d'Authentication de Message, il doit être difficile pour une adversaire de falsifier un chiffrer, lui permettant de passer la vérification du *tag*.

Transfert Inconscient. Supposons que Renée (deuxième prénom d'Alice) soit dans la plus petite banque du monde ne contenant que deux coffres-forts. L'un de ces coffres lui appartient et elle souhaite y accéder. Pour ce faire, Renée doit récupérer la clé auprès du directeur en qui elle a confiance. Malheureusement, le directeur est en congés et c'est son assistant, Edouard (deuxième prénom de Bob), en qui elle n'a pas confiance qui la reçoit. Renée ne souhaite pas lui dire quel coffre lui appartient, par soucis de confidentialité, et, logiquement, ce dernier ne veut pas lui donner les clés de chacun des coffres. La solution pour Renée est d'utiliser un Transfert Inconscient de clé, dont le processus idéal est représenté en Figure 1a.1. Renée sera la Réceptrice de la clé et Edouard sera celui qui l'enverra, donc l'Expéditeur. Nous dénotons par C_1 et C_2 les clés détenues par Edouard, et décrétons que Renée détient le coffre numéro 1.

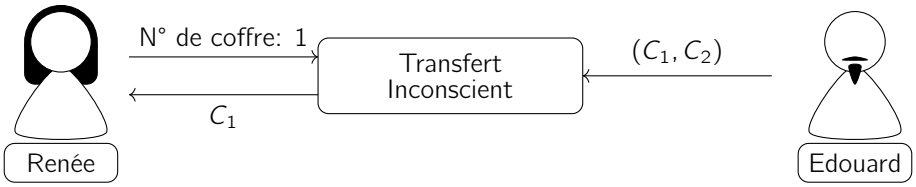
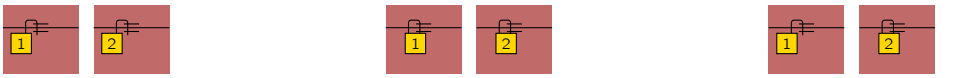


Figure 1a.1: Protocole de Transfert Inconscient

Le processus idéal décrit ci-dessus fait appel à un tiers de confiance qui récupère les deux clés d'Edouard et donne à Renée la clé qu'elle souhaite. Malheureusement, Renée et Edouard sont seuls et doivent recréer ce processus, à eux deux. Une façon pour Renée et Edouard de procéder afin de recréer ce processus, est de suivre les étapes suivantes.

- *Première étape* : Renée prend deux boîtes qui se ferment à l'aide d'un cadenas. Elle ouvre les deux cadenas et garde la clé de la première boîte, puis, agissant de manière honnête, jette la seconde sous les yeux d'Edouard. Ensuite, elle dispose sur chaque boîte le cadenas encore ouvert qui lui correspond, puis donne les boîtes à Edouard.
- *Deuxième étape* : Edouard met la clé du coffre numéro 1 dans la boîte numéro 1, et la clé du coffre numéro 2 dans la boite numéro 2, ferme les deux cadenas, puis renvoie les boîtes avec les cadenas fermés à Renée.

- *Troisième étape* : Renée utilise la clé de cadenas qu'elle a gardé pour ouvrir la boîte numéro un, à l'abri du regard d'Edouard.



(a) Première étape (b) Deuxième étape (c) Troisième étape

Figure 1a.2: État des boîtes pendant le protocole

De façon plus générale, un protocole de Transfert Inconscient permet à Renée de récupérer un élément auprès d'Edouard sans qu'il sache quel élément a été récupéré par Renée, et sans que Renée n'apprenne d'information sur l'élément qu'elle n'a pas reçu d'Edouard.

Preuves à Divulgation Nulle de Connaissance. Commençons par définir la notion de langage. Un langage $\mathcal{L}$ avec relation $\mathfrak{R}$ est un ensemble d'éléments x, appelés mots, pour lesquels il existe un témoin w tels que $\mathfrak{R}(x, w) = 1$. Le principe d'une Preuve à Divulgation Nulle de Connaissance est de prouver qu'un élément appartient à un langage, sans révéler d'information sur le témoin.

Modèle de l'oracle aléatoire. Pour simuler des valeurs aléatoires, nous utilisons des fonctions retournant des valeurs pseudo-aléatoires, appelées pseudo-random functions. Dans le cadre de preuves de sécurité, nous faisons l'hypothèse que ces valeurs sont parfaitement aléatoires en les modèlisant à l'aide d'un oracle aléatoire. Quand de telles modèlisations sont faites, nous nous plaçons dans le modèle de l'oracle aléatoire.

Smooth Projective Hash Functions. Les *Smooth Hash Projective Functions*, ou SPHF utilisent, elles aussi, la notion de langage. Une SPHF définie sur un langage $\mathcal{L} \subset \mathcal{X}$ avec valeurs dans $\mathcal{V}$, est définie ainsi :

- $\mathsf{Setup}(1^\kappa)$ génère les paramètres globaux param qui incluent une description de $\mathcal{L}$.
- $\mathsf{HashKG}(\mathsf{param})$ génère une clé de hash aléatoire hk. Retourne hk.
- $\mathsf{ProjKG}(\mathsf{hk}, x)$ dérive la clé de projection hp depuis hk et un mot x. Retourne hp.
- $\mathsf{Hash}(\mathsf{hk}, x)$ calcule la valeur hachée $H_{\mathsf{hk}} \in \mathcal{V}$ associée au mot x. Retourne H_{hk}.
- $\mathsf{ProjHash}(\mathsf{hp}, x, w)$ calcule $H_{\mathsf{hp}} \in \mathcal{V}$ utilisant un témoin w lié au mot x. Retourne H_{hp}.

Ainsi, si x est un mot de $\mathcal{L}$, nous avons $H_{\mathsf{hk}} = H_{\mathsf{hp}}$. Cette propriété est appelée **Correctness**. De plus, si x n'est pas un mot de $\mathcal{L}$, non seulement $H_{\mathsf{hk}} \neq H_{\mathsf{hp}}$, mais le couple $(\mathsf{hp}, H_{\mathsf{hk}})$ doit aussi être indiscernable de tout autre couple (hp, v) où v est un élément aléatoire de $\mathcal{V}$. Cette propriété est appelée **Smoothness**.

Forêt de Décision. Une forêt de décision est un ensemble d'arbres de décision. Un arbre de décision est une succession de nœuds de décision, où chaque nœud de décision consiste en un test. Quand un échantillon est testé, selon le résultat du test, il est redirigé vers le nœud enfant correspondant. Au bout d'un certain nombre de comparaisons, l'échantillon atteindra une feuille (au sens feuille de l'arbre) de décision, indiquant la décision associée à cet échantillon.

Nous présentons en Figure 1a.3 un arbre de décision permettant de décider quel terme employer pour désigner une viennoiserie contenant des barres de chocolat selon votre position géographique. Les feuilles de décision contiennent la bonne expression à employer.

Bien évidemment, les décisions sont souvent plus complexes à prendre que dans cet exemple et nécessitent de considérer un grand nombre de critères. De plus, ces arbres sont généralement construits à partir d'un grand nombre d'observations dont la décision finale est connue. Dans notre

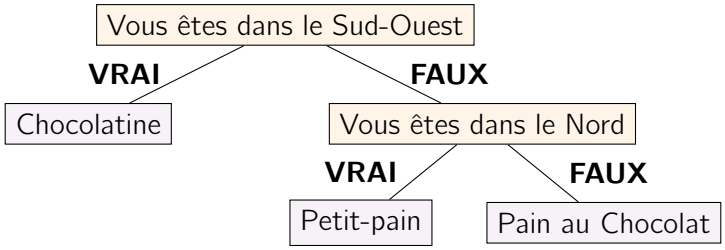


Figure 1a.3: Arbre de décision pour désigner une viennoiserie contenant des barres de chocolat.

exemple, ces observations correspondraient à un sondage où nous demandons à des personnes leur ville d'origine ainsi que le terme qu'ils emploient. Le principe d'une forêt de décision est un ensemble d'arbres ou chaque arbre ne considère qu'une partie aléatoire des critères, ainsi qu'une partie aléatoire des observations. Cela permet d'éviter le surapprentissage, qui donnerait trop d'importance à une observation déviant des autres. Toujours dans notre exemple, cela pourrait se traduire comme suit. Supposons que dans nos observations un sondé vienne d'un village, dont il est le seul sondé. Contrairement à tous les résidents des villages alentours, il emploie le terme pain au chocolat. Le surapprentissage aurait pour conséquence qu'une personne se retrouvant dans ce même village, recevra la "décision" d'utiliser le terme pain au chocolat, alors que le terme chocolatine serait plus approprié.

La décision finale se fait en prenant la décision qui est la plus souvent retournée par les arbres de décision.

1.5.2 Évaluation sécurisée de forêts de décision

Comme énoncé plus haut, notre première contribution a été d'ajouter une couche cryptographique à l'évaluation de Forêts de Décision dont la décision finale est une valeur binaire. Nous considérons dans tous les cas que le Serveur a été honnêtement implémenté par Selim, mais que ce dernier peut essayer d'apprendre des informations sur la cliente, Christelle, à partir de ce qu'il reçoit. Ainsi, nous utiliserons Selim pour parler au nom du Serveur. Nous commençons par le cas où la cliente, Christelle, est honnête-mais-curieuse, c'est-à-dire qu'elle suit le protocole, mais qu'elle peut tenter d'extraire des informations à partir des messages qu'elle reçoit.

1.5.2.1 Cliente honnête-mais-curieuse

La première étape consiste à faire en sorte que Selim envoie de manière sécurisée son modèle à Christelle, tout en étant exploitable par cette dernière. Dans un premier temps, nous avons décidé de représenter les arbres comme un ensemble de branches, comme l'illustre l'exemple donné en Figure 1a.4. Dans cet exemple, nous utilisons la base de données *iris* qui permet de classifier un iris à partir de la longueur ℓ et la largeur w de son pétale. Les données de la base n'étant pas entières, la première étape consiste à les normaliser. Pour ce faire, nous définissons $L = 10 \times (\ell - 1)$ et $W = 20 \times w$. Ainsi, toutes les valeurs de la base seront comprises entre 0 et $63 = 2^6 - 1$. Pour en faire un arbre de décision binaire, nous nous concentrons sur la décision affirmant ou non que l'iris en question est de l'espèce *virginica* ou non.

Étant donné qu'un pétale ne peut aboutir qu'à une seule feuille, il n'est pas nécessaire de considérer les feuilles ayant 0 pour décision finale. En effet, si un pétale n'atteint aucune des branches dont la décision finale est 1, cela implique que la décision finale est 0. C'est d'ailleurs cette observation qui nous contraint à considérer les arbres binaires. Ensuite pour chacune des branches, nous allons procéder comme suit. Pour chaque décision, nous allons utiliser un schéma de chiffrement additivement homomorphe pour générer un chiffré pour chaque valeur possible prise par l'échantillon pour le critère demandé, dont la valeur chiffrée dépendra de la valeur du seuil. Par

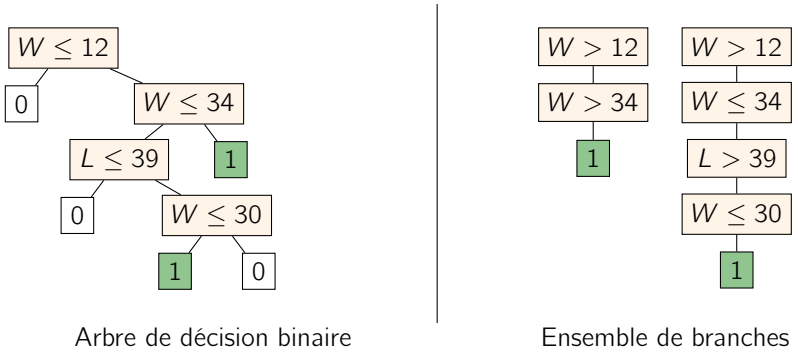


Figure 1a.4: Représentation d'un arbre de décision entraîné sur la base de données *iris* dans le cas honnête-mais-curieux

exemple, prenons la première comparaison de la première branche : $(W > 12)$. Si 64 valeurs sont possibles (de 0 à 63), avec un seuil à 12, nous générons dans un premier temps 13 chiffrés de 0, correspond aux valeurs comprises de 0 à 12, qui ne vérifient pas la comparaison ; puis 51 chiffrés de 1, correspondant aux valeurs de 13 à 63 qui vérifient la comparaison. Une fois que toutes les comparaisons sont encodées, Selim les envoie à Christelle.

Une fois que Christelle a reçu les branches encodées par Selim, elle récupère le chiffré correspondant à la valeur de son critère. En procédant ainsi pour chacun des nœuds, cela nous permet d'exploiter la propriété homomorphe du schéma de chiffrement. En effet, si toutes les comparaisons sont vérifiées pour une branche, Christelle n'aura que des chiffrés de 0, qu'elle pourra sommer grâce à la propriété homomorphe du schéma de chiffrement, aboutissant à un chiffré de 0. Dès qu'une des comparaisons n'est pas vérifiée, la somme ne vaut plus 0, et prend une valeur comprise entre 1 et la longueur de la branche. Rappelons que Christelle ne sait rien de la valeur des chiffrés qu'elle possède, que ce soit avant ou après la somme. Elle pourrait renvoyer les chiffrés obtenus directement à Selim pour qu'il les déchiffre et puisse prendre sa décision. Cependant, cela révèlerait le score de chaque branche. Pour empêcher Selim de récupérer la moindre information supplémentaires sur le score de ses branches, Christelle a deux modifications à apporter. La première consiste à permuter les branches pour empêcher Selim d'associer un score à une branche. Néanmoins, il pourra toujours calculer le score moyen des branches, ce qui constitue une information complémentaire comparativement à ce qu'il doit apprendre, c'est-à-dire le nombre de branches qui sont validées. C'est pourquoi Christelle multiplie les chiffrés obtenus par une valeur aléatoire. En multipliant un chiffré de 0 par une valeur aléatoire, cela reste un chiffré de 0, alors qu'en multipliant un chiffré d'une valeur non-nulle par une valeur aléatoire, cela devient un chiffré d'une valeur aléatoire. Ainsi, avoir permuté et randomisé ses scores, Christelle peut les envoyer à Selim.

Une fois en possession des scores chiffrés de Christelle, Selim les déchiffre. S'il obtient un 0 en déchiffrant, cela signifie qu'une branche a été validée. Le score global de Christelle se détermine donc en comptant le nombre de chiffrés qui ont donné 0 en étant déchiffrés. Ensuite, si ce score est suffisamment élevé pour Selim, ce dernier valide l'échantillon de Christelle. Sinon, il le rejette.

1.5.2.2 Cliente malveillante

Au vu de la description faite ci-dessus, si Christelle est malveillante et qu'elle ne suit pas le protocole, il est aisé pour elle de duper Selim. En effet, elle pourrait outrepasser les évaluations des branches et générer suffisamment de chiffrés de 0 pour être validée par Selim. Pour contrer cette attaque, la solution est de faire en sorte que Christelle ne sache pas quelle valeur est attendue par le serveur. Pour ce faire, nous allons à présent considérer tous les nœuds de décision dont au moins un des

nœuds fils est une feuille. Puis, pour chacun de ces nœuds, nous allons aléatoirement générer une valeur appelée *polarité*, qui déterminera l'orientation de la dernière comparaison. Quand cette polarité vaut 1, la dernière comparaison reste inchangée (en bleu). Tandis que si cette polarité vaut -1, la dernière comparaison est inversée (en rouge). Nous utilisons aussi des comparaisons factices pour éviter à Christelle de pouvoir reconstruire l'arbre en utilisant des branches toutes de même longueur et utilisant tout le temps les mêmes critères. Dans la Figure 1a.5, nous reprenons le même exemple que dans la Figure 1a.4, mais en représentant les arbres comme nous venons de le décrire. De plus, nous faisons usage de noeuds factices dont le résultat est toujours vrai ($L \leq 63$ et $W \leq 63$) ainsi que des noeuds dont le résultat est toujours faux ($W > 63$) – les valeurs prisent par L et W étant comprises entre 0 et 63. Grâce à cees noeuds factices, toutes les branches sont de même longueur et utilisent le même nombre de fois les critères.

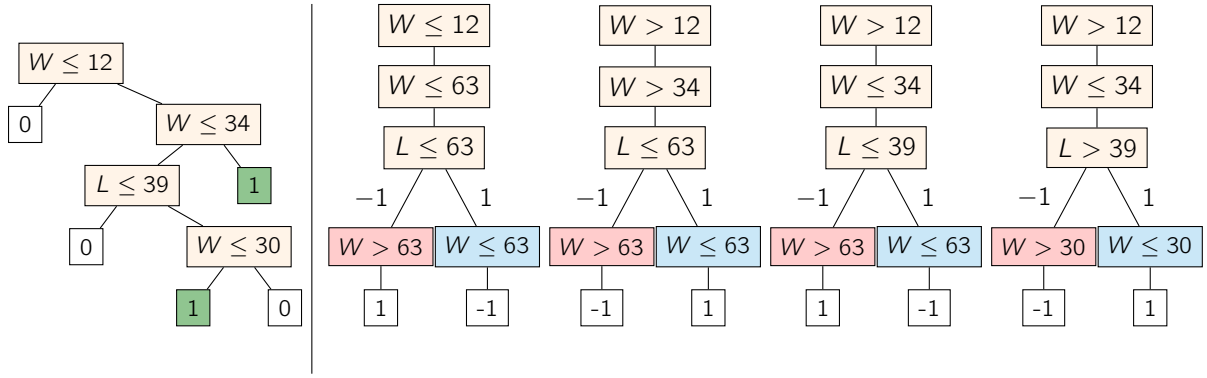


Figure 1a.5: Représentation d'un arbre dans le cas d'une cliente malveillante. Quand la polarité vaut -1, le dernier nœud de décision est celui en rouge, tandis que si la polarité vaut 1, le dernier nœud de décision est celui en bleu.

Ensuite, pour chaque chemin, jusqu'à la pénultième comparaison, les valeurs vérifiant la comparaison seront encodées par des chiffrés de 1 (là où ils étaient encodés par des chiffrés de 0 dans le cas précédent) et les valeurs ne la vérifiant pas seront encodées par des chiffrés de 0 (là où ils étaient encodés par des chiffrés de 1 dans le cas précédent). Pour la dernière comparaison, les valeurs allant dans le sens de la polarité seront encodées par des chiffrés de δ, où δ est le nombre de comparaisons dans la branche, et les valeurs n'allant pas dans le sens de la polarité seront encodées par des chiffrés de 0.

Ensuite, Christelle fait la somme, comme dans le cas précédent. Ainsi, si un échantillon passe toutes les comparaisons, la dernière comprise, sa somme sera de $2\delta - 1$. S'il passe toutes les comparaisons sauf la dernière, la somme sera de $\delta - 1$. Dans tous les autres cas, la somme peut être une valeur comprise entre 0 et $\delta - 2$, et δ et $2\delta - 2$.

La prochaine étape consiste à convertir cette somme en un score, qui vaut $-1, 0$ ou 1. Nous ne détaillerons pas le processus complet ici, mais l'idée est la suivante : il faut convertir un chiffré de $2\delta - 1$ en un chiffré de 1, un chiffré de $\delta - 1$ en un chiffré de -1, et tous les autres chiffrés en un chiffré de 0.

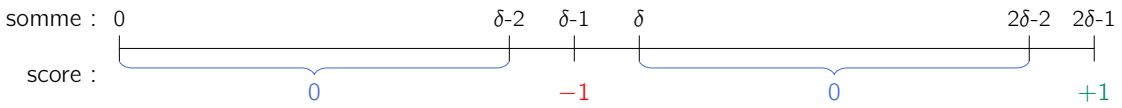


Figure 1a.6: Correspondance entre résultat et score

L'idée est de comparer, de manière sécurisée, la somme obtenue par Christelle avec $\delta - 1$ et $2\delta - 1$. Si la première comparaison est vraie, Christelle retourne à Selim un chiffré de -1, si elle

est fausse, elle retourne un chiffré de 1. Pour la deuxième comparaison, c'est l'inverse. Christelle retourne un chiffré de 1 si elle est vraie, sinon elle retourne un chiffré de -1. Notre construction inclut des preuves garantissant que Christelle envoie bien les valeurs qu'elle est censée retourner à Selim, l'empêchant ainsi d'essayer de modifier son score. Ensuite, Selim fait la somme des chiffrés. Il obtient un chiffré de -2 si la première égalité est vraie (si la première est vraie, la deuxième est forcément fausse), un chiffré de 2 si la deuxième est vraie, et un chiffré de 0 dans tous les autres cas. En divisant ce chiffré par 2, nous retrouvons bien ce qui était espéré en Figure 1a.6. La prochaine étape consiste à multiplier ce score par sa polarité, ce qui donne 1 si l'échantillon est accepté par la branche, -1 si l'échantillon est rejeté par la branche, et 0 dans tous les autres cas.

Rappelons que jusqu'ici, Selim n'est en possession que de valeurs chiffrées par Christelle, qu'elle seule peut déchiffrer. Selim va alors additionner tous les chiffrés obtenus (un par branche), puis randomiser le total, avant d'envoyer le résultat obtenu à Christelle pour qu'elle le déchiffre. Cette dernière déchiffre le score global randomisé, qu'elle retourne à Selim, qui inverse la randomisation pour retrouver le score global de Christelle. Dû à la randomisation faite par Selim, la moindre tentative de triche faite par Christelle aura pour conséquence un score final totalement aléatoire, et avec forte probabilité en dehors de l'intervalle d'acceptation de Selim.

Coûts. Dans chacun des deux contextes, le modèle encodé est envoyé hors protocole. C'est-à-dire qu'il n'est envoyé qu'une seule fois pour plusieurs exécutions faites par la cliente. La taille de l'encodage dépend de plusieurs paramètres : le nombre de branches, le nombre de valeurs possibles prises par les critères et la longueur des branches. Plus un paramètre ci-dessus est élevé, plus le résultat sera précis, mais plus le modèle sera lourd. Le meilleur compromis est donc propre à l'utilisation qui en sera faite. Le poids du modèle va de 5 Ko pour 10 branches de longueur 2 et 4 valeurs possibles par critères, à 25 Mo pour 100 branches de longueur 16 et 256 valeurs par critère. Nous référons le lecteur à la Table 3.1 pour plus détails.

Pendant l'exécution du protocole dans le cas honnête-mais-curieux, Christelle envoie un chiffré de 512 bits par branche et le serveur n'envoie rien. Dans le cas d'une exécution avec une cliente malveillante, cela impacte les coûts de communication. La cliente envoie un peu plus de 20 Ko par branche et le serveur envoie 4 Ko par branche.

Étant donné que la cliente n'envoie qu'un seul message au serveur par exécution du protocole, cela surpasse tous les autres protocoles similaires, au prix d'une phase hors protocole assez conséquente (mais qui n'est faite qu'une fois pour plusieurs protocoles). Dans le cas malveillant, notre protocole est un compromis entre le nombre de messages envoyés et le coût de communication global du protocole. Voir Table 3.3 pour plus de détails.

Performances. Nos constructions ont été mises à l'épreuve à travers deux applications faisant appel aux arbres de décision : l'authentification continue et la détection de Spam. Dans les deux cas, nous obtenons un taux de faux positifs autour de 2% pour une précision globale autour de 90%.

1.5.3 Transfert Inconscient Universellement Composable basé sur les SPHF avec zone grise

Comme énoncé plus haut, les Transferts Inconscients peuvent reposer sur la notion de *Smooth Projective Hash Functions with Gray Zone*, une primitive que nous avons introduite qui consiste en une SPHF avec la propriété d'Insolubilité de la Décomposition. Considérons $\mathcal{X}$, un ensemble de mots, $\mathcal{L}$ et $\mathcal{L}$' deux langages tels que $\mathcal{L} \cap \mathcal{L}' = \emptyset$ et $\mathcal{L} \cup \mathcal{L}' \subset \mathcal{X}$. Une *Smooth Projective Hash Function with Gray Zone* est alors définie comme :

- Setup(1^κ) génère les paramètres globaux param qui incluent une description de $\mathcal{L}$ et $\mathcal{L}$' tels que $\mathcal{L} \cap \mathcal{L}' = \emptyset$ et $\mathcal{L} \cup \mathcal{L}' \subset \mathcal{X}$.
- HashKG(param) génère une clé de hash aléatoire hk. Retourne hk.
- ProjKG(hk, x) dérive la clé de projection hp depuis hk et un mot x. Retourne hp.
- Hash(hk, x) calcule la valeur hachée $H_{\mathsf{hk}} \in \mathcal{V}$ associée au mot x. Retourne H_{hk}.
- ProjHash(hp, x, w) calcule $H_{\mathsf{hp}} \in \mathcal{V}$ utilisant un témoin w lié au mot x. Retourne H_{hp}.

Tout comme les SPHF classiques, nos SPHFwGZ verifient les propriétés suivantes, pour tout ensemble $\mathcal{L}, \mathcal{L}', \mathcal{X}$ définis à l'aide de param.

- **Correctness:** Pour tout x dans $\mathcal{L}$, $H_{\mathsf{hk}} = H_{\mathsf{hp}}$, où hk $\leftarrow$ HashKG(param), hp $\leftarrow$ ProjKG(hk, x), $H_{\mathsf{hk}} \leftarrow$ Hash(hk, x), et $H_{\mathsf{hp}} \leftarrow$ ProjHash(hp, x, w) pour le témoin w de $x \in \mathcal{L}$;
- **Smoothness:** Pour tout x dans $\mathcal{L}'$, les distributions de (hp, H_{hk}) et (hp, v) sont indiscernables, où hk $\leftarrow$ HashKG(param), hp $\leftarrow$ ProjKG(hk, x), $H_{\mathsf{hk}} \leftarrow$ Hash(hk, x), et $v \leftarrow\!\!\$\ \mathcal{V}$;

En supplément, les ensembles $\mathcal{L}, \mathcal{L}'$ et $\mathcal{X}$ doivent aussi fournir l'Insolubilité de la Décomposition.

Pour définir de manière plus formelle la notion d'Insolubilité de la Décomposition, nous introduisons la fonction ComplementWord:

- ComplementWord(ρ, x) calcule, à partir d'un mot x et d'une valeur de référence ρ, le complémentaire x' de x. Retourne x'.

Ainsi, les ensembles $\mathcal{L}, \mathcal{L}'$ et $\mathcal{X}$ fournissent l'Insolubilité de la Décomposition s'il est difficile, étant donné une valeur aléatoire ρ dans $\mathcal{X}$, de générer deux valeurs x et y hors $\mathcal{L}'$ de sorte que $y =$ ComplementWord(ρ, x).

1.5.3.1 Construction du protocole de Transfert Inconscient

Après la configuration du protocole, la réceptrice, Renée, génère un mot x_b dans $\mathcal{L}$ et son témoin, à l'aide de la fonction WordGen$\mathcal{L}$:

- WordGen$\mathcal{L}$(param) génère aléatoirement x dans $\mathcal{L}$, et retourne x ainsi que son témoin w.

Puis à l'aide de ρ, une valeur de référence commune à la réceptrice et à l'expéditeur, Renée génère x_{1-b}, le complément de x_b relativement à ρ. Grâce à l'Insolubilité de la Décomposition, x_{1-b} est dans $\mathcal{L}'$. Ensuite, Edouard calcule les clés de hash et de projection sur chacun des mots ainsi que les hash associés, qui serviront de masque pour les messages initiaux m_0 et m_1.

Dû à la propriété de correctness de x_b dans $\mathcal{L}$, Renée pourra démasquer m_b, le mot qu'elle souhaite récupérer. À l'inverse, comme x_{1-b} est dans $\mathcal{L}'$, la propriété de smoothness assure que Renée ne peut pas calculer m_{1-b}. En reprenant l'exemple pédagoqique de Transfert Inconscient donné plus haut, x_b et x_{1-b} peuvent être vus comme les cadenas, et le témoin w associé à x_b en est la clé. Edouard 'ferme' les cadenas en utilisant Hash, que Renée ouvre en utilisant ProjHash et w.

Pour réaliser la preuve – qui ne sera pas détaillée ici – que le protocole est universellement composable, ce dernier prend une deuxième valeur commune σ. Ces valeurs communes, quand elles sont générées depuis l'ensemble adéquat, permettent à la partie honnête d'extraire les inputs des parties malveillantes, les dissuadant ainsi d'agir de manière malveillante. Plus précisément, si Renée, la réceptrice, est malveillante, Edouard sera en mesure d'extraire b, que Renée ne souhaite pas divulguer. De manière similaire, Renée sera en mesure de récupérer les deux messages m_0 et m_1 si Edouard est malveillant.

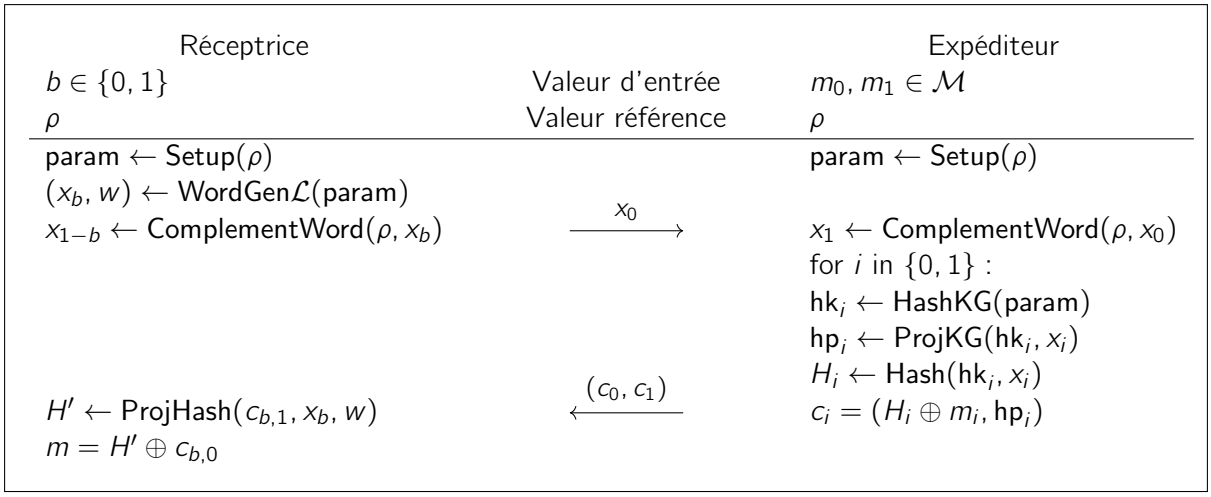


Figure 1a.7: Description du protocole de Transfert Inconscient

Configuration depuis un schéma de chiffrement homomorphe. Dans cette section, nous détaillons la configuration des langages $\mathcal{X}, \mathcal{L}, \mathcal{L}'$ à partir d'un schéma de chiffrement homomorphe, possiblement avec des échecs de déchiffrement. Un tel schéma de chiffrement sera dénoté par $\Pi = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$.

Ainsi, $\mathcal{X}$ se définit comme étant l'ensemble des chiffrés générés par Π, tandis que $\mathcal{L}$ est l'ensemble des chiffrés de 0 et $\mathcal{L}'$ est l'ensemble des chiffrés ne se déchiffrant pas en 0. De manière plus formelle, nous avons :

$$\mathcal{L} = \{\mathsf{Enc}(\mathsf{pk}, 0; r)\} \subset \mathcal{X} \text{ et } \mathcal{L}' = \{c \in \mathcal{X}, \mathsf{Dec}(\mathsf{sk}, c) \neq 0\} \subset \mathcal{X}.$$

En guise d'instanciation introductive, nous décrivons un SPHFwGZ basé sur le schéma de chiffrement d'ElGamal. Bien que sa zone grise soit vide, cela appuie le fait que notre construction peut se généraliser à tout SPHF. La seconde instanciation repose sur le schéma de chiffrement LWE. Ce schéma se base sur les réseaux euclidiens, qui sont une primitive post-quantique et présente une zone grise, qui ne pose pas de problème dans notre construction, car la propriété d'Insolubilité de la Décomposition est vérifiée.

1.5.4 Analyse de sécurité du protocole EDHOC

EDHOC est un protocole authentifié d'échange de clés optimisé pour être exécuté sur des appareils de l'Internet des Objets. Il permet à deux parties, un Initiateur et un Répondant, de partager une clé commune à la fin de l'exécution du protocole. Pour l'authentification, les deux parties peuvent soit utiliser des signatures digitales, soit utiliser une clé secrète statique.

C'est dans le contexte de l'utilisation de cette seconde solution pour l'Initiatrice et le Répondant que nous étudierons le protocole. Ce dernier va également générer une paire de clés éphémères (une publique, une secrète) pour chacune des parties, qui seront utilisées uniquement pour l'exécution dudit protocole. Ainsi, chaque partie possède quatre clés :

Ensuite les clés ci-dessus sont dérivées à l'aide de fonction d'expansion et d'extraction afin d'obtenir la clé de session finale. Nous n'allons pas détailler le processus de calcul de la clé ici (voir Chapitre 5, Figures 5.2 et 5.3), ni l'analyse, mais allons plutôt directement présenter les résultats obtenus.

Pour rappel, les trois points de sécurité à analyser sont les suivants

- *Confidentialité de la clé* : Au plus les deux participants connaissent la clé de session finale.

X_e, x_e	Clés publique et secrète éphémères de l'Initiatrice
Y_e, y_e	Clés publique et secrète éphémères du Répondant
X_s, x_s	Clés publique et secrète statiques de l'Initiatrice
Y_s, y_s	Clés publique et secrète statiques du Répondant

Table 1a.1: Clés des deux parties. En orange les clés du Répondant et en blanc, les clés de l'Initiatrice

De plus, nous devons prouver la *Perfect Forward Secrecy* qui stipule que, même si les clés statiques sont dévoilées, un attaquant ne peut pas calculer les précédentes clés de session.

- *Authentification Mutuelle* : Seuls les deux participants ont le matériel pour calculer la clé de session finale.
- *Protection de l'identité* : Au plus les deux participants connaissent l'identité des deux participants.

Pour la réalisation de la preuve, nous nous sommes placés dans le modèle de l'oracle aléatoire dans lequel les valeurs retournées par les fonctions d'expansion et d'extraction sont considérées comme étant parfaitement aléatoires.

Résultats de l'analyse. Sous les hypothèses de la difficulté du problème *Gap Diffie-Hellman*, de la sécurité sémantique et le caractère infalsifiable du chiffrement authentifié ainsi que dans le modèle de l'oracle aléatoire, le protocole EDHOC atteint les niveaux de sécurité suivants : environ 128 bits de sécurité pour la confidentialité de la clé ainsi que pour la protection de l'identité de chacune des parties. Cependant, pour la sécurité de l'authentification mutuelle, la sécurité n'atteint pas un niveau supérieur à 64 bits. Bien que cela soit prévu pour le Répondant qui peut atteindre une sécurité de 128 bits pour son authentification en envoyant un quatrième message comme le suggère la documentation du protocole, ce n'est pas le cas pour la sécurité de l'authentification de l'Initiatrice. En effet, la seule valeur envoyée par l'Initiatrice qui dépend de sa clé statique x_s est un Code d'Authentification de Message de 64 bits de long (dû aux paramètres agressifs imposant cette longueur). Ainsi, un attaquant peut deviner ce tag avec probabilité $1/2^{64}$, ce qui implique la sécurité maximale de 64 bits. Bien qu'un chiffrement supposé authentifié soit envoyé par l'Initiatrice, il peut être calculé par n'importe quel individu souhaitant usurper l'identité de l'Initiatrice s'il devine correctement le Code d'Authentification de Message évoqué plus haut. Le choix d'utiliser un chiffrement authentifé plutôt qu'un chiffrement classique n'a donc aucun impact sur la sécurité de l'authentification de l'Initiatrice.

De plus, la sécurité de chacun de ces points dépend du nombre de protocoles simultanément exécutés. En effet, dû à l'utilisation d'une chaîne vide en tant que valeur d'entrée d'une fonction d'extraction, un attaquant peut exploiter une seule requête à l'oracle DDH pour attaquer plusieurs protocoles.

Nous présentons donc maintenant des améliorations afin d'améliorer la sécurité du protocole EDHOC

Améliorations. Une première amélioration consiste à remplacer la chaîne vide ci-dessus par une valeur dépendant de la session courante. Cela permet de rendre la sécurité globale indépendante du nombre de sessions exécutés simultanément. La seconde amélioration consiste à passer la taille du Code d'Authentification de Message envoyé par l'Initiatrice de 64 bits à 128 bits, et de remplacer le chiffrement authentifié par un chiffrement *One-Time Pad*. Ainsi, la sécurité de l'authentification de l'Initiatrice atteint 128 bits, et ce, sans surcoût de communication ni de calcul. En effet, les 64 bits du chiffrement authentifié sont transposés au Code d'Authentification de Message. Ainsi, le

chiffrement n'a plus besoin d'être authentifié et le *One-Time Pad* peut être utilisé sans modifier la longueur du chiffré.

Chapter 1b

Introduction – English version

2.1 Motivations

With the recent deployment of the General Data Protection Regulation (GDPR), confidentiality has become an essential requirement. From secret messages on pieces of paper in the playground to inter-presidential communications, cryptography easily finds its place in human-to-human communications. Nowadays, the spectrum of use of cryptography is getting wider by the day. Cryptography is no longer limited to interactions between two physical persons, but also involves servers, phones, etc... Any device connected to the Internet uses cryptography. However, even if on the one hand, protecting data is obviously a good feature, on the other hand, someone could need to extract data. If we use the example of messaging services, anyone expects that only you and the person you are messaging with can get access to your messages. However, guaranteeing such a high confidentiality level also attracts malicious people, potentially using this messaging service to send messages whose content may be illegal. One could therefore hope for the messaging server to detect such content. Then, how to correctly balance confidentiality and data exploitation? A domain of cryptography studying this problem is the secure multiparty computation. In short, secure multiparty computation allows some parties to jointly and securely evaluate a function on their inputs, while keeping their inputs private. Applied to our example, this would consist for a server to be able to detect when a message contains jeopardizing content.

When data is collected, it is often used as an input to some Machine Learning model. Machine Learning is a powerful tool with many use-cases: Email filtering, fraud detection, personalized marketing, etc... Such protocols are often run on a server receiving data from an other device. However, in some cases, as pictures improvement or continuous authentication, models are run on the device itself. We now focus on the latter use-case. Continuous authentication seamlessly authenticates a user on his phone. This can be made by using the swipes he makes, taking into account several features as the swipe duration, the pressure, etc... Obviously, such data are sensitive. But the authentication model, that can be represented as a list of criteria to respect, also needs to remain private to avoid impersonation. This problem is similar to the one described above. Indeed, this can be reworded as the question:

How to use Machine Learning for Continuous Authentication while remaining privacy-friendly?

We provide an answer to this question in the third part of this thesis. Our answer provides a protocol secure against an honest-but-curious and its upgrade to provide security against malicious adversaries. An honest adversary follows the protocol but tries to extract as much information as he can whereas the malicious one can deviate from the protocol to learn more information or influence the output of the protocol. The protocol secure against malicious adversaries makes use of a two-party computation primitive named Oblivious Transfer allowing (in its simplest form) a

receiver to receive one out of two messages from a sender. At the end of an Oblivious Transfer protocol, the receiver did learn nothing about the second message of the sender, and the sender has no clue which message has been retrieved by the receiver. Several Oblivious Transfer (OT) protocols instantiations do exist and among them, we focus on those based on Smooth Projective Hash Functions (SPHF) (also known as Hash Proof Systems). SPHFs work using a language, made of words, a pair of hash functions and a pair of keys, such that when a word of the SPHF belongs to the language, the keys, together with their respective hash functions led to the same value. This property is called correctness. Also, when a word is not in the language, the hash outputs should be different and indistinguishable from random. This property is called smoothness. Oblivious Transfers based on Smooth Projective Hash Functions are well studied and efficient when based on classical cryptography. However, when using SPHF based on post-quantum cryptography rise words where one can not claim correctness nor smoothness for this subset of words, denoted Gray Zone. Ignoring them would leave room for potential attacks. Hence, we follow some recent works on this question:

How to deal with those Smooth Projective Hash Functions with Gray Zone?

Moreover, as Oblivious Transfers are part of a larger protocol, we need to ensure that composing it with other protocols does not bring security weakness for the root protocol. A protocol providing such security is said to be Universally Composable. Thus, we expect our Oblivious Transfer security to be proven in the Universal Composability framework.

As said at the beginning of this section, cryptography now involves not only humans but also devices, as server, mobile phones, etc... More specifically, more and more products get connected to the Internet in our houses. Speakers, light bulb, fridges, ovens and even trucks fuel tank cap are some examples of the devices connected to the Internet. Those products may be intended to make our life easier, funnier or safer. We can imagine a surveillance camera that may tell your heating to turn on when your car is detected (using Machine Learning), and the same camera sending you a video stream of what it sees when detecting unexpected presence. This world where everything and anything can be connected to the Internet is called *Internet of Things*, or IoT in short. In 2021, the number of active IoT devices in the world is estimated at around 12 billions, while at the same time, the mondial population was estimated at less than 8 billions. However, such devices are often limited in terms of energetic consumption, computing power or available memory. The domain studying cryptography on restraint devices is called *lightweight cryptography*. In such domain, the goal is to design primitives that provide the best trade-off between efficiency, costs, and security. One of the those primitives is lightweight key exchange protocols, where two constrained devices end with a common shared key. Among the lightweight key exchange protocols, some of them provide an additional authentication of the partnered device.

One of them, EDHOC for *Ephemeral Diffie-Hellman Over COSE*, is in development. EDHOC is expected to become an important component in securing constrained networks and devices for the Internet of Things. The keys shared by the devices after the execution of the EDHOC protocol can be used to protect future applications. Steps of development include formal and computational security analysis. The second one was part of this thesis, leading to the **Computational Security Analysis of the EDHOC protocol**.

2.2 Objectives

Privacy Preserving Machine Learning for Continuous Authentication. As said above, continuous authentication may rely on Machine Learning. In most cases, Machine Learning operations are made on the server side. However, in the use-case of continuous authentication, it is more

relevant to make the computation on the client-side in ordre to reduce the communications between the client and the server. As the computation is made on the client-side, the server has to securely send him the model. Hence, we may wonder which Machine Learning method is the most compliant with cryptography. The choice has been made to work on Decision Forest, as their theoretical conception is compliant with what can be doen in cryptography, which is not the case for others methods.

The objective was to add a cryptographic layer to this solution. More generally, with a Decision Forest model stored on a server, and the client data stored on a mobile phone, to securely evaluate the client data on the server model while keeping privacy for both of them. Indeed, the server should not learn more information about the client data than the outcome of the protocol, and the client should not learn information about the model, to avoid impersonation.

A Universally Composable post-quantum Oblivious Transfer based on SPHF. As said previously, using SPHF based on post-quantum primitives introduces a difference by opposition to classical cryptography based ones. This difference is that a gap appears, where one can not claim correctness nor smoothness opening the doors to potential attacks. The objective of this project is to tame this gap. By opposition to the solution provided by [BBB+22a], where they use zero-knowledge proofs, we will focus on required properties on the SPHF construction to ensure the gap does not open the doors for attacks. In addition, even if we ensured no attack is possible on the SPHF, this is not enough to provide Universal Composability when used for building an Oblivious Transfer protocol. Hence, we also have to prove our Oblivious Transfer construction is secure in the Universal Composability framework.

Computational Security Analysis of EDHOC. The Lightweighted Authenticated Key Exchange (LAKE) working group of the Internet Engineering Task Force (IETF), came to us with the project to study EDHOC. The EDHOC authenticated key exchange offers two ways to provide authentication. A peer can use a signature or a static Diffie-Hellman key previously owned. We focus on the case both peers use a static Diffie-Hellman key, together with aggressive parameters. Hence, our objectives were to prove, in this context, the security of the following requirements:

- Key Privacy or Implicit Authentication: At most both participants know the final session key
- Mutual Authentication or Explicit Authentication: Exactly both participants can compute the final session key
- Identity Protection: At most both participants know the identity of the involved peers.

2.3 Contributions

Privacy Preserving Machine Learning for Continuous Authentication. In our context, the server evaluates a model $\mathcal{M}$ on client data x, and accept or reject according to the outcome of the evaluation. We designed two protocols to securely evaluate a binary decision forest. In both those constructions the server is considered as Honest-but-Curious, but in the first construction the client is Honest-but-Curious while in the second one, we consider a malicious client. The use-cases we take into consideration requiring few interactions between the server and the client, we split the protocol in two phases. A first phase, offline, consists in the server sending its secure model to the server, and the second phase, online, consists in the evaluations.

In the setting of an Honest-but-Curious client, the evaluation simply consists in one flow sent from the client to the server. Even if no previous work were done considering client-side computation, this result outperforms works considering server-side computation and is optimal.

However, the extension of this protocol to the malicious setting implies two additional round-trip flows. Some previous constructions need less flows than ours, but more bandwidth, or inversely. Thus, our protocol can be seen as a trade-off between bandwidth and flows.

Our construction leaks the amount of successful paths to the server. However, rather than a real leak, this may help the server to take decisions according to this amount of successful paths, interpreted as a confidence index.

Another contribution is that, while our construction secure against malicious client uses Garbled Circuits, we do not need to use any additional techniques compared to their use in an honest-but-curious setting. Thus, we do not need to use solutions such as *Cut & Choose*. Even if *Cut & Choose* solution is well studied and optimized [MF06, LP07, Lin13, LR14, AMPR14, WMK17], it still is an expansive solution requiring ℓ *Garbled Circuits* to reach a statistic security of $2^{-\ell}$ bits. In a survey, Dupin et al. [DPB18] showed that a malicious generator can corrupt a Garbled Circuits only by adding NOT-gates or by failure attacks, but our constructions naturally provide security against these attacks in our context. Indeed, in case of failure attack, the server will reject and thus learn nothing about the threshold. If the circuit is modified, we introduced the notion of polarity, that makes the expected outcome unpredictable. Hence, a malicious client has no clue how to modify the circuit to successfully cheat and the probability to correctly modify the Garbled Circuits is the probability to correctly guess the polarity of each path.

Even if our solution has to use discrete values instead of floats – as cryptography dealing with float values is too expansive –, implying a loss of precision, we reach an accuracy of around 90% on the Spambase database as well as on a continuous authentication database.

This work has led to the following publication:

★ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Secure decision forest evaluation. In *The 16th International Conference on Availability, Reliability and Security*, pages 1–12, 2021

and the (updated) working paper is available on HAL:

► Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Secure Decision Forest Evaluation. hal-03321368, 2022. https://hal.archives-ouvertes.fr/hal-03321368

A Universally Composable post-quantum Oblivious Transfer based on SPHF. Rather than annihilate the Gray Zone specific to post-quantum-based SPHF instantiations, we introduced a new property, the Decomposition Intractability. When this property is verified, an adversary can not generate efficiently two values either in the language of the SPHF, either in that Gray Zone, that are complementary to each other, given a random element. We denote an SPHF that verifies this property a *Smooth Projective Hash Function with Gray Zone* (SPHFwGZ).

Then, we made a general construction of Oblivious Transfer protocol based on any SPHFwGZ, that has been proven secure in the Universal Composability framework. This setup has been instantiated with two such encryption schemes. The first one is the ElGamal encryption scheme. Even if the SPHFwGZ based on ElGamal does not in fact contains a gap, using it as an instantiation shows that our construction works from any SPHF as Decomposition Intractability is immediate. The second instantiation we give is based on lattices and a *Learning with Errors* (LWE) based encryption scheme. LWE encryption scheme has some decryption failures – justifying the existence of the Gray Zone – but we showed the Decomposition Intractability holds anyway. This instantiation allows to show the genericity of our construction, from classical to post-quantum cryptography.

This work has led to the following publication:

★ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Post-Quantum and UC-secure Oblivious Transfer from SPHF with Grey Zone. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022

and the (updated) working paper is available on HAL:

▶ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Post-Quantum Oblivious Transfer from Smooth Projective Hash Functions with grey[1]Zone. hal-03772089, 2022. https://hal.archives-ouvertes.fr/hal-03772089

Computational Security Analysis of EDHOC. We did the security proof of the EDHOC protocol using the authentication method based on static Diffie-Hellman keys. The proof has been made considering aggressive parameters, i.e. a MAC digest length of 64 bits.

We showed, under several classical assumptions, that the EDHOC authenticated key exchange provides a nearly 128 bits security for the Key Privacy and the Identity Protection. Nevertheless, Mutual Authentication security only reaches a 64 bits security for both the Initiator and the Responder. Even if this is assumed for the Authentication Security of the Responder, as the documentation suggests him to send a fourth authenticated message to increase its security, this is not the case for the Initiator Security. However, we emphasized that with some modifications, without any extra-cost compared to the initially analyzed version, the EDHOC protocol can provide a 128-bit Mutual Authentication security. Another of our modification makes the security proof tighter by making it independent of the number of parallels running sessions.

This work has led to the following (best paper awarded) publication:

★ Baptiste Cottier and David Pointcheval. Security Analysis of Improved EDHOC Protocol. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022

and the (updated) working paper is available on HAL:

▶ Baptiste Cottier and David Pointcheval. Security analysis of the EDHOC protocol. hal-03772082v2, 2022. https://hal.archives-ouvertes.fr/hal-03772082

2.4 Organization

This chapter introduced the reader to the context of the thesis. Chapter 2 contains all the notions required to a full understanding of the content of chapter 3, chapter 4 and chapter 5. Chapter 3 introduces our work made to add a cryptographic layer to an already existing continuous authentication protocol. It contains our solutions and their performance. Next, in the chapter 4, we detailed the construction of our Smooth Projective Hash Functions with Gray Zone, and gave some instantiations of it, on both classical and post-quantum cryptography. Chapter 5 is our security proof of the EDHOC protocol and the modifications we made to increase its security without adding communication nor computation cost. The unsuccessful first project is nevertheless detailed in Appendix as it is part of the thesis. We then provide a conclusion, containing some research prospects and open questions.

Chapter 2

Preliminaries

In this chapter, we introduce most of the necessary notions and primitives used in the thesis.

2.1 Complexity definitions

We first define the notion of negligeability. Roughly speaking, negligeability represents the idea that a function is close enough to zero to regard it as it is. In cryptography, negligeability is often associated to probabilities. Hence, a negligible probability is a probability smaller than the inverse of any polynomial.

Definition 2.1 - Negligeability

A function $\varepsilon : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive integer c, there exists an integer N_c such that for all $n > N_c$, $\left|\varepsilon(x) < \frac{1}{x^c}\right|$

Similarly, we say an event F happens *with overwhelming probability* if the probability of $\overline{F}$ (opposite event of F) is negligible. The notion of negligeability allows us to define the notion of computational indistinguishability.

Definition 2.2 - Computational Indistinguishability

Let $X = \{Xn\}_{n\in\mathbb{N}}$ and $Y = \{Yn\}_{n\in\mathbb{N}}$ be two distribution ensembles. X and Y are computationally indistinguishable if

$$|\Pr[\mathcal{A}(X_n) = 1] - \Pr[\mathcal{A}(Y_n) = 1]| < \varepsilon(n)$$

for all probabilistic polynomial time adversary $\mathcal{A}$, where ε is a negligible function.

In other words, computational indistinguishability reflects the idea that it would require an unreasonable amount of time and computation resources to distinguish those distributions. The allowed computation resources given to an adversary to cryptanalyse a cryptosystem are represented by a security parameter denoted 1^κ, used to set up protocols.

2.2 Symmetric cryptography

The oldest cryptosystems require both parties to use the same key to encrypt and decrypt a message. This symmetric process of such cryptosystems gave its name to the global family of cryptographic primitives where a unique key is used by both party implied in the primitive: the *Symmetric Cryptography*. We now define the most popular of those primitives, beginning with the symmetric encryption.

2.2.1 Symmetric encryption

Symmetric Encryption, also called Secret-Key Encryption, aims to provide confidentiality. Alice uses a secret key to encrypt her message, and then Bob uses the same key to decrypt. Some symmetric encryption schemes take as input an *Initialization Vector* that provides randomness and avoids deterministic encryption.

Definition 2.3 - Symmetric Encryption

A symmetric encryption scheme Π with messages in $\mathcal{M}$ and keys in $\mathcal{K}$ is a tuple $\Pi = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ of probabilistic polynomial time algorithms:

- $\mathsf{Setup}(1^\kappa)$ generates global parameters param based on the security parameter 1^κ. Returns param.
- $\mathsf{KGen}(\mathsf{param})$ generates a symmetric key $\mathsf{sk} \in \mathcal{K}$. Returns sk.
- $\mathsf{Enc}(\mathsf{sk}, \mathsf{IV}, m)$ computes an encryption c of $m \in \mathcal{M}$ using the key sk and a possibly empty Initialization Vector IV. Returns c.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{IV}, c)$ computes a decryption m of c using the key sk and a possibly empty Initialization Vector IV. Returns m.

We expect the decryption correctness property: $\mathsf{Dec}(\mathsf{sk}, \mathsf{IV}, \mathsf{Enc}(\mathsf{sk}, \mathsf{IV}, m)) = m$.

As said previously, the main goal of symmetric encryption is to provide confidentiality. Concretely, the probability for an adversary to extract information from a ciphertext he intercepts should be negligible. The confidentiality security level is represented by the notion of *Indistinguishability under Chosen Plaintext Attacks*.

Indistinguishability Under Chosen Plaintext Attack. The indistinguishability game varies depending on the need or not of an Initialization Vector. We first describe the game when no IV is required, then we describe the game when IV are required.

Without IV. Let $\mathcal{A}$ be an adversary with access to an encryption oracle encrypting any message of his choice. After querying a polynomially bounded number of times the oracle, the adversary chooses and sends two messages m_0 and m_1, of its choice, to the challenger who samples $b \leftarrow \{0, 1\}$ and encrypts $c \leftarrow \mathsf{Enc}(\mathsf{sk}, m_b)$. Afterward, c is sent to the adversary who, after new polynomially bounded queries to the oracle, makes a guess $b' \in \{0, 1\}$. The adversary wins if $b' = b$ and if the messages m_0 or m_1 are not in the list $\mathfrak{Q}$ of messages queried to the encryption oracle. If the adversary can not guess b with more than a negligible probability, Π provides indistinguishability under chosen plaintext attacks. In other words, Π is IND-CPA secure.

Definition 2.4 - Ciphertext indistinguishability - Without IV

Given an encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, the advantage $\mathsf{Adv}_\Pi^{\text{IND-CPA}}(\mathcal{A})$ of the adversary $\mathcal{A}$ making polynomially-many queries to an encryption oracle in violating the IND-CPA security of Π is given by

$$\left|\Pr\left[\mathbf{Exp}_\Pi^{\text{IND-CPA}-1}(\mathcal{A}) = 1\right] - \Pr\left[\mathbf{Exp}_\Pi^{\text{IND-CPA}-0}(\mathcal{A}) = 1\right]\right|$$

and the advantage function as:

$$\mathsf{Adv}_\Pi^{\text{IND-CPA}}(t) = \max_{\mathcal{A}} \left(\mathsf{Adv}_\Pi^{\text{IND-CPA}}(\mathcal{A})\right)$$

over all $\mathcal{A}$ with time-complexity at most t.

$\mathbf{Exp}_\Pi^{\text{IND-CPA}-b}(\mathcal{A})$

1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$
2. $\mathsf{sk} \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{param})$
3. $(m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk}, \cdot)}(\ )$
4. $c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{sk}, m_b)$
5. $b', \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk}, \cdot)}(c, m_0, m_1)$
6. **if** $m_0 \in \mathfrak{Q}$ **or** $m_1 \in \mathfrak{Q}$:
7. **return** 0
8. **else return** $b = b'$

Hence, Π provides IND-CPA security if $\mathsf{Adv}_\Pi^{\text{IND-CPA}}(t)$ is negligible.

On perfect secrecy. Ideally, one may expect $\mathsf{Adv}^{\text{IND-CPA}}_{\Pi}(t)$ to be null. In the definition given above of symmetric encryption, the key is generated before encryption, independently of the message m. Indeed, the key is generated according to the security parameter 1^κ, which defines the key length.

Therefore, the size of the space of keys $\mathcal{K}$ is smaller or equal to the size of the space of messages $\mathcal{M}$.

This leads the adversary advantage to be non-null.

Nonetheless, as we use it later, we introduce now the *One-Time Pad* encryption, deviating from the above definition as the key is generated according to the length of the message to be encrypted.

Definition 2.5 - One-Time Pad (OTP)

Given a message $m \in \{0,1\}^k$, the One-Time Pad encryption scheme consists in a tuple $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{KGen}(k)$: Given the length k of the message to encrypt, samples $\mathsf{sk} \leftarrow\!\!\$\ \{0,1\}^k$. Return sk;
- $\mathsf{Enc}(m, \mathsf{sk})$: Computes $c \leftarrow m \oplus \mathsf{sk}$. Returns c ;
- $\mathsf{Dec}(c, \mathsf{sk})$: Computes $m \leftarrow c \oplus \mathsf{sk}$. Returns m.

For a random key sk, a ciphertext c can not leak any information about the plaintext as it could be an encryption of any $m \in \{0,1\}^k$. One may also interpret the One-Time Pad encryption as a bitwise randomization. Hence, the One-Time Pad encryption provides perfect secrecy. However, this perfect secrecy is only valid if the key sk is used once. Indeed, consider two ciphertexts $c_1 \leftarrow \mathsf{Enc}(\mathsf{sk}, m_1) = m_1 \oplus \mathsf{sk}$ and $c_2 \leftarrow \mathsf{Enc}(\mathsf{sk}, m_2) \oplus \mathsf{sk}$. Computing $c_1 \oplus c_2 = (m_1 \oplus \mathsf{sk}) \oplus (m_1 \oplus \mathsf{sk}) = m_1 \oplus m_2$ reveals which bits of m_1 and m_2 are equals, and if the message is long enough, a frequency analysis can reveal the messages m_1 and m_2.

With IV. When the encryption scheme uses an Initialization Vector, the adversary queries an encryption oracle. This oracle, when queried with a pair (m, IV) returns an encryption of m with Initialisation Vector IV, with the property that an IV can be queried to the oracle only once for a given message. Moreover, when sending a pair of messages (m_0, m_1) to the challenger, the adversary also sends an Initialization Vector IV and the challenger computes $\mathsf{Enc}(\mathsf{sk}, \mathsf{IV}, m_b)$. Similarly to the case without IV, the adversary wins if $b' = b$ and if IV is not in the list $\mathfrak{Q}$ of the Initialization Vectors queried to the encryption oracle. Note that messages queried do not need to be stored in $\mathfrak{Q}$.

We then define the appropriate IND-CCA definition:

Definition 2.6 - Ciphertext indistinguishability - With IV

Given an encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ using Initialization Vectors, the advantage $\mathsf{Adv}^{\text{IND-CPA}}_{\Pi}(\mathcal{A})$ of the adversary $\mathcal{A}$ making a polynomial number of queries to an encryption oracle in violating the IND-CPA security of Π is given by

$$|\Pr[\mathbf{Exp}^{\text{IND-CPA}-1}_{\Pi}(\mathcal{A}) = 1] - \Pr[\mathbf{Exp}^{\text{IND-CPA}-0}_{\Pi}(\mathcal{A}) = 1]|$$

and the advantage function $\mathsf{Adv}^{\text{IND-CPA}}_{\Pi}(t)$ as:

$$max_{\mathcal{A}}(\mathsf{Adv}^{\text{IND-CPA}}_{\Pi}(\mathcal{A}))$$

over all $\mathcal{A}$ with time-complexity at most t.

$\mathbf{Exp}^{\text{IND-CPA}-b}_{\Pi}(\mathcal{A})$

1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^{\kappa})$
2. $\mathsf{sk} \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{param})$
3. $(m_0, m_1, \mathsf{IV}) \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot)}(\ )$
4. $c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{sk}, \mathsf{IV}, m_b)$
5. $b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot,\cdot)}(c, m_0, m_1)$
6. **if** $\mathsf{IV} \in \mathfrak{Q}$:
7. **return** 0
8. **else return** $b = b'$

Hence, Π provides IND-CPA security if $\mathsf{Adv}^{\text{IND-CPA}}_{\Pi}(t)$ is negligible.

In addition to confidentiality, cryptography may also be used to provide integrity and authentication. If Alice wants to give Bob the possibility to ensure the message he receives has not been altered by an eavesdropper, he may use Message Authentication Codes (MAC).

2.2.2 Message authentication code (MAC)

When using Message Authentication Codes, Alice generates a value named *tag*, using a key k she shares with Bob. As the key is secret, an eavesdropper is supposed not to be able to compute a MAC. Hence, when Bob verifies the validity of the tag, he ensures the message has not been altered (*integrity*) and was indeed sent by Alice (*authentication*). We denote by MAC the function computing the Message Authentication Code, and by tag the output of MAC.

Definition 2.7 - Message Authentication Code

A message authentication code MAC with messages in $\mathcal{M}$ and keys in $\mathcal{K}$ is a tuple $(\mathsf{KGen}, \mathsf{MAC}, \mathsf{Vf})$ of probabilistic polynomial-time algorithms:

- $\mathsf{KGen}(1^n)$ samples a key $k \in \mathcal{K}$. Returns k.
- $\mathsf{MAC}(k, m)$ given a message $m \in \mathcal{M}$ and sk, computes the tag τ associated to m. Returns τ.
- $\mathsf{Vf}(k, m, \tau)$ verifies if $\tau = \mathsf{MAC}(k, m)$. Returns a bit b set to 1 if the verification succeeds and 0 otherwise.

We expect the verification correctness property : $\mathsf{Vf}(k, m, \mathsf{MAC}(k, m)) = 1$. As MAC are used to provide integrity, we define a proper security definition of what integrity is. Someone else than Alice should only have a negligible probability to pass the verification test, i.e. forge a valid tag. Hence, we introduce the notion of Existential Unforgeability under Chosen Messages Attack (EUF-CMA).

Existential Unforgeability under Chosen Messages Attack (EUF-CMA**).** Let $k \leftarrow \mathsf{KGen}(1^{\kappa})$ and $\mathcal{A}$ be an adversary with access to an oracle knowing the key k and able to compute the tag of any message of its choice. $\mathcal{A}$ wins if after a polynomially bounded number of queries, he can

generate a tag τ for a message m not in the list $\mathfrak{Q}$ of previously queried messages to the oracle, such that $\mathsf{Vf}(k, m, \tau) = 1$.

Definition 2.8 - Existential Unforgeability under Chosen Messages Attack (EUF-CMA)

Given a MAC scheme $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, the advantage of the adversary $\mathcal{A}$ making polynomially-many queries to a MAC oracle in violating the EUF-CMA security of MAC is given by:

$$\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{MAC}}(\mathcal{A}) = 1\right]$$

and the advantage function $\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{MAC}}(t)$ as

$$\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{MAC}}(t) = \max_{\mathcal{A}} \left(\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{MAC}}(\mathcal{A})\right)$$

over all $\mathcal{A}$ with time-complexity at most t.

$\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{MAC}}(\mathcal{A}, 1^\kappa)$

1. $k \leftarrow\!\$\ \mathsf{KGen}(1^\kappa)$
2. $(m, \tau) \leftarrow \mathcal{A}^{\mathsf{MAC}(k,\cdot)}(\ )$
3. **if** $m \in \mathfrak{Q}$: **return** 0
4. **else return** $\mathsf{Vf}(k, m, \tau)$

Hence, MAC provides EUF-CMA security if $\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{MAC}}(t)$ is negligible.

It is possible to combine both Symmetric Encryption and MAC under a single primitive, called *Authenticated Encryption*.

2.2.3 Authenticated encryption (with associated data)

Authenticated Encryption combines the confidentiality of a Symmetric Encryption and the authentication and integrity of a Message Authentication Code. In authenticated encryption, the encryption process generates the ciphertext and the associated tag, while the decryption returns either the plaintext, or an error if the authentication fails.

There exists three approaches to combine the MAC and the encryption process: Encrypt-**then**-MAC, Encrypt-**and**-MAC, MAC-**then**-Encrypt. In this thesis, we will only meet the Mac-then-Encrypt approach. Hence, we use Authenticated Encryption to denote MtE-Authenticated Encryption When receiving an authenticated encryption, Bob first decrypts the ciphertext and verifies the tag validity. If the verification fails, Bob aborts. A variant of Authenticated Encryption, *Authenticated Encryption with Associated Data* (AEAD) allows Alice to send authenticated plain data to Bob. When the additional data contains some information about the context of the protocol, this prevents a reuse of the ciphertext by an attacker.

Definition 2.9 - Authenticated encryption

An authenticated encryption (with possibly empty Associated Data d) scheme (with MtE approach) Π with messages in $\mathcal{M}$ and keys in $\mathcal{K}$ is a tuple $\Pi = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ of probabilistic polynomial time algorithms:

- $\mathsf{Setup}(1^\kappa)$ generates global cryptosystem parameter param based on the security parameter. Returns param
- $\mathsf{KGen}(\mathsf{param})$ generates a symmetric encryption key sk. Returns sk.
- $\mathsf{Enc}(\mathsf{sk}, \mathsf{IV}, m, d)$ computes a tag $t \leftarrow \mathsf{MAC}(\mathsf{sk}, m\|d)$ and next, an encryption c of $m\|t$ using the key sk and potentially an Initialization Vector IV. Returns $C = (c\|d)$.
- $\mathsf{Dec}(\mathsf{sk}, IV, C)$ parses C as $(c\|d)$ and obtains $m\|t$ by decrypting c using sk and IV. Returns $m\|d$ if $\mathsf{Vf}(\mathsf{sk}, m\|d, t) \neq 1$, and aborts otherwise.

We expect the decryption correctness property : $\mathsf{Dec}(\mathsf{sk}, \mathsf{IV}, \mathsf{Enc}(\mathsf{sk}, \mathsf{IV}, m, d)) = m\|d$. The

correctness of the MAC is implicit as the decryption process terminates only if the MAC verification returns 1. Formally, the verification correctness property ensures $\mathsf{Dec}(\mathsf{sk}, \mathsf{IV}, \mathsf{Enc}(\mathsf{sk}, \mathsf{IV}, m, d)) \neq \perp$.

Another important and widely used symmetric primitive providing data integrity is hash functions.

2.2.4 Hash functions

A hash function H is a one-way compression function. Namely, given an input m of *arbitrary* length, it must be easy to compute $h \leftarrow \mathsf{H}(m)$ of *fixed* length, while given h, it must be hard to compute m.

Definition 2.10 - Hash Functions

A hash function with output length ℓ_H is a polynomial-time algorithm H such that:

- $\mathsf{H}(m)$: On input with an arbitrary length string $m \in \{0,1\}^*$, output a digest $h \in \{0,1\}^{\ell_H}$

With hash functions, the verification consists in recalculating the same hash on the same string. A use of hash functions is file integrity checking, for instance, before digitally signing them. By computing a hash on the file to be signed, it ensures the signature can not be used for other files. Indeed, an adversary cannot computationally generate a file with the same hash value, and can not generate two different files with the same hash value. More precisely, from a hash function we expect the following properties:

- *Pre-image resistance:* Given a hash value h, an adversary should not be able to find m such that $\mathsf{H}(m) = h$ with overwhelming probability.
- *Second pre-image resistance:* Given a message m_1, an adversary should not be able to find m_2 such that $\mathsf{H}(m_1) = \mathsf{H}(m_2)$ with overwhelming probability.
- *Collision resistance:* An adversary should not be able to find two messages m_1, m_2 such that $\mathsf{H}(m_1) = \mathsf{H}(m_2)$. Due to the birthday attack, this implies a digest size at least twice as long as the required bit-security with overwhelming probability.

In cryptography, we expect an extra pseuo-random property. Indeed, if we set $\tilde{m}$ as m with a single random bit flip, we expect $(m, H(\tilde{m}))$ to be indistinguishable from $(m, h \leftarrow\!\!\$\ \{0,1\}^{\ell_{\text{hash}}})$. In the rest of the document, hash functions implicitly are *cryptographic* hash functions.

2.2.5 Random oracle model

When hash functions are modelized using a random oracle returning perfectly random digest for security proofs, we say the proof is made in the *Random Oracle Model* [BR93].

Definition 2.11 - Random Oracle Model

If a protocol is proven in the Random Oracle Model, calls to a hash function H are replaced with calls to an oracle $\mathcal{O}_\mathsf{H}$. When calling H on m, the oracle acts as follows

- If m has never been queried before, $\mathcal{O}_\mathsf{H}$ generates $s_m \leftarrow\!\!\$\ \{0,1\}^{\ell_\mathsf{H}}$, returns s_m and stores $[m, s_m]$
- If m has ever been queries, $\mathcal{O}_\mathsf{H}$ returns s_m

2.2.6 Pseudorandom functions

Pseudorandom functions (PRF) take as input a random key and are used to generate output computationally indistinguishable from truly random output.

Definition 2.12 - Pseudorandom Function (PRF)

Let F be a function with input a random key $k \in \{0,1\}^s$ and some input $x \in \{0,1\}^m$. The advantage of the adversary $\mathcal{A}$ making polynomially-many queries to the oracle in breaking the randomness of F is given by:

$$\mathrm{Adv}_F^{PRF}(\mathcal{A}) = |\Pr[\mathbf{Exp}_F^{PRF-1}(\mathcal{A}) = 1] - \Pr[\mathbf{Exp}_F^{PRF-0}(\mathcal{A}) = 1]|$$

and the advantage function $\mathrm{Adv}_F^{PRF}(t)$ as

$$\mathrm{Adv}_F^{PRF}(t) = \max_{\mathcal{A}}(\mathrm{Adv}_F^{PRF}(\mathcal{A}))$$

over all $\mathcal{A}$ with time-complexity at most t.

Hence, F is a pseudorandom function if $\mathrm{Adv}_F^{PRF}(t)$ is negligible.

$\mathbf{Exp}_F^{PRF-b}(\mathcal{A}, 1^\kappa)$

1. $k \leftarrow\$ \{0,1\}^\kappa$
2. $b' \leftarrow \mathcal{A}^{F_b(k,\cdot)}(\ )$
3. **return** $b = b'$

Oracles details

$F_1(k,x) = F(k,x)$

$F_2(k,x) \leftarrow\$ \{0,1\}^m$

Pseudorandom functions are a good option to build MACs. Under the properties of a PRF, we reach the EUF-CMA property of a MAC. Moreover, when the PRF is implemented with a random oracle, the EUF-CMA property is statistical.

2.3 Asymmetric cryptography

The main drawback of the symmetric encryption is the fact the key has to be previously shared between Alice and Bob to securely communicate. To overcome this problem, two ideas are introduced by Withfield Diffie and Martin Hellman in their seminal article [DH76]. The first one is to securely share a key over an untrusted channel, and the second one is to design cryptosystems where no shared key is required. In [DH76], only the first idea, latter called Diffie-Hellman key exchange, is instantiated, while the second idea waited one more year to be concretized by Rivest, Shamir and Adleman [RSA78], leading to the RSA cryptosystem. Both those solutions rely on mathematical problems and security assumptions we define hereunder.

2.3.1 Discrete Logarithm problem and Diffie-Hellman assumptions

We begin by introducing the Discrete Logarithm problem.

Definition 2.13 - Discrete Logarithm (DL)

Let $\mathbb{G}$ be a cyclic group of order p with generator g. Given $h \leftarrow\$ \mathbb{G}$, the discrete logarithm problem for the group $\mathbb{G}$ consists in finding $x \in \mathbb{Z}_p$ such that $g^x = h$

We now introduce assumptions related to the Discrete Logarithm Problem.

Definition 2.14 - Computational Diffie-Hellman (CDH)

Let $\mathbb{G}$ be a cyclic group of order p. The Computational Diffie-Hellman assumption states that given a tuple (g, g^u, g^v) where u and v were drawn at random from $\mathbb{Z}_p$, it is hard to compute g^{uv}. More precisely the advantage of the adversary $\mathcal{A}$ in violating the Computational Diffie-Hellman assumption in $\mathbb{G}$, is given by:

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A}) = 1\right]$$

and the advantage function $\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$ as

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t) = \max_{\mathcal{A}}\left(\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A})\right)$$

over all $\mathcal{A}$ with time-complexity at most t.

Hence, the CDH assumption is hard in $\mathbb{G}$ if $\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$ is negligible.

$\mathbf{Exp}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A})$

1. $u, v \leftarrow\!\$\ \mathbb{Z}_p$
2. $U \leftarrow g^u, V \leftarrow g^v$
3. $Z \leftarrow \mathcal{A}(U, V)$
4. **return** $Z = g^{uv}$

From now, we denote by DH-tuple a tuple $(g, g^u g^v, g^{u \cdot v})$.

The second assumption considers the ability to decide if a given tuple is a DH-tuple. We then introduce the Decisional Diffie-Hellman.

Definition 2.15 - Decisional Diffie-Hellman (DDH)

Let $\mathbb{G}$ be a cyclic group of order p. The Decisional Diffie-Hellman assumption states that given g^u and g^v, where u, v were drawn at random from $\mathbb{Z}_p$, the value g^{uv} is indistinguishable from $g^r, r \leftarrow\!\$\ \mathbb{Z}_p$. More precisely the advantage of the adversary $\mathcal{A}$ in violating the Decisional Diffie-Hellman assumption in $\mathbb{G}$, is given by:

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A}) = \left|\Pr\left[\mathbf{Exp}^{\mathsf{DDH}-1}_{\mathbb{G}}(\mathcal{A}) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{DDH}-0}_{\mathbb{G}}(\mathcal{A}) = 1\right]\right|$$

and the advantage function $\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(t)$ as

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(t) = \max_{\mathcal{A}}\left(\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A})\right)$$

over all $\mathcal{A}$ with time-complexity at most t.

Hence, the DDH assumption is hard in $\mathbb{G}$ if $\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(t)$ is negligible.

$\mathbf{Exp}^{\mathsf{DDH}-b}_{\mathbb{G}}(\mathcal{A})$

1. $u, v, r \leftarrow\!\$\ \mathbb{Z}_p$
2. $U \leftarrow g^u, V \leftarrow g^v$
3. $W_0 \leftarrow g^{uv}, W_1 \leftarrow g^r$
4. $b' \leftarrow \mathcal{A}(U, V, W_b)$
5. **return** $b = b'$

A last stronger assumption lets the CDH-adversary make polynomially bounded queries to an oracle that can decide if a given tuple (other than the one instantiated in the problem) (g, g^a, g^b, g^c) is a DH-tuple or not.

Definition 2.16 - Gap Diffie-Hellman (GDH)

Let $\mathbb{G}$ be a cyclic group. The Gap Diffie-Hellman assumption states that given g^u and g^v, where u, v were drawn at random from $\mathbb{Z}_p$, the value g^{uv} is hard to compute, even when the adversary has access to a DDH oracle returning 1 if a tuple (g, g^a, g^b, g^c) is a Diffie-Hellman tuple, and 0 otherwise. More precisely the advantage of the adversary $\mathcal{A}$ in violating the Computational Diffie-Hellman assumption in $\mathbb{G}$ making polynomially-many queries to the DDH oracle is given by:

$$\mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}^{\mathrm{GDH}}_{\mathbb{G}}(\mathcal{A}) = 1\right]$$

and the advantage function $\mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t)$ as

$$\mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t) = \max_{\mathcal{A}} \left(\mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(\mathcal{A})\right)$$

over all $\mathcal{A}$ with time-complexity at most t.

Hence, the GDH assumption is hard in $\mathbb{G}$ if $\mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t)$ is negligible.

$\mathbf{Exp}^{\mathrm{GDH}}_{\mathbb{G}}(\mathcal{A})$

1. $u, v \leftarrow\!\!\$\ \mathbb{Z}_p$
2. $U \leftarrow g^u, V \leftarrow g^v$
3. $Z \leftarrow \mathcal{A}^{\mathrm{DDH}(\cdot,\cdot,\cdot,\cdot)}(U, V)$
4. **return** $Z = g^{uv}$

2.3.2 Factorization problems

Another widely use problem in public key encryption is the Factorization Problem.

Definition 2.17 - Factorisation Problem

Let N be a composite number. The factorization problem consists in finding one factor of N.

In most of the cases this problem is easy. However, when N is known to be a product of two large prime numbers (N is called semiprime), the factorization problem becomes hard. Another problem relying on the factorization problem is the RSA problem.

Definition 2.18 - RSA problem

Let $N = pq$ be a product of two large numbers and $2 < e < N$ such that e and $\Phi(N) = (p-1)(q-1)$ are coprime. Given $c \in \mathbb{Z}_n$, the RSA problem consists in finding m such that $m^e = c \mod N$

2.3.3 Public-key encryption

A public-key cryptosystem, or asymmetric cryptosystem, generates a secret key sk, and a public key pk associated to this secret key. The public key is used for encryption, and, as it is public, should not reveal any information about sk, the secret key used for decryption.

Definition 2.19 - Public-Key Encryption

A public-key encryption scheme Γ with messages in $\mathcal{M}$ and keys in $\mathcal{K}$ is a tuple of probabilistic polynomial-time algorithms $\Gamma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- $\mathsf{Setup}(1^\kappa)$ generates global cryptosystem parameter param based on the security parameter. Returns param.
- KGen(param) generates a secret key sk and its associated public key pk. Returns (pk,sk).
- Enc(pk, m) computes an encryption c of $m \in \mathcal{M}$ using the key pk. Returns c.
- Dec(sk, c) computes a decryption m of c using the key sk. Returns m.

We expect the decryption correctness property : With $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{param})$, $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$. A single public key can therefore be used by as many people that wish to use it. This reduces the amount of required key in multiparty communication setting. Hence, consider a class of n students, where each student should be able to privately send a message to any other student. When using symmetric encryption, one key is needed by pair of students, i.e. $n(n+1)/2$ keys, while in the context of asymmetric encryption, only $2n$ keys, one public and one private per student, are required. However, in contrast to symmetric cryptography that is fast, asymmetric cryptography is slow due to more complex computations.

Analogously to the symmetric encryption, an asymmetric encryption scheme Γ may provide ciphertext indistinguishability under chosen plaintext attacks.

Definition 2.20 - Ciphertext indistinguishability under Chosen Plaintext Attacks - IND-CPA

Given an encryption scheme $\Gamma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, the advantage $\mathsf{Adv}_\Gamma^{\text{IND-CPA}}(\mathcal{A})$ of the adversary in violating the IND-CPA security of Γ is given by

$$|\Pr[\mathbf{Exp}_\Gamma^{\text{IND-CPA}-1}(\mathcal{A}) = 1] - \Pr[\mathbf{Exp}_\Gamma^{\text{IND-CPA}-0}(\mathcal{A}) = 1]|$$

and the advantage function $\mathsf{Adv}_\Gamma^{\text{IND-CPA}}(t)$ as

$$\mathsf{Adv}_\Gamma^{\text{IND-CPA}}(t) = \max_{\mathcal{A}}(\mathsf{Adv}_\Gamma^{\text{IND-CPA}}(\mathcal{A}))$$

over all $\mathcal{A}$ with time-complexity at most t.

Hence, Γ provides IND-CPA security if $\mathsf{Adv}_\Gamma^{\text{IND-CPA}}(t)$ is negligible.

$\mathbf{Exp}_\Gamma^{\text{IND-CPA}-b}(\mathcal{A})$

1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{param})$
3. $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$
4. $c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m_b)$
5. $b' \leftarrow \mathcal{A}(c, m_0, m_1)$
6. **return** $b = b'$

In addition to the IND-CPA property, a public-key encryption scheme may provide INDistinguishability under Chosen *Ciphertext* Attack.

Indistinguishability Under Chosen Ciphertexts Attack. Two notions of such indistinguishability are widely known : IND-CCA1 and IND-CCA2. To better understand the nuance between these two notions, consider the following indistinguishability games.

The challenger generates a pair of keys $(\mathsf{pk}, \mathsf{sk})$. Then the adversary has access to a decryption oracle, which given a ciphertext, returns its decryption. After a polynomially bounded number of queries, the adversary sends two messages m_0, m_1 of his choice to the challenger. The challenger samples $b \in \{0, 1\}$ and returns an encryption c of m_b to the adversary. In the IND-CCA2 game, the adversary is given a new polynomially bounded access to the decryption oracle, while in the IND-CCA1 game, no access to the decryption oracle is given before the adversary makes a guess b' corresponding to the message $m_{b'}$ he thinks c is an encryption of, based on pk, c, m_0 and m_1. If the adversary queries $\mathfrak{Q}$ with the challenge c, the game will return 0 and queries from the adversary are no longer answered. The adversary therefore wins if $b = b'$ and the challenge c is not in the list $\mathfrak{Q}$ of ciphertexts queried to the decryption oracle.

Definition 2.21 - Ciphertext indistinguishability under Chosen Ciphertexts Attacks - IND-CCA

Let $\Gamma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an asymmetrical encryption scheme. The advantage $\mathsf{Adv}_{\Gamma}^{\text{IND-CCA}\mu}(\mathcal{A})$ of the adversary $\mathcal{A}$ in violating the IND-CCAμ security of Γ, $\mu \in \{1, 2\}$, making polynomially-many global queries to the decryption oracle, is given by:

$$|\Pr[\mathbf{Exp}_{\Gamma}^{\text{IND-CCA}\mu-1}(\mathcal{A}) = 1] - \Pr[\mathbf{Exp}_{\Gamma}^{\text{IND-CCA}\mu-0}(\mathcal{A}) = 1]|$$

and the advantage function $\mathsf{Adv}_{\Gamma}^{\text{IND-CCA}\mu}(t)$ as

$$\mathsf{Adv}_{\Gamma}^{\text{IND-CCA}\mu}(t) = \max_{\mathcal{A}}(\mathsf{Adv}_{\Gamma}^{\text{IND-CCA}\mu}(\mathcal{A}))$$

over all $\mathcal{A}$ with time-complexity at most t.

Hence, Γ provides IND-CCAμ security if $\mathsf{Adv}_{\Gamma}^{\text{IND-CCA}\mu}(t)$ is negligible.

$\mathbf{Exp}_{\Gamma}^{\text{IND-CCA}\mu-b}(\mathcal{A}, 1^{\kappa})$

1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^{\kappa})$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{param})$
3. $(m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{pk})$
4. $c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m_b)$
5. $b' \leftarrow \mathcal{A}^{\mathsf{Dec}_{\mu}(\mathsf{sk},\cdot)}(c, m_0, m_1, \mathsf{pk})$
6. **if** $c \in \mathfrak{Q}$: **return** 0
7. **else return** $b = b'$

Oracles Description

$\mathsf{Dec}_1(\mathsf{sk}, \cdot) = \emptyset$

$\mathsf{Dec}_2(\mathsf{sk}, \cdot) = \mathsf{Dec}(\mathsf{sk}, \cdot)$

In addition to asymmetric encryption, the use of a different key for the encryption and the decryption led to a new primitive, *digital signatures*.

2.3.4 Digital signatures

The idea behind digital signatures is the same as the idea behind handwritten signatures: they aim to provide integrity and authentication. In addition, a signature scheme may provide non-repudiation. Signatures rely on the fact that only the owner of a secret key is supposed to know it. Hence, when signing, Alice uses her secret key to compute a tag, that Bob, and anyone can verify using Alice's public key.

Definition 2.22 - Digital Signatures

A Digital Signature scheme Σ with messages in $\mathcal{M}$ and keys in $\mathcal{K}$ is a tuple of probabilistic polynomial time $\Sigma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ as follows:

- $\mathsf{Setup}(1^{\kappa})$ generates the global parameters param from 1^{κ}. Returns param .
- $\mathsf{KGen}(\mathsf{param})$ generates a secret key sk and a public key pk. Returns $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{Sign}(\mathsf{sk}, m)$ generates a signature σ of m with the key sk. Returns σ.
- $\mathsf{Vf}(\mathsf{pk}, \sigma)$ verifies if the signature σ is a valid signature using the public key pk. Returns 1 if σ is a valid signature and 0 otherwise.

We expect the decryption correctness property : With $\mathsf{param} \leftarrow \mathsf{Setup}(1^{\kappa})$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{param})$, $\mathsf{Vf}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, m)) = 1$. Similarly to MAC, one should not be able to forge signatures:

Existential Unforgeability under Chosen Messages Attack. Let $\mathsf{sk} \leftarrow \mathsf{KGen}(1^{\kappa})$ and $\mathcal{A}$ be an adversary with access to an oracle knowing the key sk and able to compute the tag of any message. $\mathcal{A}$ wins if after a polynomially bounded number of queries, he can generate a signature σ for a message m not in the list $\mathfrak{Q}$ of previously queried messages to the oracle, such that $\mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1$.

Definition 2.23 - Existential Unforgeability under Chosen Messages Attack (EUF-CMA)

Given a signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$, the advantage of the adversary $\mathcal{A}$ making polynomially-many queries to a signature oracle in violating the EUF-CMA security of Σ is given by:

$$\mathrm{Adv}_{\Sigma}^{\text{EUF-CMA}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}_{\Sigma}^{\text{EUF-CMA}}(\mathcal{A}) = 1\right]$$

and the advantage function $\mathrm{Adv}_{\Sigma}^{\text{EUF-CMA}}(t)$ as

$$\mathrm{Adv}_{\Sigma}^{\text{EUF-CMA}}(t) = \max_{\mathcal{A}} \left(\mathrm{Adv}_{\Sigma}^{\text{EUF-CMA}}(\mathcal{A})\right)$$

over all $\mathcal{A}$ with time-complexity at most t.

$\mathbf{Exp}_{\Sigma}^{\text{EUF-CMA}}(\mathcal{A}, 1^{\kappa})$

1. $\mathsf{sk} \leftarrow\!\$\ \mathsf{KGen}(1^{\kappa})$
2. $(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\ )$
3. **if** $m \in \mathfrak{Q}$: **return** 0
4. **else return** $\mathsf{Vf}(\mathsf{pk}, \sigma)$

Hence, Σprovides EUF-CMA security if $\mathrm{Adv}_{\Sigma}^{\text{EUF-CMA}}(t)$ is negligible.

2.4 Homomorphic encryption

One of the most popular tool for Secure Multiparty Computation is Homomorphic Encryption. This primitive allows to evaluate a circuit on encrypted data. Depending on the circuit family, homomorphic encryption schemes are split into three variants:

- *Partially Homomorphic Encryption*: Unbounded computation of a single operation among additive and multiplicative homomorphism. Those are to most popular of the homomorphic encryption algorithms. An example of such an encryption scheme is denoted in figure 2.1.
- *Somewhat / Leverage Homomorphic Encryption*: Circuits that allow an unbounded computaton of additions and finite computation of multiplications.
- *Fully Homomorphic Encryption*: unbounded computation of both additive and multiplicative Homomorphism. Such systems have a long time thought to be unfeasible until the seminal thesis of Gentry [Gen09] in which the first FHE scheme is designed, based on lattices. However, such cryptosystems are not efficient.

More formally, one can define:

Definition 2.24 - Homomorphic Encryption

An Homomorphic Encryption Scheme Γ is a tuple of probabilistic polynomial time algorithms $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$:

- $\mathsf{Setup}(1^{\kappa})$ generates the global parameters param from 1^{κ}. Returns param .
- $\mathsf{KGen}(\mathsf{param})$ generates and outputs a secret key sk and its associated public key pk. Returns $(\mathsf{pk},\mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$ computes an encryption c of $m \in \mathcal{M}$ using the key pk. Returns c.
- $\mathsf{Eval}(\mathsf{pk}, \mathcal{C}, \{c_i\})$ computes c', an evalutation of the circuit $\mathcal{C}$ using ciphertexts $\{c_i\}$ and public key pk. Returns c'.
- $\mathsf{Dec}(\mathsf{sk}, c)$, computes a decryption m of c using the key sk. Returns m.

In table 2.1, we detail variants of homomorphic encryption, according to the maximum additive and maximum multiplicative depth of the circuit, together with examples of such cryptosystems:

In this thesis, we use several times a well-known homomorphic encryption scheme, namely the Elgamal encryption scheme: Let $\mathbb{G}$ be a cyclic group of prime order q, with generator g:

$+$	$\times$	Family of Γ	Examples
∞	0	Additive Homomorphic Encryption	[GM82, Pai99]
0	∞	Multiplicative Homomorphic Encryption	[RSA78, ElG85]
∞	$k \in \mathbb{Z}$	Somewhat Homomorphic Encryption	[BGN05, AGH10]
∞	∞	Fully Homomorphic Encryption	[Gen09]

Table 2.1: Summary of the Homomorphic Encryption families

KGen(1^κ)	Enc(pk, m)	Dec(sk, m)
1. $x \leftarrow\!\$\ \mathbb{Z}_q$	1. $r \leftarrow\!\$\ \mathbb{Z}_q$	1. $m = c_2 \cdot c_1^{-\mathsf{sk}}$
2. $y \leftarrow g^x$	2. $c_1 = g^r$	2. **return** m
3. $\mathsf{pk} \leftarrow (g, y, q)$	3. $c_2 = (m \cdot y^r)$	
4. $\mathsf{sk} \leftarrow x$	4. **return** $C = (c_1, c_2)$	
5. **return** (pk, sk)		

Figure 2.1: ElGamal Encryption Scheme

To understand the homomorphic properties of ElGamal, consider two encryptions $C = (c_1, c_2) = \mathsf{Enc}(\mathsf{pk}, m)$ and $C' = (c_1', c_2') = \mathsf{Enc}(\mathsf{pk}, m')$ and compute $C'' = (c_1 \times c_1', c_2 \times c_2')$. We have:

$$c_1'' = c_1 \times c_1' = g^r \times g^{r'} = g^{r+r'} \text{ and}$$

$$c_2'' = c_2 \times c_2' = ((m \cdot y^r)) \times ((m' \cdot y^{r'})) = (m \cdot m') \cdot y^{r+r'}$$

The pair $C'' = (c_1'', c_2'')$ is a ciphertext for $m \cdot m'$. Hence, we have

$$C'' = \mathsf{Enc}(\mathsf{pk}, m \cdot m') = \mathsf{Enc}(\mathsf{pk}, m) \cdot \mathsf{Enc}(pk, m')$$

ElGamal cryptosystem therefore is a multiplicative homomorphic scheme. We can convert the ElGamal encryption scheme to an additive encryption scheme by encrypting g^m rather than m. The obtained encryption scheme is denoted *lifted* ElGamal encryption scheme. However, using a lifted version of ElGamal implies discrete logarithm computations.

2.5 Post-Quantum cryptography

Due to the Shor's algorithm [Sho94], asymmetric cryptosystems based on the RSA and Discrete Logarithm problems are no longer secure when using quantum computers. Thus, the National Institute of Standards and Technologies launched a program to standardize post-quantum Public-Key Encryption / Key Encryption Mechanism and Signature. Starting from 59 PKE/KEM schemes and 23 signatures, four have been standardized [SAB+20, LDK+20, PFH+20, HBD+20] after Round 3 and three are still in the running to be standardized [AAB+22, ABB+22, ABC+22]. In reality, a fourth candidate [JAC+22] also reached round 4. However, the security of its signature relies on an Isogeny-Based assumption of which an efficient attack has been published [CD22], making this candidate obsolete. The three candidates still in competition are all based on Code-Based Cryptography, while already standardized candidates use Hash-Based Cryptography or Lattice-Based Cryptography. It is the latter that is briefly introduced here.

Definition 2.25 - Lattices

A lattice $L \subset \mathbb{R}^n$ is the set of all linear combinations of vector from a basis $\boldsymbol{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$:

$$L = \left\{ \sum_{i=0}^{n} a_i \boldsymbol{b}_i \ , \ a_i \in \mathbb{Z} \right\}$$

We now introduce some computational problems based on lattices. We begin with the basic Shortest Vector Problem.

Definition 2.26 - Shortest Vector Problem (SVP)

Given a lattice L, the Shortest Vector Problem consists in finding the shortest non-zero vector in L. In other words, giving a lattice L and its basis $\boldsymbol{B}$, find $\boldsymbol{x} \in L$ such that

$$\|\boldsymbol{x}\| = \lambda(L) := \min_{v \in L \setminus \{0\}} \|v\|$$

A weaker version of the SVP includes a factor γ representing the relaxation of the previous problem where the goal is not necessary to find the shortest vector, but a vector more or less close to the shortest vector, according to γ.

Definition 2.27 - Approximate Shortest Vector Problem (SVP_γ)

Given an n-dimensional lattice L and its basis $\boldsymbol{B}$, find $\boldsymbol{x} \in L$ such that

$$\|\boldsymbol{x}\| \ \leq \gamma(n) \cdot \lambda(L) = \min_{v \in L \setminus \{0\}} \|v\|$$

A last problem consists in deciding if the shortest vector of a given lattice has a norm smaller than 1 or higher than the parameter γ.

Definition 2.28 - Decisional Approximate Shortest Vector Problem (GapSVP_γ)

Given a basis $\boldsymbol{B}$ of an n-dimensional lattice $L = L(\boldsymbol{B})$ where either $\lambda(L) \leq 1$ or $\lambda(L) > \gamma(n)$, determine which is the case.

We now introduce another problem, namely the Learning With Errors (LWE) problem. This problem is not a variation of the SVP problem, but in his seminal paper [Reg05], Regev proved that solving the worst-case hardness of this problem can be reduced to the GapSVP_γ problem.

Definition 2.29 - Learning With Errors (LWE)

Let $q \geq 2$, and χ be a distribution over $\mathbf{G}$. Given a polynomial number of samples, the *Learning With Errors* problem $\text{LWE}_{\chi,q}$ consists in distinguishing the two following distributions:

- $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$, where $\mathbf{a}$ is uniform in $\mathbf{G}_q^n$, $e \leftarrow \chi$, and $\mathbf{s} \in \mathbf{G}_q^n$ is a fixed secret chosen uniformly, and where $\langle \mathbf{a}, \mathbf{s} \rangle$ denotes the standard inner product.
- $(\mathbf{a}, b)$, where $\mathbf{a}$ is uniform in $\mathbf{G}_q^n$, and b is uniform in $\mathbf{G}_q$.

2.6 Zero-knowledge proofs

A *Zero-knowledge proof* (ZKP) is an interactive protocol between a prover P, who wants to prove to a verifier V that he knows a witness w for a given statement x in a language.

Definition 2.30 - Language

A language $\mathcal{L}$ is a subset of a space of words $\mathcal{X}$ with witness relation $\mathfrak{R}_{\mathcal{L}}$ such that

$$\mathcal{L} = \{x \in \mathcal{X} \mid \exists\ w \textbf{ s.t. } \mathfrak{R}_{\mathcal{L}}(x, w) = 1\}.$$

At the end of the protocol, the verifier should have learned nothing more about the statement than its veracity. Zero-Knowledge proofs can be made non-interactive, and we only consider such Non-Interactive Zero-Knowledge proofs (NIZK) in the remaining of the thesis.

Definition 2.31 - Zero-Knowledge proofs

A Non-Interactive Zero-knowledge Proof NIZK is a tuple of probabilistic polynomial time NIZK = (Setup, ZKGen, ZKVerify, ZKSim) as follows:

- Setup(1^{κ}) generates global parameters **param** based on the security parameter. Returns **param** .
- ZKGen$(x, \mathcal{L}, w)$ generates π_x, a zero-knowledge proof of the statement $(x \in \mathcal{L})$, using the witness w property and the relation $\mathfrak{R}_{\mathcal{L}}$. Returns π_x.
- ZKVerify$(x, \mathcal{L}, \pi_x)$ verifies if π_x is a correct proof of the statement $(x \in \mathcal{L})$. Returns **accept** if true, **reject** otherwise.
- ZKSim$(x, \mathcal{L})$ simulates a proof π_x of any (possibly false) statement $(x \in \mathcal{L})$ without any witness. Returns π_x.

A zero-knowledge proof of knowledge requires three properties:

- **Completeness**: ZKGen always generate an acceptable proof when $x \in \mathcal{L}$;
- **Soundness**: no adversary can generate an acceptable proof when $x \notin \mathcal{L}$, but with negligible probability;
- **Zero-Knowledge**: ZKSim$(x, \mathcal{L})$ generates proofs that are indistinguishable to proofs generated by ZKGen$(x, \mathcal{L}, w)$, on valid statements but without the witness.

When the verifier is assumed to be honest, we use *Honest Verifier Zero Knowledge* (HVZK) proofs

2.7 Smooth Projective Hash Functions

Introduced in 2002 [CS02], Smooth Projective Hash Functions (SPHF), also known as Hash Proof Systems (HPS), initially aimed to be the first public key encryption scheme secure against chosen ciphertext attacks. Nowadays, SPHF are mainly used for Honest Verifier Zero Knowledge Proofs or witness encryption. SPHFs are based on languages. More precisely, Smooth Projective Hash Functions work on NP-languages $\mathcal{L} \subset \mathcal{X}$. They require two keys: a randomly generated key **hk**, based on the description of $\mathcal{L}$, and a second key **hp** computed depending on **hk**. On the one hand, the key **hk** is used to link a word $x \in \mathcal{X}$ to its hash H_{hk}. On the other hand, the key **hp** is used together with a word $x \in \mathcal{X}$ and a witness w to compute a hash H_{hp}. For every word $x \in \mathcal{L}$ and its witness w, we have $H_{\mathsf{hk}} = H_{\mathsf{hp}}$. This is the *correctness* property of an SPHF. In opposition, when $x \notin \mathcal{L}$, H_{hk} should be indistinguishable from random.

Definition 2.32 - Smooth Projective Hash Functon

An SPHF defined on $\mathcal{L} \subset \mathcal{X}$ with values in $\mathcal{V}$ is defined by five algorithms:

- $\mathsf{Setup}(1^\kappa)$ generates the global parameters param from 1^κ, which includes a description of $\mathcal{L}$. Returns param .
- $\mathsf{HashKG}(\mathsf{param})$ generates a random hash key hk. Returns hk.
- $\mathsf{ProjKG}(\mathsf{hk}, [x])$ derives the projection key hp, potentially based on a word x. Returns hp.
- $\mathsf{Hash}(\mathsf{hk}, x)$ computes the hash value $H_{\mathsf{hk}} \in \mathcal{V}$ associated to the word x. Returns H_{hk}.
- $\mathsf{ProjHash}(\mathsf{hp}, x, w)$ computes $H_{\mathsf{hp}} \in \mathcal{V}$ using a witness w linked to the word x. Returns H_{hp}.

Those algorithms should ensure two requirements:

- **Correctness:** For any $x \in \mathcal{L}$, with witness w such that $\mathcal{R}_{\mathcal{L}}(x, w) = 1$, we have $H_{\mathsf{hk}} = H_{\mathsf{hp}}$.
- **Smoothness:** For any $x \in \mathcal{X} \backslash \mathcal{L}$, the distributions of $(\mathsf{hp}, H_{\mathsf{hk}})$ and $(\mathsf{hp}, v \leftarrow\!\!\$\ \mathcal{V})$ are indistinguishable.

A summary of the definition and properties of SPHF is depicted in Figure 2.2.

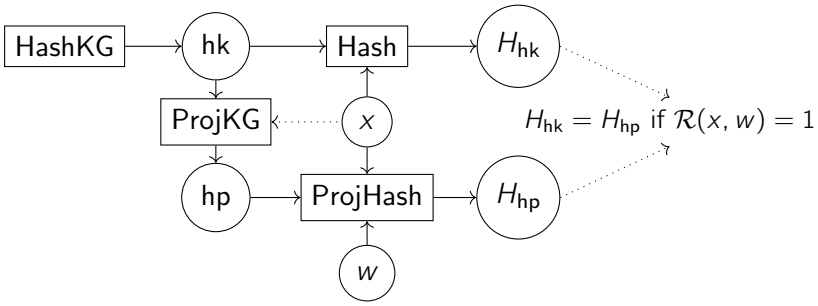


Figure 2.2: Smooth Projective Hash Functions Summary from [Ben16]

The aforementioned definition of Smooth Projective Hash Function states that the key hp may be computed according to x. In the initial definition provided by Cramer and Shoup [CS02], the key hp does not depend on x. Hence, an SPHF using this definition is denoted CS-SPHF and is expected to provide CS-smoothness. A first variation has been provided by Gennaro and Lindell in [GL03], where hp depends on x, leading to the notion of GL-SPHF and GL-smoothness. A second variant, introduced by Katz and Vaikuntanathan in [KV09] considers the ability for an attacker to maliciously generate the word x after seeing the projection key hp. In KV-SPHF, the projection depends only on the hashing key and ensure the smoothness holds even if the word x is chosen after having seen the projection key.

The definitions of Gennaro and Lindell will be enough for the project introduced in section 2.1. Therefore, when using SPHF, we implicitly refer to GL-SPHF, with GL-smoothness.

2.8 Secure multi-party computation

Suppose you are millionaire and at a restaurant table with a millionaire friend. You agree to let the richest of you pay the bill, but do not want to reveal your exact wealth. How to proceed ? A first solution is to find a trusted third person, who will take knowledge of your wealth as well as your friend's. Then, he will compare them and tell who pays. However, if you do not even trust your friend, who will you trust? So let's update the problem: *How to proceed without telling anyone your wealth?*

This problem, known as the Yao's millionaires' Problem, was stated by Andrew Yao in [Yao86]. While the notion of secure computation was first introduced by Shamir, Rivest and Adleman consider a mental poker game [SRA81], Yao formally introduced it as secure two-party computation (2PC). More formally, secure two-party computation implies two parties, commonly denoted Alice and Bob with respective inputs x_A and x_B. Their goal is to compute

$$z = (z_A, z_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$$

where $f_A(x_A, x_B)$ is the output returned to Alice and $f_B(x_A, x_B)$ the output returned to Bob. In terms of privacy, Bob should learn nothing more about Alice's input than what he learns from his output, and vice versa.

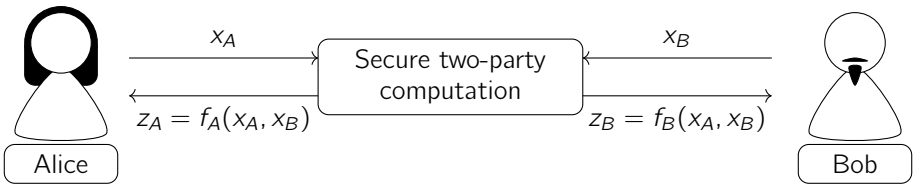


Figure 2.3: Secure two-party computation protocol

In a multiparty secure computation protocol, we do not only consider the adversary as an external eavesdropper, but also as one of the participants. In the latter case, his power is defined according to his ability to deviate from the protocol.

MPC security models. If the adversarial party is forced to follow the protocol, but can try to extract information from the transcript of the protocol, he is said to be *Honest-but-Curious*. If he can deviate from the protocol, he is said to be *malicious*. A goal of a malicious party is not only to extract sensitive information, he also may orient the protocol in a well-chosen way. In [GMW87], Goldreich *et al.* introduced a compiler, converting a protocol secure against Honest-but-Curious adversaries into a protocol secure against malicious adversaries. However, protocols secure against malicious adversaries are often inefficient. Thus, Aumann and Lindell introduced *covert* adversaries [AL07]. Rather than preventing malicious adversaries, the model of covert adversaries detects (with good probability) and reports malicious adversaries, potentially bringing discredit on them.

We now describe the main primitives to perform secure multiparty computation.

2.8.1 Oblivious Transfers

Oblivious Transfers (OT), first introduced in [EGL82] are a primitive widely used as a component of larger multiparty computation protocols as they allow a party to obliviously retrieve a value from another party, with privacy guarantees for both of them.

An Oblivious Transfer, in its simplest form, is a cryptographic primitive implying a *sender* with two messages (m_0, m_1) and a *receiver* with a selection bit b. The receiver aims to retrieve m_b, such that the sender does not learn any information about b. Also, the receiver must not learn any information about m_{1-b}. As a secure two-party computation, an instantiation of an Oblivious Transfer can be written as: $f(b, (m_0, m_1)) = (m_b, \emptyset)$. This basic protocol can be generalized, where the receiver aims to retrieve k messages among n and is denoted k-out-of-n OT.

We now give a formal definition of an Oblivious Transfer protocol:

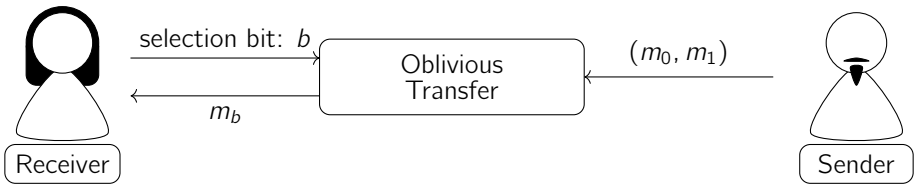


Figure 2.4: Oblivious Transfer protocol

Definition 2.33 - Oblivious Transfer

An Oblivious Transfer OT can be defined using a tuple of PPT algorithms $\mathsf{OT} = (\mathsf{Encode}, \mathsf{Compute}, \mathsf{Decode})$ as follows:

- $\mathsf{Encode}(b)$ generates a pair $\widetilde{b} = (\widetilde{b_{\mathsf{Enc}}}, \widetilde{b_{\mathsf{Dec}}})$. Returns $\widetilde{b}$
- $\mathsf{Compute}((m_0, m_1), \widetilde{b_{\mathsf{Enc}}})$ encodes m_0 and m_1 as $(\widetilde{m_0}, \widetilde{m_1})$ using the encoded expected bit $\widetilde{b_{\mathsf{Enc}}}$. Returns $(\widetilde{m_0}, \widetilde{m_1})$
- $\mathsf{Decode}((\widetilde{m_0}, \widetilde{m_1}), \widetilde{b_{\mathsf{Dec}}})$ decodes $(\widetilde{m_0}, \widetilde{m_1})$ using $\widetilde{b_{\mathsf{Dec}}}$. Returns m_b

We now describe two simple constructions, one that is proven secure in the honest-but-curious model, while the other one is secure against malicious parties.

Bellare-Micali's Construction [BM90]. We first introduce the construction from Bellare and Micali. This construction relies on the discrete logarithm problem and uses ElGamal encryption scheme.

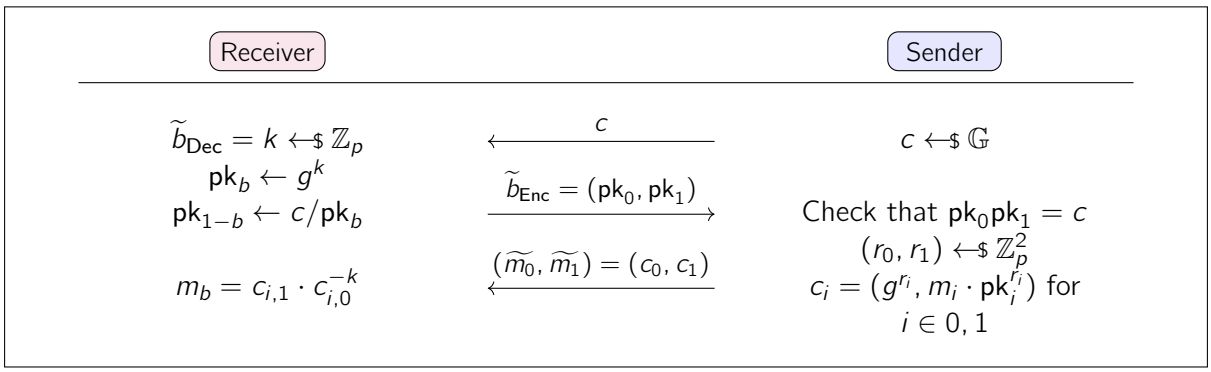


Figure 2.5: Bellare-Micali's Oblivious Transfer

This construction is secure against honest-but-curious adversaries under the DDH assumption. However, in the case of a malicious receiver, we can not prove whether the DDH assumption is enough to guarantee the security. At least, no proof has been provided so far. Hence, we can not exclude the case where the receiver generates pk_0 and pk_1, knowing partial information about each private keys, allowing her to extract information about both m_0 and m_1. Even if she is not able to totally recover m_0 and m_1, knowing any bit of information about more than a message is considered as a security leak. For example, being able to extract a single bit of $m_0 + m_1$, even without extracting m_0 or m_1 is a security leak that makes the protocol unsecure.

We now describe an Oblivious Transfer protocol from Naor and Pinkas, safe against malicious adversaries.

Naor-Pinkas construction [NP01]. Consider a group $\mathbb{G}$ where the DDH assumption holds.

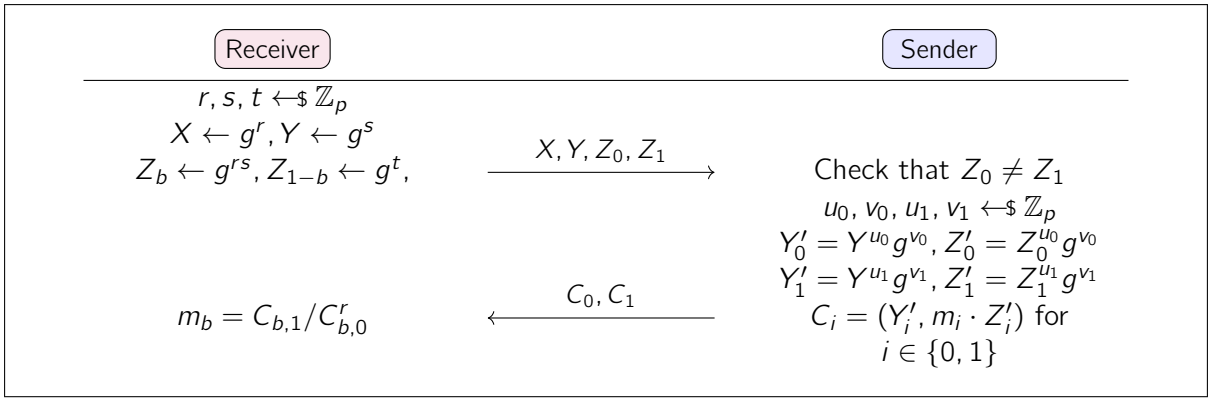


Figure 2.6: Naor-Pinkas' Oblivious Transfer construction

In this protocol, the protection against malicious adversaries is provided by the randomization of Y, Z_b and Z_{1-b}. Indeed, (X, Y, Z_b) is a DDH tuple, and so is (X, Y'_b, Z'_b), but as $Z_0 \neq Z_1$, the tuple (X, Y, Z_{1-b}) can not be a DDH tuple. In addition, the randomization operated by the sender transforms (X, Y, Z_b) in a perfectly inoperable random tuple (X, Y'_{1-b}, Z_{1-b}).

In some context, one may be required to compute many Oblivious Transfers. To reduce costs, a solution is to use Extended Oblivious Transfers.

2.8.1.1 Extended Oblivious Transfer [IKNP03]

When it comes to large amount of Oblivious Transfers to compute, using classical Oblivious Transfers as described previously may represent a massive communication cost. Suppose the receiver gets $\sigma = (\sigma_1, \cdots, \sigma_n)$ as input and the sender gets $((m_0^0, m_0^1), \ldots, (m_n^0, m_n^1))$ with $n \gg k = |m_i^j|$.

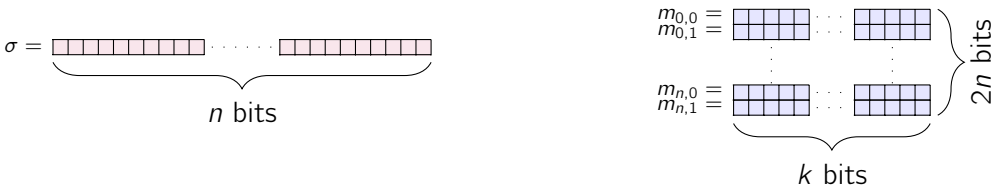


Figure 2.7: Initial Sender and Receiver inputs

The first step of OT extension is to generate new inputs. The receiver, Alice, generates a random bit-matrix T of size $n \times k$ and generates k pairs $(T_i, T_i \oplus \sigma)$. The sender, Bob, generates a random string $s \in \{0, 1\}^k$.

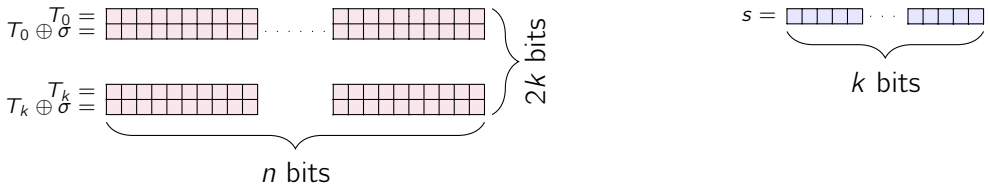


Figure 2.8: New Sender and Receiver inputs

One may guess the next step by comparing Figures 2.7 and 2.8. Alice and Bob will proceed to $k \ll n$ standard Oblivious Transfers but with Alice as sender and Bob as receiver. After the k

Oblivious Transfers, Bob has k strings, each of length n bits. We denote by Q_i the i-th message retrieved by Bob. We have :

$$Q_i = \begin{cases} T_i & \textbf{if } s_i = 0. \\ T_i \oplus \sigma & \textbf{if } s_i = 1. \end{cases}$$

Let T^j and Q^j denoting the j-th column of T and Q. We have $T_i^j = Q_i^j$ when $\sigma_j = 0$ no matter the value of s_i. When $\sigma_j = 1$, this has an impact only when $s_i = 1$. More precisely, when $\sigma_j = 1$ we have $T_i^j = Q_i^j \oplus s_i$. Hence, we have:

$$T^j = \begin{cases} Q^j & \textbf{if } \sigma_j = 0. \\ Q^j \oplus s & \textbf{if } \sigma_j = 1. \end{cases}$$

With this relation, Bob can now send to Alice the initial messages. Given a hash function H, Bob computes and sends

$$y_j^0 = H(j, Q^j) \oplus m_0^j \quad \text{and} \quad y_j^1 = H(j, Q^j \oplus s) \oplus m_1^j$$

Finally Alice computes $m_{\sigma_j}^j = H(j, T^j) \oplus y_j^{\sigma_j}$, the list of the j-th bit of each message m_{i,σ_i}.

2.8.2 Garbled Circuits

In this section, we describe the Garbled Circuits as a technique. We refer the reader to [BHR12] for a definition of Garbled Schemes as a primitive. The Introduction of Garbled Circuits is first credited to Andrew Yao as he introduced the idea in the oral presentation of [Yao86]. Then, this idea was first put down on paper in 1987 [GMW87] while the first appearance of the term *Garbled Circuits* is in 1990 [BMR90].

A circuit is a boolean representation of a function. This representation uses logical gates as computation steps. A logical gate is a small function taking 1 bit as input (NOT) or 2 (AND, OR, NAND, NOR, XOR or XNOR) and returning one bit as output. As NOT, OR, NAND, NOR and XNOR gates can be rewritten using only AND and XOR gates, this is enough to focus on those two logical gates. AND returns 1 only if both inputs bit equal 1 and 0 if at least one of the input bit is 0. XOR computes the exclusive-OR and returns 1 only if the input bits are not equal, and 0 otherwise.

α	β	$\alpha \wedge \beta$
0	0	0
0	1	0
1	0	0
1	1	1

(a)

α	β	$\alpha \oplus \beta$
0	0	0
0	1	1
1	0	1
1	1	0

(b)

Table 2.2: Truth table for the AND logical gate, denoted $\wedge$ (a) and the XOR logical gate, denoted $\oplus$ (b)

When a circuit is garbled, that can be seen as an "encryption" of the circuit. A Garbled Circuits implies two parties, a *Garbler* that is interpreted by Gisele, and an *evaluator*, that is represented by Edmond. Input bits are represented with random keys, also called labels, that only the garbler can invert to associated bits. Then, using Oblivious Transfers, the evaluator retrieves the encoding of his bits, generated by the garbler, and can then evaluate the circuit using the ciphetexts computed by the generator. Finally, to convert the output label of the circuit, the garbler sends, together with the circuit and her input labels, a transition table mapping output label with the corresponding output bit.

We first describe the garbling phase as it was initially done in [Yao86], using double encryption.

2.8.2.1 Garbling

The initial garbling solution uses a double symmetric encryption Π. Given a function $f : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}^*$, Gisele first represents f as a circuit $\mathcal{C}_f$ made of AND and XOR gates. Then, gates after gates, Gisele garbles the circuit $\mathcal{C}_f$. For each gate, she generates three pairs of keys : a pair (k_0^G, k_1^G) for the wire 1, defined as Gisele's input wire, a pair (k_0^E, k_1^E) for the wire 2, set as Edmond's wire, and a pair (k_0^{out}, k_1^{out}) for the output wire.

First, let us focus on the case where Gisele's input is 0, and Edmond's input is 0. Her key will be k_0^G and Edmond's key will be k_0^E. Looking at the Table 2.2, the output of the non-garbled gate is $0 \wedge 0 = 0$ which associated key is k_0^{out}. Hence, Gisele computes $E_{k_0^G, k_0^E}(k_0^{out}) = \Pi.\mathsf{Enc}(H(k_0^G || k_0^E), k_0^{out})$ where H is a Hash Function. Applying the same process for all possible cases leads to the table obtained in Table 2.3.

wire 1	wire 2	output	G's key	E's key	Output key	Garbled Table
0	0	0	k_0^G	k_0^E	k_0^{out}	$E_{k_0^G, k_0^E}(\ k_0^{out}\ )$
0	1	0	k_0^G	k_1^E	k_0^{out}	$E_{k_0^G, k_1^E}(\ k_0^{out}\ )$
1	0	0	k_1^G	k_0^E	k_0^{out}	$E_{k_1^G, k_0^E}(\ k_0^{out}\ )$
1	1	1	k_1^G	k_1^E	k_1^{out}	$E_{k_1^G, k_1^E}(\ k_1^{out}\ )$

Table 2.3: Unpermutted Garbled Table for an AND gate

Analogously, the garbling of an XOR gate is given in Table 2.4.

wire 1	wire 2	output	G's key	E's key	Output key	Garbled table
0	0	0	k_0^G	k_0^E	k_0^{out}	$E_{k_0^G, k_0^E}(\ k_0^{out}\ )$
0	1	1	k_0^G	k_1^E	k_1^{out}	$E_{k_0^G, k_1^E}(\ k_1^{out}\ )$
1	0	1	k_1^G	k_0^E	k_1^{out}	$E_{k_1^G, k_0^E}(\ k_1^{out}\ )$
1	1	0	k_1^G	k_1^E	k_0^{out}	$E_{k_1^G, k_1^E}(\ k_0^{out}\ )$

Table 2.4: Unpermutted Garbled Table for an XOR gate

Once the garbler generated a garbled table, she permutes its entry such that an entry index does not depend on the inputs. Then, when garbling the last gate, she generates the transition table $((K_0^{out}, 0); (K_1^{out}, 1))$, later allowing the evaluator to convert the inoperable output key in an output bit. Once done, Gisele sends the Garbled Circuits $\widetilde{\mathcal{C}_f}$ containing the garbled tables, the transition table T, and its input keys $k_\alpha^G = (k_{i,\alpha[i]}^G)$ to Edmond (where $k_{\alpha[i]}^G$ is the label corresponding to i-th bit of α), who will proceed to the evaluation.

2.8.2.2 Evaluation

When the evaluator receives the garbled tables, the input labels of Gisele and the transition table from Gisele, he has to retrieve the keys $k_\beta^E = (k_{i,\beta[i]}^E)$ corresponding to his input. As, on the one hand, Edmond wants his input to remain private and, on the other hand, if Gisele reveals both keys for each input wire, this would allow Edmond to decrypt two entries by garbled tables. The solution for Edmond is to retrieve his key using one Oblivious Transfer by input bit, where Gisele will be the sender and Edmond the receiver. Once Edmond retrieved his keys, he evaluates each gate, where for each gate, Edmond has two keys, k_A and k_B, that he used to decrypt entries of the corresponding Garbled Table. Once a decryption success, detected thanks to some padding, the freshly obtained key as output of the gate is now either used as an input of another gate, or is converted to an output bit thanks to a transition table.

Cost. We can see that for each gate, the garbler needs to generate a four entry garbled table (one for each key combination). Denoting n_{AND} the amount of AND gates, n_{XOR} the amount of XOR gates and κ the security parameter (i.e. the bit-length of the keys), the total communication cost is of $4(n_{\text{AND}} + n_{\text{XOR}}) \cdot \kappa$ bits. Also, the Evaluator will have to decrypt the four entries in the worst cases and 2.5 keys on average. This can be considerably reduced, using optimizations.

2.8.2.3 Optimizations

We now present the main optimizations leading to the best performances for implementing Garbled Circuits. As a toy example, we describe the ciphertexts obtained for all optimizations for a circuit computing $\underbrace{(A \wedge B)}_{D} \oplus C$. Without optimization, the resulting garbled table are:

$E_{k_1^A, k_0^B}(\, k_0^D \,)$	$E_{k_1^D, k_0^C}(\, k_1^{out} \,)$
$E_{k_0^A, k_1^B}(\, k_0^D \,)$	$E_{k_0^D, k_0^C}(\, k_0^{out} \,)$
$E_{k_1^A, k_1^B}(\, k_1^D \,)$	$E_{k_1^D, k_1^C}(\, k_0^{out} \,)$
$E_{k_0^A, k_0^B}(\, k_0^D \,)$	$E_{k_0^D, k_1^C}(\, k_1^{out} \,)$

Table 2.5: Garbled tables - Without optimizations

FreeXOR. In 2008, Kolesnikov *et al.* introduced freeXOR [KS08]. The freeXOR technique applies to XOR gates and works as follows: The garbler first randomly generates k_0^G and k_0^E and a random offset $\Delta \in \{0,1\}^\kappa$. Then she sets $k_1^G = k_0^G \oplus \Delta$, $k_1^E = k_0^E \oplus \Delta$ and $k_0^{out} = k_0^G \oplus k_0^E$, $k_1^{out} = k_0^{out} \oplus \Delta$. With such a setup, the evaluator can evaluate a XOR gate simply by computing a XOR between both input keys. Figure 2.9 shows the correctness of this optimization by testing all possible key combination.

		Gisele's key	
		k_0^G	k_1^G
Edmond's key	k_0^E	$k_0^G \oplus k_0^E = k_0^{out}$	$k_1^G \oplus k_0^E = k_0^G \oplus R \oplus k_0^E = k_0^{out} \oplus R = k_1^{out}$
	k_1^E	$k_0^G \oplus k_1^E = k_0^G \oplus k_0^E \oplus R = k_0^{out} \oplus R = k_1^{out}$	$k_1^G \oplus k_1^E = k_0^G \oplus R \oplus k_0^E \oplus R = k_0^{out}$

Figure 2.9: FreeXOR optimization

This optimization does not require the garbler to send a garbled table to the evaluator as the evaluation is made by xoring the two input keys. Hence, the communication and computation cost of XOR gates are free with this optimization.

As the evaluation of XOR gates are now free, the total communication and computation cost of the protocol relies only on the AND gates. Recall that without optimization, the evaluation of an AND gate costs 4 ciphertexts and 2.5 decryptions on average.

Point and Permute. To reduce the amount of decryptions, Beaver *et al.* introduced the *Point and Permute (P&P)* optimization in [BMR90]. When considering an AND gate with the keys $(k_0^G, k_1^G); (k_0^E, k_1^E)$, Gisele samples random *signal* bits p_G and p_E, and update the keys as

$E_{k_1^A,k_0^B}(\ k_0^D\ )$
$E_{k_0^A,k_1^B}(\ k_0^D\ )$
$E_{k_1^A,k_1^B}(\ k_1^D\ )$
$E_{k_0^A,k_0^B}(\ k_0^D\ )$

Table 2.6: Garbled tables - With FreeXOR

$$(k_0^G, k_1^G) \leftarrow (k_0^G \| p_G\ ,\ k_1^G \| \overline{p_G}) \text{ and } (k_0^E, k_1^E) \leftarrow (k_0^E \| p_E\ ,\ k_1^E \| \overline{p_E}).$$

Then, the garbler permutes the garbled gates according to the signal bits. Consider the input keys k_0^G and k_0^E, their respective signal bit is p_G and p_E. Thus, the garbled table entry $E_{k_0^G,k_0^E}(\ k_0^{out}\ )$ has for index $2 \times p_G + p_E$. The garbler acts in the same way, mutatis mutandis, for the other key combinations. When evaluating, Edmond extracts the signal bit of each key used as inputs of the gate and deduces the index of the ciphertext to decrypt. This reduces the computational cost from 2.5 decryptions in average to a single decryption in any case.

As this is the best computational cost we can expect for an AND gate, we now introduce optimizations to reduce the communication cost of the AND gates. Also, now that the evaluator has only one decryption to proceed, no more padding is required, and we can set $E_{k_A,k_B}(\ k_{out}\ ) = k_{out} \oplus H(k_A||k_B)$ where H is a hash function.

Garbled Row Reduction 3. Let $k_A \in \{k_0^G, k_1^G\}$ and $k_B \in \{k_0^E, k_1^E\}$ be input keys with signal bit at 0. In [NPS99], Naor *et al.* introduced *Garble Row Reduction 3* (GRR3) where k_{out} is set as $\mathsf{Dec}_{k_A,k_B}(0)$. The garbler therefore only needs to send three ciphertexts instead of four, and decrypts as before, where the encryption at index 0 of the AND gate garbled table is 0. This reduces the communication cost by a factor $\frac{1}{4}$.

$E_{k_0^A,k_1^B}(\ k_0^D\ )$
$E_{k_1^A,k_1^B}(\ k_1^D\ )$
$E_{k_0^A,k_0^B}(\ k_0^D\ )$

Table 2.7: Garbled tables - With Garbled Row Reduction 3

Garbled Row Reduction 2. Pinkas *et al.* in [PSSW09] introduced *Garbled Row Reduction 2* (GRR2). They use polynomial interpolation to reduce the communication cost such that only two ciphertexts by AND gate are required. The idea is to generate two polynomials. To convert labels to coordinates, Gisele acts as follows: Given two keys k_A, k_B, the abscissa is $1+2\times p(k_A)+p(k_B)$ where $p(\cdot)$ extracts the signal bits, while the ordinate is given by $H(k_A||k_B)$, where H is a hash function. For the sake of simplicity, we set $p_G = 0$ and $p_E = 1$, hence the four points are:

$$(1, H(k_0^G||k_1^E)), (2, H(k_0^G||k_0^E)), (3, H(k_1^G||k_1^E)), (4, H(k_1^G||k_0^E)).$$

We have seen in Table 2.3 that three combinations of keys yield to the output label k_0^{out} and only one to k_1^{out}. Hence, the first step is to interpolate the points associated to these three combinations, generating a degree-2 polynomial $\mathcal{P}$. In our case, those points are

$$(1, H(k_0^G||k_1^E)), (2, H(k_0^G||k_0^E)), (4, H(k_1^G||k_0^E)).$$

The key k_0^{out} is set as $\mathcal{P}(0)$. As the evaluator will only know one point of the polynomial, he needs two more points to compute the interpolation and retrieve k_0^{out}. Those points are set as $(5, \mathcal{P}(5))$

and $(6, \mathcal{P}(6))$. Then, the garbler interpolates $(5, \mathcal{P}(5))$ and $(6, \mathcal{P}(6))$ with the remaining point (in our case $(3, H(k_1^G || k_1^E))$) to generate a second polynomial $\mathcal{Q}$. This implies that $\mathcal{P}(5) = \mathcal{Q}(5)$ and $\mathcal{P}(6) = \mathcal{Q}(6)$. Similarly, the key k_1^{out} is set as $\mathcal{P}(0)$. The ciphertexts of the gate Gisele sends to Edmond are $\mathcal{P}(5)$ and $\mathcal{P}(6)$. Finally, the evaluator interpolates $(5, \mathcal{P}(5))$ and $(6, \mathcal{P}(6))$ with the hash of the input labels.

The reader may note that with this optimization, there is no way to maintain the offset used in the FreeXOR optimization. Hence, the GRR2 optimization is not compatible with the FreeXOR optimization. However, the GRR2 can also be applied to the XOR gates with this method. The idea is nearly the same as for AND gates, except that, as both outputs bit are returned in two combinations, one generate two affine polynomials $\mathcal{P}$ and $\mathcal{Q}$, and the ciphertexts consist in $\mathcal{P}(5)$ and $\mathcal{Q}(5)$. A graphical representation of this optimization is depicted in Figure 2.10.

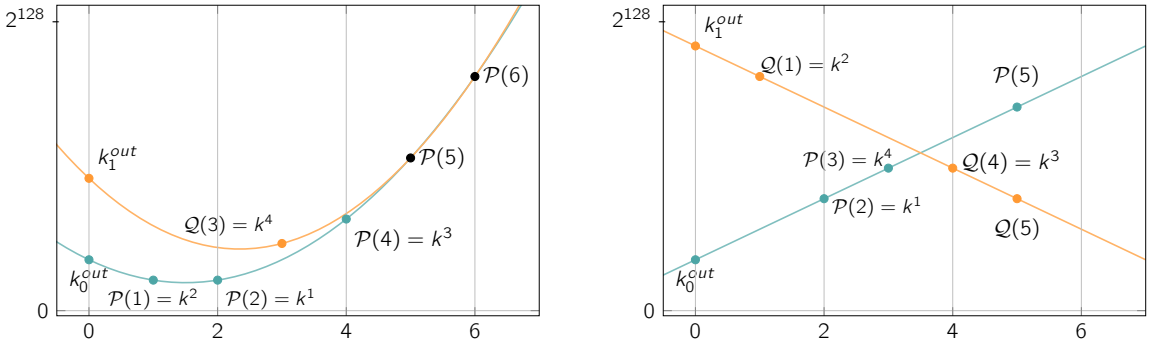


Figure 2.10: Graphical representation of GRR2 with an AND gate on the left and an XOR gate on the right. In orange is the polynomial $\mathcal{P}$ associated to bit output 0, and in green the polynomial $\mathcal{Q}$ associated to the bit output 1.

Hence, we have two ciphertexts by gate, whether it is an AND gate or an XOR gate. This made the GRR2 optimization better than combining GRR3 and FreeXOR if there is less than twice more XOR gates than AND gates.

$\mathcal{P}(5)$
$\mathcal{P}(6)$

$\mathcal{P}(5)$
$\mathcal{Q}(5)$

Table 2.8: Garbled tables - With Garbled Row Reduction 2 optimization

However, in 2015, Zahur *et al.* introduced HalfGates, resulting in a two-ciphertexts table, compatible with the freeXOR optimization.

HalfGates. In [ZRE15], Zahur *et al.* introduced *Halfgates* where they used the fact that for any $r \in \{0, 1\}$:

$$a \wedge b = (a \wedge r) \oplus (a \wedge (b \oplus r)).$$

The Halfgates optimization sets $r = p_B$, the random select bit for the wire B.

A – Garbler's half gate. In the first half gate, Gisele knows r. Thus, the output only depends on a. If $a = 0$, the output of the half gate is always 0, hence the first ciphertext is $H(k_0^A) \oplus k_0^G$. If $a = 1$, the output depends on the value of r, hence the second ciphertext is $H(k_1^A) \oplus k_r^G$. When evaluating, Edmond looks at the select bit of k_A and xor the corresponding ciphertext with $H(k_A)$.

Correctness. If $a = 0$, we have $k^A = k_0^A$. Edmond then computes

$$\left(H(k_0^G) \oplus k_0^G\right) \oplus H(k^A) = H(k_0^G) \oplus k_0^G \oplus H(k_0^G) = k_0^G = k_{0\wedge r}^G.$$

If $a = 1$, we have $k^A = k_1^A$, Edmond then computes

$$\left(H(k_1^G) \oplus k_r^G\right) \oplus H(k^A) = H(k_1^G) \oplus k_r^G \oplus H(k_1^G) = k_r^G = k_{1\wedge r}^G.$$

Edmond indeed computed $k_{a\wedge r}^G$.

B – Evaluator's half gate. In the second half gate, Edmond can retrieve $b \oplus r$ without learning information about r. Indeed, if $b = 0, b \oplus r = r = p(k_0^E)$ by definition while if $b = 1, b \oplus r = \bar{r} = p(k_1^E)$. Hence, $s = b \oplus r = p(k_b^E)$, allowing Edmond to retrieve s without knowing b. The process for the evaluator half gate is trickier as Edmond has to behave differently according to s. First, Gisele computes $H(k_r^B) \oplus k_0^E$ as the first ciphertext and $H(k_{1-r}^B) \oplus k_0^G \oplus k_0^E$ as the second ciphertext. When evaluating, Edmond extracts s based on select bit of his key.

Correctness. If $s = b \oplus r = 0$, Edmond takes the first ciphertext and computes

$$\left(H(k_r^B) \oplus k_0^E\right) \oplus H(k_b^B) = k_0^E = k_{a\wedge 0}^E.$$

If $s = b \oplus r = 1$, he takes the second ciphertext and computes

$$\left(H(k_{1-r}^B) \oplus k_0^G \oplus k_0^E\right) \oplus H(k_b^E) \oplus k_0^G \oplus a\Delta = k_0^E \oplus a\Delta = k_{a\wedge 1}^E.$$

Edmond indeed computed $k_{a\wedge(b\oplus r)}^E$.

C – Two halves make a whole. The final step for Edmond consists in computing $k_{out} = k^G \oplus k^E$. Table 2.11 shows the correctness of the HalfGates optimization.

a	b	r	$k_{a\wedge r}^G$	$k_{a\wedge(b\oplus r)}^E$	k_{out}	$a \wedge b$
0	0	0	k_0^G	k_0^E	$k_0^G \oplus k_0^E = k_0^{out}$	0
		1	k_0^G	k_0^E	$k_0^G \oplus k_0^E = k_0^{out}$	0
	1	0	k_0^G	k_0^E	$k_0^G \oplus k_0^E = k_0^{out}$	0
		1	k_0^G	k_0^E	$k_0^G \oplus k_0^E = k_0^{out}$	0
1	0	0	k_0^G	k_0^E	$k_0^G \oplus k_0^E = k_0^{out}$	0
		1	$k_1^G = k_0^G \oplus \Delta$	$k_1^E = k_0^E \oplus \Delta$	$k_1^G \oplus k_1^E = k_0^G \oplus \Delta \oplus k_0^E \oplus \Delta = k_0^{out}$	0
	1	0	k_0^G	$k_1^E = k_0^E \oplus \Delta$	$k_0^G \oplus k_1^E = k_0^G \oplus k_0^E \oplus \Delta = k_0^{out} \oplus \Delta = k_1^{out}$	1
		1	$k_1^G = k_0^G \oplus \Delta$	k_0^E	$k_1^G \oplus k_0^E = k_0^G \oplus \Delta \oplus k_0^E = k_0^{out} \oplus \Delta = k_1^{out}$	1

Figure 2.11: Correctness table for the HalfGates optimization

We have seen that the garbler generates two ciphertexts by AND gate. However, for each AND gate, Gisele can use the Garble Row Reduction 3 optimization and set $k_0^G = H(k_0^A)$ making the first ciphertext of the garbler half gate equal to zero. Similarly, by setting $k_0^E = H(k_r^B)$, the first ciphertext of the evaluator half gate is zero. As a consequence, Zahur *et al.* reduce communication cost to two ciphertexts by AND gate while remaining compatible with the FreeXOR optimization.

$H(k_1^A) \oplus k_r^D$
$H(k_{1-r}^B) \oplus k_0^D \oplus H(k_r^B)$

Table 2.9: Garbled tables - With HalfGates optimization

2.8.2.4 From honest-but-curious to malicious Security

The evaluator does not really know if the circuit garbled by Gisele is exactly the one she was supposed to garble. She may act maliciously and garble a circuit leaking information about Edmond's input or direct the result in a way benefiting her.

Circuit Consistency. Hence, to provide circuit consistency, one may use the Cut & Choose technique [LP07]. This technique constrains the garbler to generate and send λ circuits $\widetilde{\mathcal{C}}_1, \ldots, \widetilde{\mathcal{C}}_\lambda$. Then, the evaluator generates $\sigma \leftarrow\$ \{0,1\}^\lambda$ and asks Gisele to prove the subset of circuits $\{\widetilde{\mathcal{C}}_i \textbf{ s.t. } \sigma_i = 1\}$ by sending him the randomness she uses for garbling. Using the randomness received from Gisele, Edmond verifies the circuits he received. If they are all valid, Gisele and Edmond proceed normally with the remaining circuits. The final output is the majority output. Hence, as each circuit is checked with probability $\frac{1}{2}$, Gisele can cheat with probability $2^{-\lambda/2}$.

However, guaranteeing circuit consistency is not enough to prevent malicious attacks. Indeed, Gisele may also manipulate the inputs. We now show how to prevent input inconsistency.

Input Consistency. On the remaining circuits, Gisele may provide different inputs for each circuit. Taking the example of a circuit computing an inner product, Gisele sets her input to the i-th circuit as a vector of 0 except a 1 at i-th position such that the output is the i-th bit of Edmond's input. Hence, Gisele knows if Edmond's input contains more 0's or more 1's. To counteract this input consistency, one may use commitments, MAC or hash functions.

Lastly, Gisele may set Edmond's input labels such that the evaluation fails depending on Edmond's input. The solution [LP07] to counter the attack is to modify Edmond's input using error correcting codes. This solution has an impact on the circuit size and does not prevent from selective failure, but it avoids Gisele to extract information on Edmond's input.

2.9 Universal Composability

Consider a protocol between n parties denoted $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$ and respective outputs $(y_1, \ldots, y_n)$. In an *Ideal World*, we can imagine a trusted third party, called a *Functionality* and denoted $\mathcal{F}$. As $\mathcal{F}$ is trusted by all parties, they can privately send it their respective inputs. Then, $\mathcal{F}$ executes the protocol and returns the outputs to the respective parties. We denote by ϕ the protocol in such an ideal world.

However, in the real world, we can not rely on such a trusted third party. We have to consider the potential attacks made by an adversary $\mathcal{A}$. Therefore, the aim of a protocol running in the real world is to act similarly to the functionality running in the ideal world. As the adversary is part of the protocol, the distinguisher in this case is the environment, denoted $\mathcal{Z}$. The environment chooses the inputs $x_1, \cdots, x_n$ of the parties $P_1, \ldots, P_n$ and then has to distinguish between an execution of Π with the adversary $\mathcal{A}$ that has access to the communication between the parties but not their inputs nor outputs, and the execution of ϕ with an ideal adversary $\mathcal{S}$ interacting solely with the functionality $\mathcal{F}$.

Definition 2.34 - UC-emulation

We say that a protocol Π UC-emulates the protocol ϕ if for any real adversary $\mathcal{A}$ there exists an ideal simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$ we have:

$$\text{UC-EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{UC-EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$$

and a protocol Π UC-realizes an ideal functionality $\mathcal{F}$ if Π UC-emulates the ideal protocol for $\mathcal{F}$.

We represent in Figure 2.12 the Ideal/Real world paradigm represented by an execution of both the ideal and real protocol, and their interactions with their respective adversary $\mathcal{S}$ and $\mathcal{A}$, and the communication with the environment $\mathcal{Z}$.

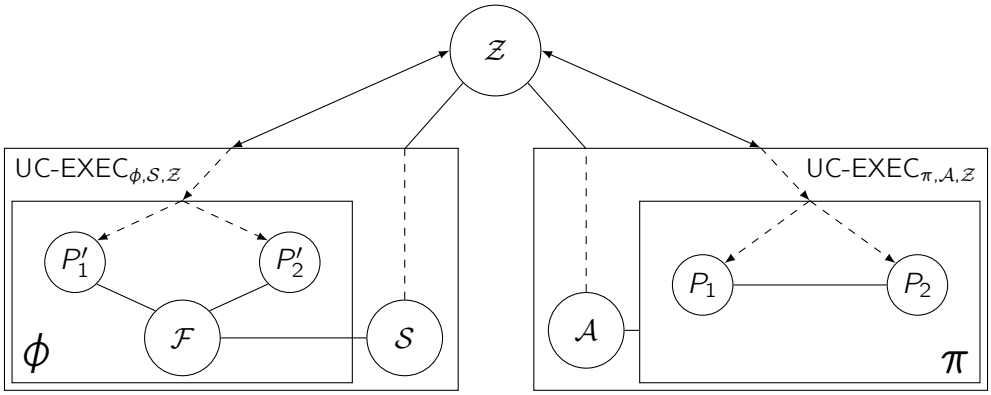


Figure 2.12: The Real/Ideal World paradigm, inspired from [GMP+08]

If the protocol Π UC-realizes the functionality $\mathcal{F}$ only if it is composed with a protocol Π' that UC-realizes the functionality $\mathcal{F}'$, we say that the protocol Π UC-realizes $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model.

2.10 Decision forest

Machine Learning (ML) is a branch of Artificial Intelligence giving algorithms the ability to learn from data and therefore improve their accuracy with the time. It is widely used nowadays by example for facial or vocal recognition, meteorological predictions, traffic... A popular technique to achieve Machine Learning is to use Decision Trees. A Decision Tree consists in branches, decision nodes and decision leaves. A decision node contains a predicate, taking a sample as input and outputting either **false** or **true**. According to the output of this predicate, the sample is directed to the corresponding next decision node, or decision leaf. When a sample reaches a decision leaf, it gets associated to a prediction. Either this prediction is a label (YES/NO, brands, animal species, etc...) and the decision tree is a classification tree, either the prediction is a value (house price, drive length, etc...) and the decision tree is a regression tree. In Figure 2.13, we build a decision tree answering the question: "Is it worth to take some time looking for a funny Decision Tree example?"

We call *depth* of the tree the maximum number of nodes from the root to a leaf. In Figure 2.13, the depth of the tree is 3.

A Decision Forest consists in a set of Decision Trees. Each tree returns a label, and the global label is the value returned a maximum number of time. However, if all trees of the forest are trained on the same database, they will all be very similar leading to *overfitting*. When a model

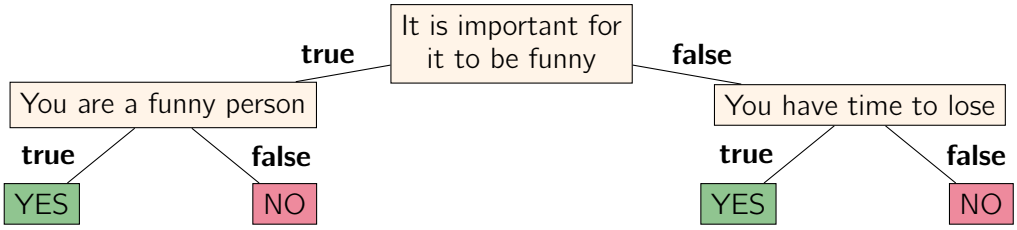


Figure 2.13: Decision Classification Tree deciding if it is worth to take some time looking for a funny Decision Tree Example. Blue boxes denote decision nodes, green and red boxes denote decision nodes.

overfits a set of data, it corresponds too closely to a particular set of data and can make future prediction erroneous.

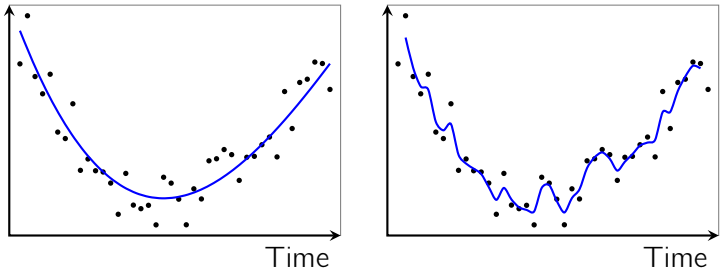


Figure 2.14: Representation of a good fitting and overfitting (LaTeX code from Tex StackExchage)

Overfitting may happen if the tree is too deep, making the model very specific to the training data. To overcome this problem, an idea is to select random features and random observations. Choosing random observations is named *tree bagging* while choosing random features is named *feature sampling*. By mutual agreement, with a database of m observations $\times n$ features, we sample (with replacement) $\sqrt{m}$ observations and $\sqrt{n}$ features. In Figure 2.15, we take the example of a 4×9 database. The tree bagging operation results in a 2×9 database, then after the feature sampling, we retrieve the final training dataset (in green), with size 2×4, as due to draw replacement, a column has been drawn twice.

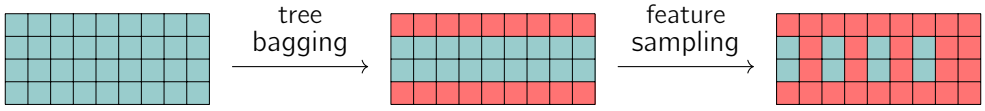


Figure 2.15: Database Sampling. The final training dataset is in green

Chapter 3

Secure decision forest evaluation

This chapter describes a protocol to securely evaluate binary decision trees, that are decision trees with binary outcome. Initially, this project has been motivated by the will to add a cryptographic layer to an already existing continuous authentication protocol based on Machine Learning. In continuous authentication, users are authenticated using a set of features that strengthen usual authentication credentials such as passwords or security tokens. When the user's identity needs to be validated after some time interval or after some inactivity, continuous authentication offers a user-friendly experience as it avoids interrupting legitimate users and reduces the number of times they have to authenticate explicitly. The usual scenario of continuous authentication consists of a server authenticating users based on behavioral biometrics. A variety of features such as keystroke patterns, swiping gestures and scrolling duration are collected on the user's device and sent to the server. The server then evaluates the client inputs with respect to its model in order to make an authentication decision. The model is usually generated during a training step using a dedicated training dataset.

It goes without saying that adding a privacy layer to such a protocol finds more applications than continuous authentication. Indeed, over the past years, companies have tremendously increased the amount of data they collect from their users. These data are often feed to machine learning algorithms in order to turn them into valuable business insights that are used to develop innovative services. In addition to continuous authentication, other applications include fraud detection in banking systems, recommendation services as well as spam detection. Collecting and processing these data raises privacy concerns since they generally contain sensitive information regarding users. Besides, the models used to evaluate these data may contain critical business information that also need to be protected. We consider the general case where a client who holds sensitive data interacts with a server who holds a decision forest model in order to jointly evaluate the client inputs with respect to the server model. Our goal is to ensure that the privacy of both the client and the server are guaranteed on their respective inputs. We also investigate the scenario where the client is malicious and intends to bias the outcome of the protocol.

The work presented in this chapter led to a publication in the 16th International Conference on Availability, Reliability and Security (ARES21):

★ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Secure decision forest evaluation. In *The 16th International Conference on Availability, Reliability and Security*, pages 1–12, 2021

Contents

Notations

In addition to global notations, we list here the notations used in the chapter

Decision Trees Notations	
$\mathcal{M}$	Model on the server
$\boldsymbol{C}$	Encoding of the model
P	Number of paths
δ	(Max) depth of the paths
ν	Feature range ($[0 \,..\, 2^{\nu} - 1]$)
τ	Global threshold
p_i	Polarity of the i-th path
$(t_{i,j})$	Threshold in the j-th node of the i-th path
$(v_{i,j})$	Orientation of the comparison of the j-th node of the i-th path
$(x_{i,j})$	Feature of the comparison of the j-th node of the i-th path

Garbled Circuits Notations	
$\mathfrak{C}$	Garbling of circuit C.
$\mathfrak{I}$	Garbled Circuits input labels
$\mathfrak{T}$	Transition tables
$\mathfrak{O}$	Output labels

Other	
ζ	NIZK proofs
$[\![\ m\ ]\!]$	Encryption of m

3.1 Honest-but-curious client and server

The first step of the protocol elaboration was to find the best way to encode the model while keeping privacy and being efficient. The model can be sent offline, and used multiple times. We can imagine for example the model updating itself each morning and is then usable online the whole day by the user.

3.1.1 Model encoding

As one of our requirements is a client-side computation, we first have to find the best way to represent and encode the decision forest. The first observation is that saving the structure of the tree implies leakage of the order of the required feature. Indeed, when evaluating, the first used feature would be the feature at the root of the tree and this violates the server privacy. Leaking used feature is an acceptable leakage, but leaking the position of the node in the tree in which the feature is used is a higher leakage, and we do not consider it as acceptable. Thus, we choose to "decompose" the tree as set of paths. Hence, rather than direct a sample through the correct nodes, to reach a single leaf, the sample will go through all possible paths leading to a leaf deciding 1. Indeed, as we consider binary output decision trees, only considering paths with label 1 is equivalent to evaluate the whole tree. Before encoding, we assume that the model has been computed on a normalized database, where each value is a positive integer smaller than 2^ν. In this chapter, we use the well-known iris database classing iris species, as an example. We consider a petal with length ℓ and width w as features. Width goes from 0.1 cm to 2.5 cm and length from 1 cm to 6.9 cm. We thus normalize by defining L as $(10(\ell - 1))$ and W as $20w$ leading to integer values in $[0 \,..\, 63 = 2^6 - 1]$. The model is trained to decide if a sample corresponds to the *virginica* species or not. Figure 3.1 shows the transition from the classical representation of a decision tree, to our representation for its encoding with the iris database example.

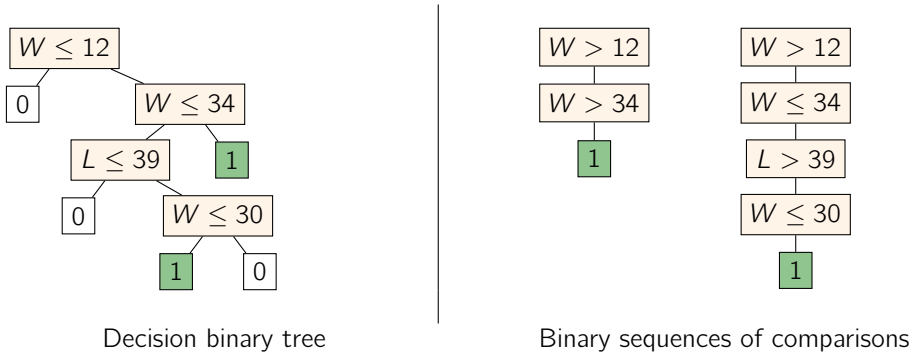


Figure 3.1: Decision tree representation for the normalized Iris database, in the Honest-but-Curious setting.

Now, we define P as the number of paths of which label is 1 and δ as the depth of the deepest path, and for each path i, and each node at depth j of path i, we define as $x_{i,j}$ the value of the feature, that will be compared to a threshold $t_{i,j}$ according to $v_{i,j}$, the orientation of the threshold. More precisely, we have $v_{i,j} = 1$ when the expected result is $x_{i,j} \leq t_{i,j}$ and $v_{i,j} = 0$ otherwise. This allows us to define a model $\mathcal{M}$ as $\mathcal{M} = (P, \delta, \nu, \tau, (t_{i,j}, v_{i,j})_{i \in [P], j \in [\delta]})$.

Then, we use an Additive Homomorphic Encryption $\mathsf{AHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with plaintexts in $\mathbb{Z}_q$, and define an additional algorithm $\mathsf{MultScal}$ such that, given the security parameter 1^κ, with $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{param})$, $\alpha \in \mathbb{Z}_q$ and a ciphertext $c_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1)$, $\mathsf{MultScal}(\mathsf{pk}, \alpha, c_1)$ returns an encryption of $\alpha \cdot m_1$. Finally, for each $i \in \mathfrak{S}_P$, $j \in \mathfrak{S}_\delta$, a ciphertext $C^k_{i,j}$ is computed for each comparison node (i, j) and each possible input value $k \in [2^\nu]$ as follows:

$$C_{i,j}^{k} = \begin{cases} \mathsf{AHE.Enc}(\mathsf{pk}, 1 - v_{i,j}) & = [\![1 - v_{i,j}]\!] \quad \textbf{if } k \leq t_{i,j}; \\ \mathsf{AHE.Enc}(\mathsf{pk}, v_{i,j}) & = [\![v_{i,j}]\!] \quad \textbf{otherwise}. \end{cases}$$

As $v_{i,j} = 1$ when the expected comparison result is $(x_{i,j} \leq t_{i,j})$ and $v_{i,j} = 0$ otherwise, $C_{i,j}^{k}$ is a ciphertext of 0 (respectively a ciphertext of 1) whenever the input value k satisfies (respectively does not satisfy) the comparison. One can see that the number of $C_{i,j}^{k}$ ciphertexts is exponential with respect to ν, but we stress that one only needs a few bits of precision in order to get a meaningful outcome. This number is thus linear in the number of comparisons in practice. The whole process described above is defined as the EncodeModel algorithm, taking the homomorphic encryption public key pk and a model $\mathcal{M}$ as input and returning $\boldsymbol{C} = \{C_{i,j}^{k}\}_{i \in [P], j \in [\delta]}^{k \in [2^{\nu}]}$.

We now describe the remaining of the protocol, step by step.

3.1.2 Evaluation description

Once the client is in possession of an encoded model $\boldsymbol{C}$, he will evaluate a new sample $x = (x_{i,j})_{i \in [P], j \in [\delta]}$, where $x_{i,j}$ is a ν-bits value, by evaluating it path after path. When evaluating the path $i \in [P]$, for each comparison $j \in [\delta]$ of the path, the client sets $[\![S_{i,j}]\!] = C_{i,j}^{x_{i,j}}$. Then he computes the homomorphic sum of the comparisons: $[\![S_i]\!] = \boxplus_j [\![S_{i,j}]\!]$. This evaluation algorithm is denoted by EvaluatePaths which takes as input the public homomorphic key pk, the client input $\boldsymbol{x}$, an encoded model $\boldsymbol{C}$ and returns $\mathbb{S}$, the set of the encryptions $[\![S_i]\!]$ of the score of the i-th path. If all the comparisons of a path are valid, then $[\![S_i]\!]$ is an encryption of 0. Otherwise, if at least one comparison fails, $[\![S_i]\!]$ encrypts a value in $[1 \,..\, \delta]$.

Sending the ciphertexts $\mathbb{S}$ as they are, would lead to some leakage. Indeed, the server would learn the score of each branch. By consequent, the next algorithm consists in the randomization of the scores. In reality, two randomizations are required. Indeed, the scores need to be randomized, and the paths need to be permuted. No permutation would allow the server to learn which path is valid, and which are not, while no randomization of the score, even if paths are permuted, leaks the mean score of the client. Thus, the client first samples $\pi \leftarrow\!\!\$\ \mathfrak{S}_P$ and $(\alpha_i)_{i \in [P]}$, then for $i \in [1 \,..\, P]$, he sets $[\![\widetilde{S}_i]\!] = \mathsf{MultScal}(\mathsf{pk}, \alpha_i, [\![S_{\pi(i)}]\!]) = [\![\alpha_i \cdot S_{\pi(i)}]\!]$. We denote this algorithm by RandomizeScores, taking the public encryption key pk and the ciphertext set $\mathbb{S}$ and returns $\widetilde{\mathbb{S}}$, the set of randomized and permuted ciphertexts.

We depicted the evaluation (without the final randomization) with the example of the iris database in Figure 3.2. Ciphertexts crossed by samples are homomorphically added. Samples crossing only green ciphertexts therefore result in an encryption of 0, while ciphertexts crossing at least one red ciphertext returns a non-null value. Once randomized, encryption of zeros remains encryption of zeros, while encryption of non-zero values become encryption of random values.

Now, encryptions are sent to the server, who evaluates the model. To do this, he decrypts each ciphertext in $\widetilde{\mathbb{S}}$ and counts those decrypting as 0 to retrieve the amount of successful paths S and thus, makes a decision according to a threshold τ. More precisely, if $S > \tau$, the server returns accept and returns reject otherwise. This final algorithm is denoted EncodeModel and takes the secret homomorphic encryption key sk, the set $\widetilde{\mathbb{S}}$ of randomized ciphertexts and the threshold τ, and returns a label in $\{\mathsf{accept}, \mathsf{reject}\}$.

The resulting protocol in the Honest-but-Curious setting is formalized represented in Figure 3.3.

3.1.3 Protocol security

In this section, we first show the correctness and soundness of our protocol. We then provide a proof of both the client and server privacy.

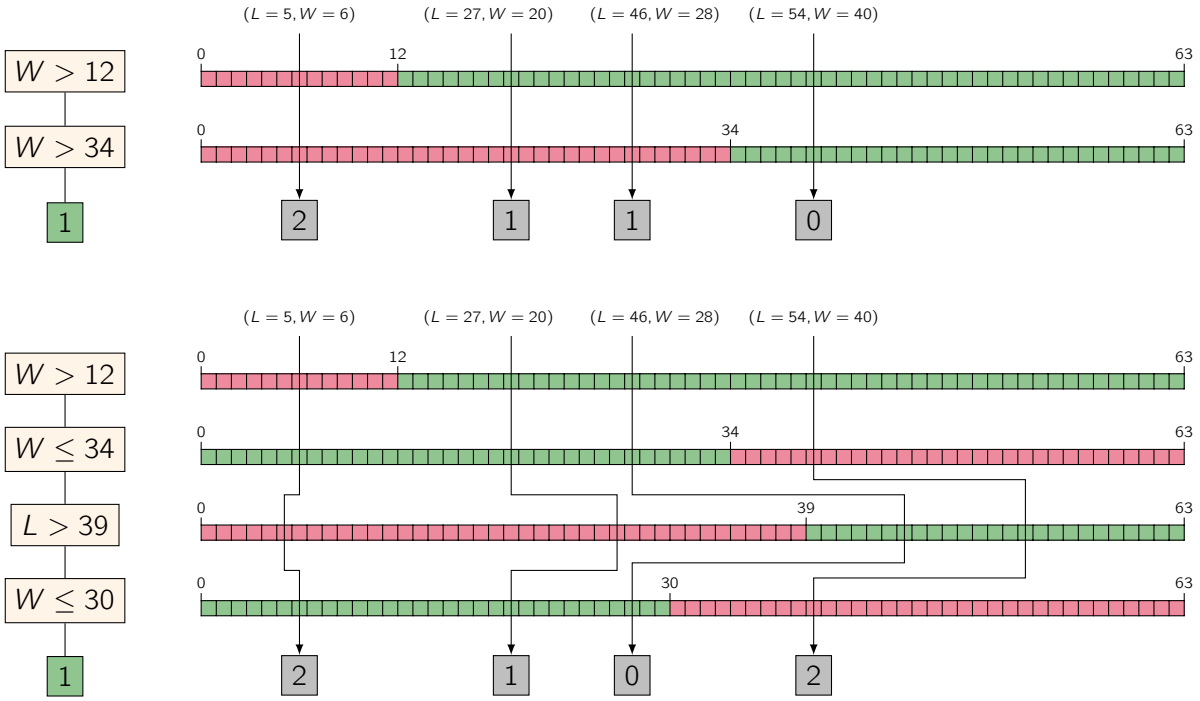


Figure 3.2: Client-side evaluation phase (before randomization) with four normalized samples. From left to right: iris-setosa, iris-versicolor, iris-virginica , iris-virginica. Encryptions of 1 are represented with red boxes whereas encryptions of 0 are represented with green boxes.

Correctness and Soundness. The correctness of our protocol directly follows from the construction of the ciphertexts. If all the comparisons are correct, each $S_{i,j}$ is equal to 0 and the sum S_i equals 0 (correctness). Otherwise, at least one $S_{i,j}$ equals 1 and S_i does not equal 0 (soundness). After the client randomizes the S_i, zeros are still zeros, while other S_i become random values. When decrypting, the server counts the zeros. The threshold τ is applied for the final decision.

Client Privacy. An honest-but-curious server should not learn any information on the client samples, except what it can learn from the outcome, namely the number of successful paths (See Figure 3.3, on the left). We thus consider the following adversary $\mathcal{A}$ against the privacy of the client: $\mathcal{A}$ first chooses a model $\mathcal{M}$ for the server and two sets of possible inputs $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ for the client. It also provides the random tape ρ of the server. The adversary sees the transcript between a server using $\mathcal{M}$ and ρ, and a client using $\boldsymbol{x}_b$ for a random bit b, and it should guess b. There is the natural restriction that $\mathcal{M}(\boldsymbol{x}_0) = \mathcal{M}(\boldsymbol{x}_1)$. The random tape ρ will be used by the server for encoding the model $\mathcal{M}$. In the client-privacy security game (see Figure 3.4), we denote the advantage of any adversary $\mathcal{A}$ by

$$\mathsf{Adv}^{\text{client-privacy}}(\mathcal{A}) = \left|\Pr[\mathsf{Exp}^{\text{client-privacy}-1}(\kappa, \mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\text{client-privacy}-0}(\kappa, \mathcal{A}) = 1]\right|.$$

In our scheme, the client privacy is provided thanks to the permutation and randomization of the encrypted scores. From the expected outcome, one can encrypt the correct number of 0, and the other values are non-zero random values. One can then randomize and permute them. This is indistinguishable from the server point of view. The server then only learns the number of successful paths. But we have to formally prove it.

Game G_0. The first game is the actual security game, with the execution of the protocol between the client with input $\boldsymbol{x}_b$ and the server with input $(\mathcal{M}, \rho)$. We thus detail the Execute procedure:

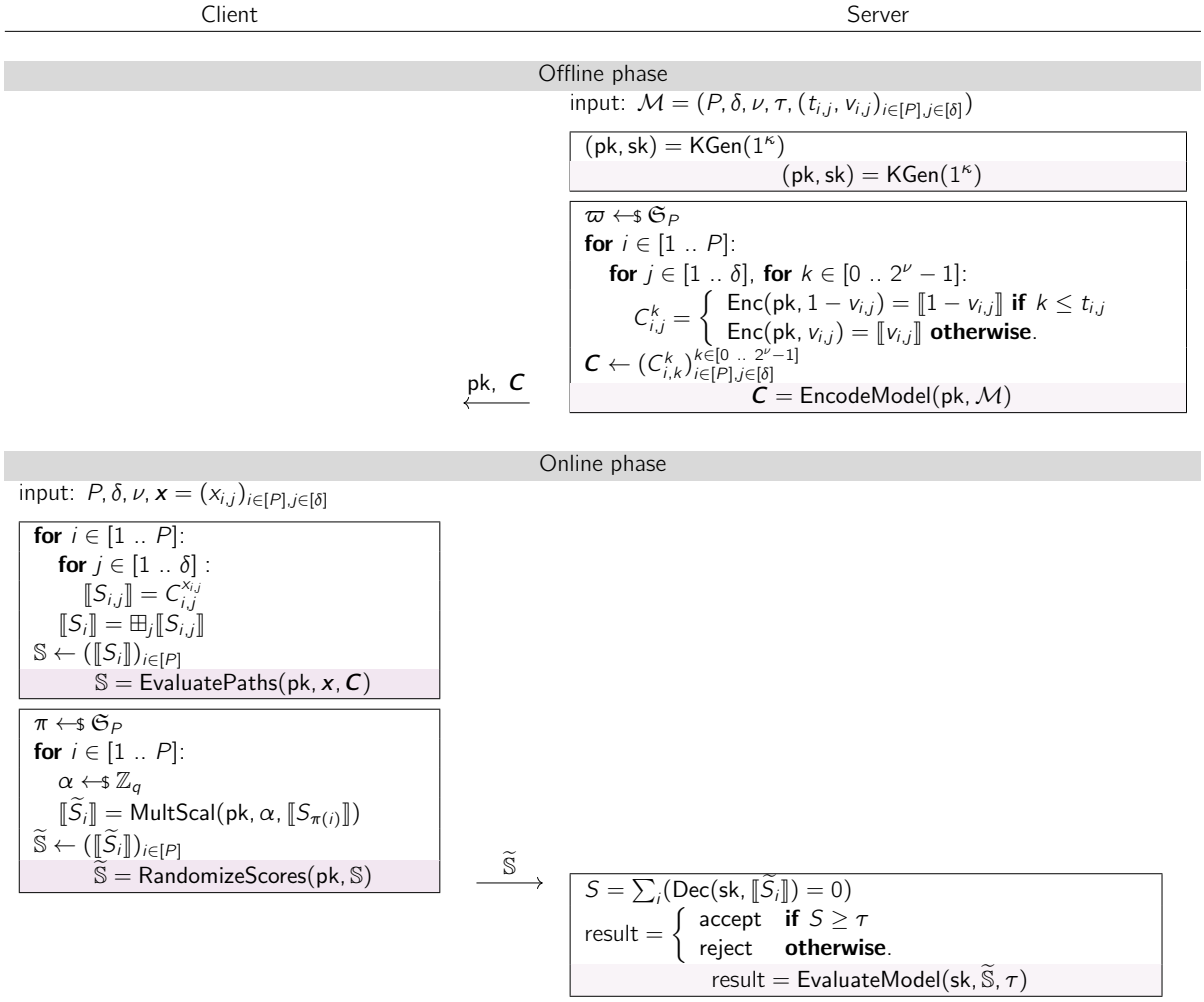


Figure 3.3: Secure decision forest evaluation for Honest-but-Curious Client setting

Execute
1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa; \rho)$
2. $\boldsymbol{C} \leftarrow \mathsf{EncodeModel}(\mathsf{pk}, \mathcal{M}; \rho)$
3. $\mathbb{S} \leftarrow \mathsf{EvaluatePaths}(\mathsf{pk}, \boldsymbol{x}_b, \boldsymbol{C})$
4. $\widetilde{\mathbb{S}} \leftarrow \mathsf{RandomizeScores}(\mathsf{pk}, \mathbb{S})$
5. result $\leftarrow \mathsf{EvaluateModel}(\mathsf{sk}, \widetilde{\mathbb{S}}, \tau)$
6. transcript $\leftarrow (\mathsf{pk}, \boldsymbol{C}, \widetilde{\mathbb{S}})$

Game G_1. To prepare the next game, we introduce RandomizeScores' which is a variant of RandomizeScores only taking the number of successful paths $L = \mathcal{M}(\boldsymbol{x}_0) = \mathcal{M}(\boldsymbol{x}_1)$ as input. It outputs random ciphertexts according to this sole leakage L by generating a permutation of L ciphertexts of 0 and $P - L$ non-zero random ciphertexts:

Experiment $\mathsf{Exp}^{\text{client-privacy}-b}_{\mathcal{A}}(\kappa, \mathcal{A})$
1. $((\boldsymbol{x}_0, \boldsymbol{x}_1), (\mathcal{M}, \rho)) \leftarrow \mathcal{A}.\mathsf{Find}()$
2. $\text{transcript} \leftarrow \mathsf{Execute}((\boldsymbol{x}_b), (\mathcal{M}, \rho))$
3. $b' \leftarrow \mathcal{A}.\mathsf{Guess}(\text{transcript})$
4. **if** $\mathcal{M}(\boldsymbol{x}_0) = \mathcal{M}(\boldsymbol{x}_1)$: **return** $(b' = b)$
5. **else return** $\widetilde{b} \leftarrow\$ \{0, 1\}$

Figure 3.4: Client-Privacy Security Game in the Honest-but-Curious Setting

$\mathsf{RandomizeScores}(\mathsf{pk}, \mathbb{S})$	$\mathsf{RandomizeScores}'(L)$
$\pi \leftarrow\$ \mathfrak{S}_P$	$\pi' \leftarrow\$ \mathfrak{S}_P$
for $i \in [1 \,..\, P]$:	**for** $i \in [1 \,..\, L]$: $[\![\widetilde{S}_{\pi'(i)}]\!] = [\![0]\!]$
$\quad \alpha \leftarrow\$ \mathbb{Z}_q$	**for** $i \in [L+1 \,..\, P]$:
$\quad [\![\widetilde{S}_i]\!] = \mathsf{MultScal}(\mathsf{pk}, \alpha, [\![S_{\pi(i)}]\!])$	$\quad [\![\widetilde{S}_{\pi'(i)}]\!] = [\![k_i \leftarrow\$ \mathbb{Z}_q^*]\!]$
$\widetilde{\mathbb{S}} \leftarrow ([\![\widetilde{S}_i]\!])_{i \in [P]}$	$\widetilde{\mathbb{S}} \leftarrow ([\![\widetilde{S}_i]\!])_{i \in [P]}$

Indeed, instead of sending a permutation of the genuinely randomized encrypted scores of the paths, fresh ciphertexts are generated. Thanks to random multiplications, which keep zeroes but randomize other non-zero plaintexts, and the permutation, this game is perfectly indistinguishable from the previous one. Since the transcript only contains the new $\widetilde{\boldsymbol{S}}$ (which is independent of b), and not $\boldsymbol{S}$, then the advantage in this game is 0. This proves that we have perfect client privacy as $\mathsf{Adv}^{\text{client-privacy}}(\mathcal{A}) = 0$.

Server Privacy. An honest-but-curious client should not learn any information about the model owned by the server, except the feature used in the case where no dummy comparison are used. We do not consider the result as a leak as the server may act differently according to this value. Hence, we consider an adversary that chooses some inputs $\boldsymbol{x}$ for the client and its random tape ρ, but two different models $\mathcal{M}_0 = (P, \delta, \nu, \boldsymbol{t}_0, \boldsymbol{v}_0, \tau_0)$ and $\mathcal{M}_1 = (P, \delta, \nu, \boldsymbol{t}_1, \boldsymbol{v}_1, \tau_1)$, with the constraint that evaluating $\boldsymbol{x}$ with respect to the two models $\mathcal{M}_0$ and $\mathcal{M}_1$ should produce the same result. The random tape ρ will be used by the client for randomizing the ciphertexts. The adversary should then distinguish transcripts involving the two models.

Experiment $\mathsf{Exp}^{\text{server-privacy}-b}_{\mathcal{A}}(\kappa, \mathcal{A})$
1. $((\boldsymbol{x}, \rho), (\mathcal{M}_0, \mathcal{M}_1)) \leftarrow \mathcal{A}.\mathsf{Find}()$
2. $\text{transcript} \leftarrow \mathsf{Execute}((\boldsymbol{x}, \rho), \mathcal{M}_b)$
3. $b' \leftarrow \mathcal{A}.\mathsf{Guess}(\text{transcript})$
4. **if** $\mathcal{M}_0(\boldsymbol{x}) = \mathcal{M}_1(\boldsymbol{x})$: **return** $(b' = b)$
5. **else return** $\widetilde{b} \leftarrow\$ \{0, 1\}$

Figure 3.5: Server-Privacy Security Game in the Honest-but-Curious Setting

We denote the advantage of any adversary $\mathcal{A}$ by

$$\mathsf{Adv}^{\text{server-privacy}}(\mathcal{A}) = \left|\Pr[\mathsf{Exp}^{\text{server-privacy}-1}(\kappa, \mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\text{server-privacy}-0}(\kappa, \mathcal{A}) = 1]\right|.$$

For our scheme, the server's privacy is provided by the encryption of the model in $\boldsymbol{C}$, during the offline phase. The private server's information are the thresholds $t_{i,j}$ and the Boolean values $v_{i,j}$ for each comparison as well as the final threshold τ. One can note that the client learns which

feature is used in a given comparison. This can be avoided by adding dummy comparisons so that each feature (or many features) is used in every path as discussed previously. For the formal proof, as the scheme leaks the number of accepting paths, this is used by the simulator for the final outcome, without needing the decryption key. Then, as the decryption key is not known anymore, using IND-CPA, we can replace all the ciphertexts in the offline phase by encryptions of 0, thus the client cannot learn anything anymore.

Game G_0. The first game is the actual security game, with the execution of the protocol between the client with input $(\boldsymbol{x}, \rho)$ and the server with input $\mathcal{M}_b$. We thus detail the Execute procedure:

Execute
1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa; \rho)$
2. $\boldsymbol{C} \leftarrow \mathsf{EncodeModel}(\mathsf{pk}, \mathcal{M}_b; \rho)$
3. $\mathbb{S} \leftarrow \mathsf{EvaluatePaths}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{C})$
4. $\widetilde{\mathbb{S}} \leftarrow \mathsf{RandomizeScores}(\mathsf{pk}, \mathbb{S})$
5. $\mathsf{result} \leftarrow \mathsf{EvaluateModel}(\mathsf{sk}, \widetilde{\mathbb{S}}, \tau)$
6. $\mathsf{transcript} \leftarrow (\mathsf{pk}, \boldsymbol{C}, \widetilde{\mathbb{S}})$

Game G_1. For the next game, we need to introduce a variant EncodeModel' of EncodeModel, which outputs a list of encryptions of 0:

$\mathsf{EncodeModel}(\mathsf{pk}, \mathcal{M}_b)$	$\mathsf{EncodeModel}'()$

for $i \in [1 \,..\, P]$:
  for $j \in [1 \,..\, \delta]$:
  for $k \in [0 \,..\, 2^\nu - 1]$:
  $C_{i,j}^k = \begin{cases} \mathsf{Enc}(\mathsf{pk}, 1 - v_{i,j}) = [\![1 - v_{i,j}]\!] \textbf{ if } k \leq t_{i,j} \\ \mathsf{Enc}(\mathsf{pk}, v_{i,j}) = [\![v_{i,j}]\!] \textbf{ otherwise}. \end{cases}$
$\boldsymbol{C} \leftarrow (C_{i,j}^k)_{i \in [P], j \in [\delta]}^{k \in [0 \,..\, 2^\nu - 1]}$
	for $i \in [1 \,..\, P]$:
	  for $j \in [1 \,..\, \delta]$:
	  for $k \in [0 \,..\, 2^\nu - 1]$:
	  $C_{i,j}^k = [\![0]\!]$
	$\boldsymbol{C} \leftarrow (C_{i,j}^k)_{i \in [P], j \in [\delta]}^{k \in [0 \,..\, 2^\nu - 1]}$

The difference between the two games involves the indistinguishability of the encryption scheme in $P \times \delta \times 2^\nu$ ciphertexts. With a classical hybrid proof, one can show this is bounded by $P \cdot \delta \cdot 2^\nu \times \mathsf{Adv}_{\mathcal{E}}^{\text{IND-CPA}}(t)$, where t is the running time of the adversary $\mathcal{A}$. Hence, we have

$$\mathsf{Adv}^{\text{server-privacy}}(\mathcal{A}) \leq (P \cdot \delta \cdot 2^\nu) \cdot \mathsf{Adv}_{\mathcal{E}}^{\text{IND-CPA}}(t).$$

This proves that an adversary has negligible advantage when playing the server privacy game.

Those proofs are done in the Honest-but-Curious setting. We now detail how to upgrade our protocol to make it secure against a malicious Client and an Honest-but-Curious Server.

3.2 Malicious client and honest-but-curious server

Unfortunately, a malicious client could trivially bias the outcome of the protocol described in Figure 3.3 by setting all the $[\![\widetilde{S}_i]\!]$ as encryptions of zeros, independently of his input. Hence, the server will always accept the authentication.

In this section, we describe a protocol providing secure decision forests evaluation even if the client behaves maliciously in order to get accepted, while the server is still honest-but-curious. Our security goals remain unchanged from Section 3.1, however the client may now deviate from

the protocol to influence the evaluation outcome. In order to secure our protocol, we add some randomness within the model through the notion of *path polarity*, making path expected output unpredictable. In other words, some paths will be expected to be valid, while other paths will be expected to be rejected, according to the polarity p of the path.

3.2.1 Model encoding

We consider an enriched model $\mathcal{M} = (P, T, \delta, \nu, \tau, (t_{i,j}, v_{i,j}, p_i)_{i\in[P],j\in[\delta]})$, with ternary sequences of comparisons with a polarity p_i. Indeed, to be able to exploit the polarity, a path should have three possible outcomes: accept, reject, ignore. The mapping from the score to the outcome is done according to the leaf label and the polarity of the path. The path always ignore when the score is 0. Then, when the label is 1, the path outcomes accept if the score equals the polarity, reject when they are opposite and vice versa when the label is 0. In figure 3.1, we depict the transition from the classical tree decisoin reprensentation to our reprensetation for encoding, for a malicious client. From a binary tree, such a path is now the path down to the last node of which at least one child node is a decision leaves: on an input $\boldsymbol{x}$, if it does not reach the last node (some comparisons fails before), one outputs 0, otherwise one outputs -1 or +1, whether the leaf is rejecting or accepting. From now, the polarized node of a path denotes the last one.

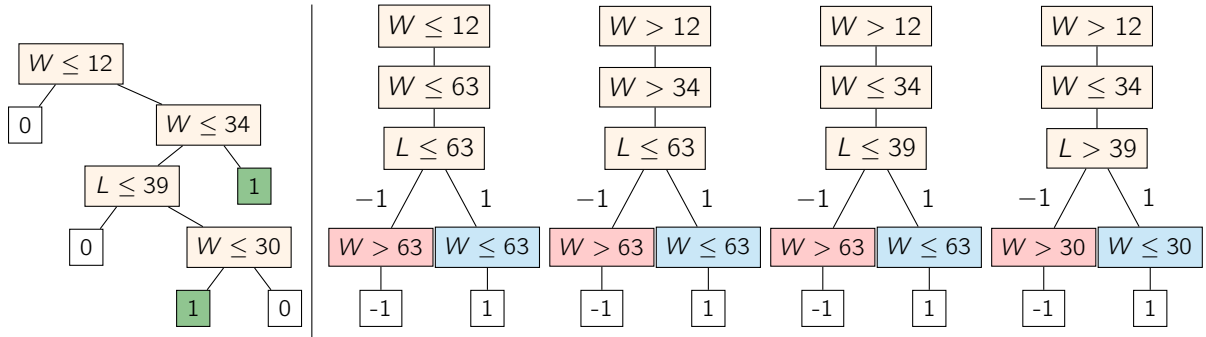


Figure 3.6: Decision tree representation in the malicious setting. The polarized node is in red (resp. blue) when polarity is sampled to -1 (resp 1).

A tree of depth δ has at most $2^{\delta-1}$ such disjoint paths: an input $\boldsymbol{x}$ must be accepted or rejected by exactly one path only, all the other paths should output ignore. Indeed, as in the plain case, a sample can only reach a single leaf, there is a single path such that the score of the path is 0 when the sample reach the final comparison. It is possible to extract such paths from any binary decision tree, by possibly adding some 'always true'/'always false' nodes to make path of length δ. Such 'always true'/'always false' comparisons will also be added to hide the actually used features. This will lead to an impossibility for the malicious client to predict the outcome of a path from the features used in the comparisons: the client does not know if the comparison is really exploited, and the client does not know the polarity of this path and thus whether it should force +1 or -1 to increase the score.

However, even if the client can not predict the expected outcome of the paths, he may still try to guess enough of them to get a score higher than the threshold. But as the polarity is random and hidden (no information leaks, as proven later), the sum of the outputs of a malicious client will follow a binomial distribution with bias 1/2. And the expected sum of the score is 0. If we set the threshold not too low, the probability to get accepted is negligible. An alternative is also to add some 'always accepting' paths, to artificially increase the expected sum of an honest user. With

20 such paths, we can set $\tau = 20$. Let us thus consider 120 paths with a threshold $\tau = 20$ (with either additional 'always accepting' paths, or an initially high threshold): to pass the threshold, one needs 20 correct guesses (probability bounded by $1/10^6$, on exactly 20 non-zero outputs), or at least 70 successes (as if there is at least 70 success, the score passes the threshold) among 120 (probability less than 3%, with random -1/1 outputs). This remains reasonable with respect to usual accuracy of such models. To evaluate a path on an input $\boldsymbol{x}$, with a ternary result, we do not use the same weights in each comparison: the values encrypted in the $C^k_{i,j}$ will still be 1 or 0 for all the active comparisons up to the polarized one, but the encryption logic is inverted ; the polarized node (shown in blue and red on Figure 3.6), will contain δ or 0, where δ is the length of the paths. Hereafter, we use $[\![m]\!]_S$ (respectively $[\![m]\!]_C$) to denote an encryption of m under the public key of the server (respectively, the client) in order to avoid any confusion with the future encryption made with client's public key. The EncodeModel algorithm is thus updated as follows:

for $i \in [1 \,..\, P]$: $\pi \leftarrow \mathfrak{S}_\delta$
  for $k \in [0 \,..\, 2^\nu - 1]$
    for $j \in [1 \,..\, \delta - 1]$:

$$C^k_{i,\pi(j)} = \begin{cases} [\![1 - v_{i,j}]\!]_S \textbf{ if } k \leq t_{i,j} \\ [\![v_{i,j}]\!]_S \textbf{ otherwise.} \end{cases}$$

$$C^k_{i,\pi(\delta)} = \begin{cases} [\![((1 + p_i)/2 - v_{i,\delta} p_i) \cdot \delta]\!]_S \textbf{ if } k \leq t_{i,j} \\ [\![((1 - p_i)/2 + v_{i,\delta} p_i) \cdot \delta]\!]_S \textbf{ otherwise.} \end{cases}$$

$\boldsymbol{C} = \mathsf{EncodeModel}(\mathsf{pk}_S, \mathcal{M})$

3.2.2 Protocol overview

When evaluating, if all the comparisons pass, the sum is $2\delta - 1$, if all but the last comparison pass, the sum is $\delta - 1$. In any other case, the sum is between 0 and $\delta - 2$ or δ and $2\delta - 2$. The idea is to map the score to either $-1, 0$ or 1, according to the mapping depicted in figure 3.7: A score of $2\delta - 1$ will map to $+1$, a score of $\delta - 1$ will map to -1, otherwise, the score will map to 0.

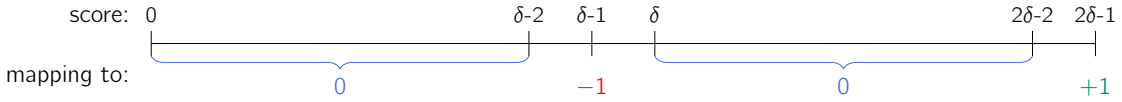


Figure 3.7: Mapping of a path score

One cannot rely on path permutations anymore to enforce client privacy once path polarity is used. Indeed, the server needs to know for which path a score has been computed in order to later involve the correct path polarity p_i. To overcome this issue, we rely on a secure double equality test based on Garbled Circuits along with an Oblivious Transfer, to obliviously convert the above sums between 0 and $2\delta - 1$ into another ciphertext (under the client key) of $+1$, -1 or 0.

First, the client randomizes his scores using a random value α, and we denote by β_i the resulting encrypted value. Thus, if the score of a branch is $\delta - 1$, we have $\beta = \alpha + \delta - 1$, if the score is $2\delta - 1$, we have $\beta = \alpha = 2\delta - 1$. The Garbled Circuits therefore test the equality between $\alpha + \delta - 1$ (client's input) and both β and $\beta - \delta$ (server's inputs). The transition tables are computed as follows: If $\alpha + \delta - 1 = \beta$, the outcome label is an encryption of $+1$ with the client public key ($[\![1]\!]_C$), otherwise, the outcome label is $[\![-1]\!]_C$. If $\alpha + \delta - 1 = \beta - \delta$, the outcome label is -1 ($[\![-1]\!]_C$), otherwise, the outcome label is $[\![+1]\!]_C$. The evaluator then homomorphically sums the two values he obtained and multiplies the result by 2^{-1}. If $\alpha + \delta - 1 = \beta$, the first test returns $[\![1]\!]_C$ and the second one $[\![1]\!]_C$, resulting in a final encryption of $[\![(1 + 1)/2]\!]_C = [\![1]\!]_C$. If $\alpha + \delta - 1 = \beta - \delta$, the first test returns $[\![-1]\!]_C$ and the second one $[\![-1]\!]_C$, resulting in a final encryption of $[\![(-1 + (-1))/2]\!]_C = [\![-1]\!]_C$. In any other case, the first test returns $[\![-1]\!]_C$ and the

second one $[\![1]\!]_C$, resulting in a final encryption of $[\![(-1+1)/2]\!]_C = [\![0]\!]_C$. We use Zero-knowledge Proofs to ensure that one of the label encrypts -1 and the other one encrypts 1.

Once the server computes all the paths scores S_i (that are encrypted with the client public key), he depolarizes them with p_i, adds them to a random value θ resulting in an encryption of $S_\Omega \leftarrow \theta + \Sigma_i p_i \cdot S_i$. Before asking the client to decrypt, we blind the score by using another random value ξ, avoiding him to increase its score. Hence, the encrypted value sent to the client is $S_\Omega \leftarrow \xi \cdot S_\Omega$. The client therefore returns S_Ω, from which the server extracts the global score by computing $S \leftarrow \xi^{-1} S_\Omega - \theta$. If $S \in [\tau, T]$, where α is the acceptation threshold and T is the number of tree (and the maximal score), the server returns **accept**, and **reject** otherwise. A visual representation of the evalution is given in figure 3.8.

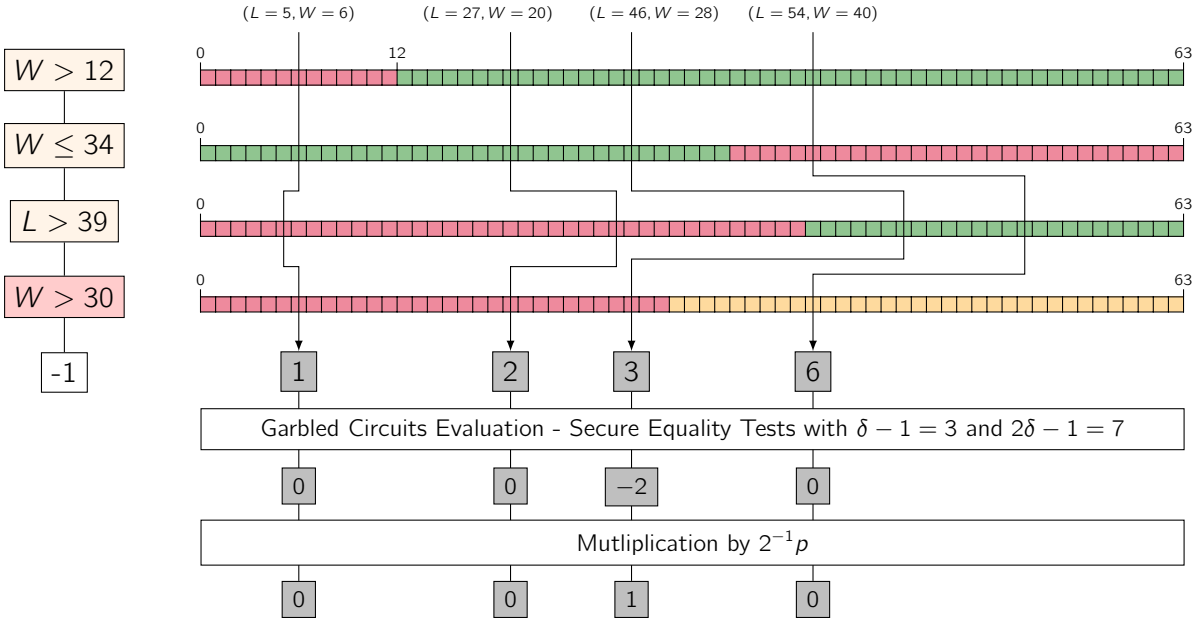


Figure 3.8: Secure Evaluation (without randomization steps) of one path, with polarity $p = -1$, in the malicious client setting. Red boxes are encryptions of 0, green ones are encryptions of 1 and yellow ones are encryptions of $\delta = 4$. Grey boxes denote the path score along the evaluation

3.2.3 Formal description

The client starts by computing the path score $[\![S_i]\!]_S = \boxplus_j [\![S_{i,j}]\!]_S$ as previously and masks it with a random value $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p$ in order to obtain $[\![\beta_i]\!]_S = [\![\alpha_i + S_i]\!]_S$. On the one hand, if $(S_i = \delta - 1)$, we have $(\beta_i = \alpha_i + \delta - 1)$ and on the other hand when $(S_i = 2\delta - 1)$ we have $(\beta_i = \alpha_i + 2\delta - 1)$. By consequent, the client generates a Garbled Circuits testing equality of β_i with $\alpha_i + \delta - 1$ and with $\alpha_i + 2\delta - 1$ by calling InitializeGC:

for $i \in [1 \,..\, P]$:
  $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p$
  $\alpha_{input} \leftarrow (\alpha_i + \delta - 1, \alpha_i + 2\delta - 1)$
  $(\mathfrak{C}_i, \mathfrak{I}^{\mathsf{G}}_{\alpha_i}, \mathfrak{I}^{\mathsf{E}}_i, \mathfrak{T}_i) \leftarrow \mathsf{GC.Generate}(\mathsf{pk}_C, \mathcal{C}_{EQ}, \alpha_{input})$
$(\boldsymbol{\alpha}, \mathfrak{C}, \mathfrak{I}^{\mathsf{G}}_{\boldsymbol{\alpha}}, \mathfrak{I}^{\mathsf{E}}, \mathfrak{T}) = \mathsf{InitializeGC}(\mathsf{pk}_C)$

More precisely, the algorithm InitializeGC takes as input the public key pk_C of the client, and then, for each path, garbles $\mathcal{C}_{EQ}$, a circuit computing equality tests with two inputs from the client, supposedly being $\alpha_i + \delta - 1$ and $\alpha + 2\delta - 1$ and with β_i as server input. The GC.Generate algorithm takes pk_C as extra argument compared to its definition in section 2.8.2. Indeed, the

first equality test labels output encryptions of $+1$ in the positive case and -1 otherwise; while the second ones output encryptions of -1 in the positive case and $+1$ otherwise. Thus, for each Garbled Circuits, the transition table $\mathfrak{T}_i$ is built as follows. As our circuit has two outcomes, $\mathfrak{T}_i$ actually contains two transitions tables $\mathfrak{T}_i^1$ and $\mathfrak{T}_i^2$, each consisting of a pair of encryptions $(\sigma_0, \sigma_1) \in \{([\![-1]\!]_C, [\![+1]\!]_C), ([\![+1]\!]_C, [\![-1]\!]_C)\}$ and a NIZK proof ζ that the pair contains one encryption of -1 and one encryption of 1:

$$\mathfrak{T} = \left(\mathfrak{T}_i^1, \mathfrak{T}_i^2\right)_{i\in[P]} = \left(\left(\left(\mathfrak{O}_{i,0}^1, \sigma_{i,0}^1\right), \left(\mathfrak{O}_{i,1}^1, \sigma_{i,1}^1\right), \zeta_i^1\right) \ ; \ \left(\left(\mathfrak{O}_{i,0}^2, \sigma_{i,0}^2\right), \left(\mathfrak{O}_{i,1}^2, \sigma_{i,1}^2\right), \zeta_i^2\right)\right)_{i\in[P]}$$

Secure Equality Test. We now describe the circuit used. A Garbled Circuits testing the equality between two κ-bit values α and β can be computed by:

$$\begin{aligned}(\alpha = \beta) &= (\alpha[\ell] = \beta[\ell], \forall \ell \in [\kappa]) = (\alpha[\ell] \oplus \beta[\ell] = 0, \forall \ell \in [\kappa]) \\ &= \left(\bigwedge_{\ell=1}^{\kappa} (\overline{\alpha[\ell] \oplus \beta[\ell]}) = 1\right) = \left(\bigwedge_{\ell=1}^{\kappa} (\alpha[\ell] \oplus \overline{\beta}[\ell]) = 1\right).\end{aligned}$$

Each equality test requires κ XOR-gates and $\kappa - 1$ AND-gates. Using the free-XOR and half-gates optimizations, such an equality test can be computed using $2(\kappa - 1)$ ciphertexts. One may compare hashed values of α and β, of shorter length, at the cost of possibly false positive cases. To avoid the computation of the negation of bits of β, the evaluator will simply set the bits of $\overline{\beta}$ as inputs for the incoming Oblivious Transfers. The circuit the client garbles is two parallel equality tests, where equality is tested bitwise.

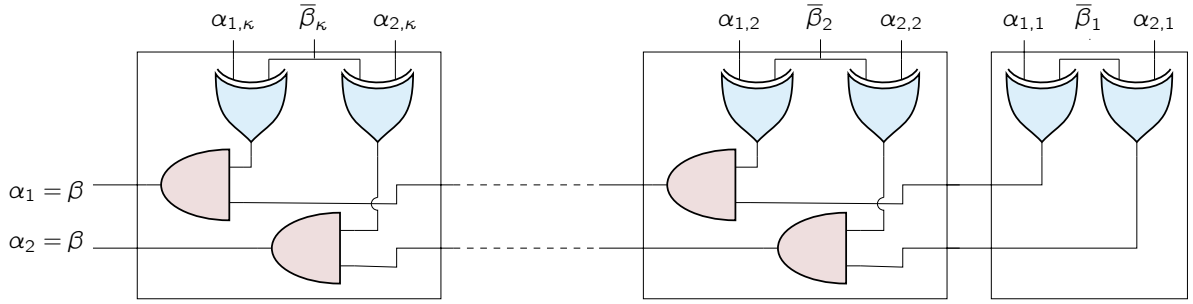


Figure 3.9: Circuit to compute a dual equality test, where β is tested with α_1 and α_2, and inputs are ℓ bits long. XOR gates are depicted in blue while AND gates are maroon.

The algorithm InitializeGC returns $\boldsymbol{\alpha} = (\alpha_i)_{i\in[P]}$, $\mathfrak{C} = (\mathfrak{C}_i)_{i\in[P]}$ the set of garbling of the circuit $\mathcal{C}_{EQ}$, $\mathfrak{I}_{\boldsymbol{\alpha}}^{\mathsf{G}}$, the set of pair of input labels corresponding to the inputs $\boldsymbol{\alpha}_{input}$, $\mathfrak{I}^E$ the labels for the server's inputs and the set of transition tables $\mathfrak{T}$.

After initializating the Garbled Circuits, the client now evaluates the paths with his data $\boldsymbol{x}$ by calling EvaluatePaths. To evaluate a path, the client retrieves the required ciphertexts $C_{i,j}^{x_{i,j}}$ in the encoded model $\boldsymbol{C}$ and sums them using the server public key pk_{S}. Then, he randomizes the score by adding to it the random value $\widetilde{\alpha}_i$ returned by the algorithm InitializeGC. At the end of the evaluation, the client gets a set $\mathfrak{B} = ([\![\beta_i]\!]_{\mathsf{S}})_{i\in[P]}$ of randomized and encrypted scores.

for $i \in [1 \,..\, P]$:
  for $j \in [1 \,..\, \delta] : [\![S_{i,j}]\!]_{\mathsf{S}} = C_{i,j}^{x_{i,j}}$
  $[\![\beta_i]\!]_{\mathsf{S}} = [\![\alpha_i]\!]_{\mathsf{S}} \boxplus \left(\boxplus_j [\![S_{i,j}]\!]_{\mathsf{S}}\right)$
$\mathfrak{B} \leftarrow ([\![\beta_i]\!]_{\mathsf{S}})_{i\in[P]}$
$\mathfrak{B} = \mathsf{EvaluatePaths}(\mathsf{pk}_{\mathsf{S}}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{C})$

The client sends his public encryption key pk_C, the set $\beth$ of encrypted scores, and the Garbled Circuits material. The server first verifies the consistency of the transition table $\mathfrak{T}$ by testing all the underlying NIZK $(\zeta_i^j)_{i\in[P],j\in\{1,2\}}$ and aborts if any verification fails. Otherwise, the server obtains β_i by decrypting $[\![\beta_i]\!]_\mathsf{S}$ and retrieves the corresponding circuit inputs using OTs. Thus, as the Garbled Circuits takes $\overline{\beta_i}$ as input, the server encodes the bits of $\overline{\beta_i}$ for $i \in [P]$ and obtain $\widetilde{\boldsymbol{\beta}}$. As the message space of the AHE scheme is $\mathbb{Z}_q$, we assume the size of β_i is q. The aforementioned process is denoted EncodeOT and is formally described in this way:

$$\begin{array}{l}
\textbf{if } \mathsf{ZK.Vf}(\mathfrak{T}) \neq 1 : \textbf{return } \perp \\
\textbf{for } i \in [1 \,..\, P] : \beta_i = \mathsf{Dec}(\mathsf{sk}_\mathsf{S}, [\![\beta_i]\!]_\mathsf{S}) \\
\quad \textbf{for } k \in [1 \,..\, q] : \widetilde{\beta}_i^k = \mathsf{OT.Encode}(\overline{\beta_i[k]}) \\
\qquad \widetilde{\boldsymbol{\beta}} = \mathsf{EncodeOT}(\mathfrak{T}, \mathsf{sk}_\mathsf{S}, \beth = ([\![\beta_i]\!]_\mathsf{S})_{i\in[P]})
\end{array}$$

In the ElGamal setting, one may use g^{β_i} instead of β_i for the comparisons, which avoids the server to compute a discrete logarithm during the decryption.

After receiving the set of encoded bits $\widetilde{\boldsymbol{\beta}}$ from the sever, the client encodes the server's labels $\mathfrak{I}^\mathsf{E}$ using them, resulting in $\widetilde{\mathfrak{I}}^\mathsf{E}$. Those encoded labels are then sent to the server.

$$\begin{array}{l}
\textbf{for } i \in [1 \,..\, P] \\
\quad \textbf{for } k \in [1 \,..\, q] : \\
\qquad \widetilde{\mathfrak{I}}^\mathsf{E}_{i,k} \leftarrow \mathsf{OT.Compute}(\mathfrak{I}^\mathsf{E}_i, \widetilde{\beta}_i^k) \\
\qquad\qquad \widetilde{\mathfrak{I}}^\mathsf{E} = \mathsf{ComputeOT}(\mathfrak{I}^\mathsf{E}, \widetilde{\boldsymbol{\beta}})
\end{array}$$

Once the server is in possession of its encoded labels, it proceeds to the score computation. The score is first set as an encryption of a random $\theta \in \mathbb{Z}_q$. This is required for the future shared decryption. Then for every path $i \in [P]$, the server decodes its labels, and input them to $\mathfrak{C}_i$ together with the client's input and the transition table. If any of the circuit evaluation fails, the global score S_Ω is set as a random value such that the final output is random, to avoid failure attacks. For every successful circuit, the output is a pair (σ_0, σ_1) of ciphertexts where $\sigma_b \in \{[\![-1]\!]_\mathsf{C}, [\![1]\!]_\mathsf{C}\}$. Then, the server homomorphically computes the polarized score of the path, by computing $\sigma_0 \boxplus \sigma_1$. The server multiplies it by $2^{-1}p_i$, where p_i is the polarity of the branch i, resulting in a ciphertext in $\{[\![-1]\!]_\mathsf{C}, [\![0]\!]_\mathsf{C}, [\![1]\!]_\mathsf{C}\}$, and add the result to $[\![S_\Omega]\!]_\mathsf{C}$. Lastly, the score is randomly blinded using $\xi \leftarrow\!\$\, \mathbb{Z}_p^*$. Hence, $S_\Omega = \xi \cdot (\theta + \Sigma p_i \cdot S_i)$.

$$\begin{array}{l}
\theta \leftarrow\!\$\, \mathbb{Z}_p;\ [\![S_\Omega]\!]_\mathsf{C} \leftarrow [\![\theta]\!]_\mathsf{C} \\
\textbf{for } i \in [1 \,..\, P] : \\
\quad \textbf{for } k \in [1 \,..\, q] : \mathfrak{I}^\mathsf{E}_{\beta_i^k} \leftarrow \mathsf{OT.Decode}(\widetilde{\mathfrak{I}}^\mathsf{E}_{i,k}) \\
\quad \textbf{if } \mathsf{GC.Eval}(\mathfrak{C}_i, \mathfrak{I}^\mathsf{G}_{\alpha_i}, \mathfrak{I}^\mathsf{E}_{\beta_i}, \mathfrak{T}_i) = \perp : \\
\qquad \theta' \leftarrow\!\$\, \mathbb{Z}_p \\
\qquad [\![S_\Omega]\!]_\mathsf{C} \leftarrow [\![\theta']\!]_\mathsf{C} \\
\quad \textbf{else } (\sigma_0, \sigma_1) \leftarrow \mathsf{GC.Eval}(\mathfrak{C}_i, \mathfrak{I}^\mathsf{G}_{\alpha_i}, \mathfrak{I}^\mathsf{E}_{\beta_i}, \mathfrak{T}_i) \\
\quad [\![S_\Omega]\!]_\mathsf{C} \leftarrow [\![S_\Omega]\!]_\mathsf{C} \boxplus (2^{-1}p_i \boxdot (\sigma_0 \boxplus \sigma_1)) \\
\xi \leftarrow\!\$\, \mathbb{Z}_q^* \\
[\![S_\Omega]\!]_\mathsf{C} \leftarrow \xi \boxdot [\![S_\Omega]\!]_\mathsf{C} \\
\qquad ([\![S_\Omega]\!]_\mathsf{C}, \theta, \xi) = \mathsf{ComputeScore}(\mathsf{pk}_\mathsf{C}, \mathcal{M}, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \widetilde{\mathfrak{I}}^\mathsf{E}, \mathfrak{T})
\end{array}$$

Now, the server will ask the client to decrypt this ciphertext. It is risk-free as ξ avoid the client to increase his score, while θ considerably reduces the chances for the adversary to guess ξ. Hence, the slightest modification leads to a random score.

$$\begin{array}{l}
\widetilde{S_\Omega} \leftarrow \mathsf{Dec}(\mathsf{sk}_\mathsf{C}, [\![S_\Omega]\!]_\mathsf{C}) \\
\qquad S_\Omega = \mathsf{DecryptScore}(\mathsf{sk}_\mathsf{C}, [\![S_\Omega]\!]_\mathsf{C})
\end{array}$$

Once the client has decrypted the score S_Ω, he sends it back to the server, who make it back to a value in $[0 \,..\, P]$. First, he removes the blinding factor ξ and subtracts the initial random score θ. Then, the server acts according to the score, the threshold and the number of trees T (as at most one path by tree returns 1).

$S \leftarrow (\xi^{-1} \cdot S_\Omega - \theta)$
if $S \in [\tau, T]$: result $\leftarrow$ accept
else result $\leftarrow$ reject
result = ComputeResult$(\mathcal{M}, \theta, \zeta, \widetilde{S_\Omega})$

Figure 3.10 described the resulting protocol.

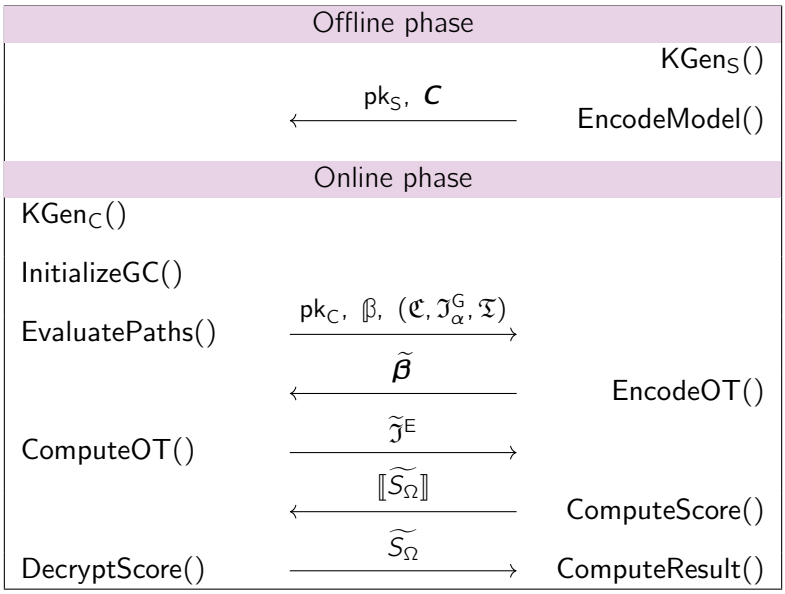


Figure 3.10: Secure Decison Forest Evaluation Protocol with malicious client

3.2.4 Protocol security

Correctness and Soundness. Thanks to the polarity, the server is ensured that the client can no more arbitrarily choose the outcome of a path. Indeed, in contrast to the honest-but-curious case where expected paths values were zero, the expected value sent to the server (the outcome of the Garbled Circuits) will depend on the path polarity and the initial leaf value. With p denoted the path polarity, if the leaf value is 0, the expected output is $-p$, and p if the initial leaf value is 1. If the score is 0, the server will ignore the path. The paths cannot all be ignored, otherwise, there is no chance to be above the threshold τ, hence preventing the two extreme attacks presented before: either the client specifically guesses τ values to be correct, and set all the other to zero (with a success probability bounded by $2^{-\tau}$), or the client tries non-zero values for all the outputs and the success probability follows a binomial distribution with parameters $(P, 1/2)$, where the number of successes must be greater than $(P + \tau)/2$.

The Garbled Circuits will evaluate the initial scores before polarity, and the unknown polarity bit p_i will restore the exact outcome of the path. We show in table 3.11 the correctness of the mapping from a path score to an encryption of either $-1, 0$ or 1. A malicious client has no other choice than a random guess of the polarity to fake the output labels of the Garbled Circuits. He could cheat with a bad encoding of the circuit to bias the output, but only with $+1/-1$ or $-1/+1$ as the output table is proven to contain encryptions of $+1$ and -1 with a zero-knowledge proof. But since the player has no idea about the polarity bit p_i, the final outcome for the path is -1 or

Path Polarity p_i	-1			1		
$(\alpha_i + \delta - 1 = \beta_i)$	0		1	0		1
GC Outcome σ_0	$[\![1]\!]_C$		$[\![-1]\!]_C$	$[\![1]\!]_C$		$[\![-1]\!]_C$
$(\alpha_i + \delta - 1 = \beta_i - \delta)$	0	1	0	0	1	0
GC Outcome σ_1	$[\![-1]\!]_C$	$[\![1]\!]_C$	$[\![-1]\!]_C$	$[\![-1]\!]_C$	$[\![1]\!]_C$	$[\![-1]\!]_C$
$(2^{-1}p_i) \boxdot (\sigma_0 \boxplus \sigma_1)$	$[\![0]\!]_C$	$[\![-1]\!]_C$	$[\![1]\!]_C$	$[\![0]\!]_C$	$[\![1]\!]_C$	$[\![-1]\!]_C$
Expected Output	0	-1	1	0	1	-1

Figure 3.11: Mapping Correctness

$+1$ with identical probability, if positive and negative polarities are balanced. Hence, alteration of the result of a path, without knowing the polarity, will make the sum closer to 0. If the threshold is not too close to 0, the probability for the adversary to impersonate the user is negligible. More generally, the impact of the malicious behavior of the client on false positive outcome will be small, compared to initial accuracy of the system (in clear).

Of course, another cheating strategy can be sending a false zero-knowledge proof or wrong ciphertexts for the Garbled Circuits gates. This would lead to failure attacks with a random value in S_Ω (enforced in the protocol). Similarly, an incorrect decryption of $[\![S_\Omega]\!]_C$ would lead to a random value for $\xi^{-1} \cdot S_\Omega - \theta$, with the detection probability greater than $1 - T/p$, which is overwhelming. This concludes in a reject.

As a consequence, we just have to take care of the accuracy of the model in the clear, for an honest execution, and we will also have to consider the impact of malicious behaviours on the false positive decisions, which is the most critical in the case of continuous authentication. In some other applications, false negative decisions might be more important to limit (such as for spam detection).

Client Privacy. We are still considering the client privacy against an honest-but-curious server. However, in this protocol, we no longer use permutations, because of the path polarity. Nonetheless, from the client privacy of the Oblivious Transfers in the Garbled Circuits, there is no leakage about the α_i's. Then the outcome S_i of the path is encrypted under the client key, which hides it from the server. Eventually, the server only gets the decryption of $\sum_i S_i$, which is the number of accepting trees, the expected result to obtain the classification with confidence score.

The advantage of an adversary in the $\mathsf{Exp}^{\text{client-privacy}-b}(\kappa, \mathcal{A})$ experiment (see Figure 3.4) is defined as

$$\mathsf{Adv}^{\text{client-privacy}}(\mathcal{A}) = \left|\Pr[\mathsf{Exp}^{\text{client-privacy}-1}(\kappa, \mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\text{client-privacy}-0}(\kappa, \mathcal{A}) = 1]\right|.$$

Game G_0. The first game is the actual security game, with the execution of the protocol between the client with input $\boldsymbol{x}_b$ and the server with input $(\mathcal{M}, \rho)$. We thus detail the Execute procedure:

Execute
1. $(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S}) = \mathsf{KGen}(1^\kappa; \rho)$
2. $\boldsymbol{C} \leftarrow \mathsf{EncodeModel}(\mathsf{pk}_\mathsf{S}, \mathcal{M}; \rho)$
3. $(\mathsf{pk}_\mathsf{C}, \mathsf{sk}_\mathsf{C}) = \mathsf{KGen}(1^\kappa)$
4. $(\boldsymbol{\alpha}, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \mathfrak{I}^\mathsf{E}, \mathfrak{T}) \leftarrow \mathsf{InitializeGC}(\mathsf{pk}_\mathsf{C})$
5. $\beta \leftarrow \mathsf{EvaluatePaths}(\mathsf{pk}_\mathsf{S}, x_b, \boldsymbol{\alpha}, \boldsymbol{C})$
6. $\widetilde{\boldsymbol{\beta}} \leftarrow \mathsf{EncodeOT}(\mathfrak{T}, \mathsf{sk}_\mathsf{S}, \beta; \rho)$
7. $\widetilde{\mathfrak{I}^\mathsf{E}} \leftarrow \mathsf{ComputeOT}(\mathfrak{I}^\mathsf{E}, \widetilde{\boldsymbol{\beta}})$
8. $([\![S_\Omega]\!]_\mathsf{C}, \theta, \xi) \leftarrow \mathsf{ComputeScore}(\mathsf{pk}_\mathsf{C}, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \widetilde{\mathfrak{I}}^\mathsf{E}, \mathfrak{T}; \rho)$
9. $S_\Omega \leftarrow \mathsf{DecryptScore}(\mathsf{sk}_\mathsf{C}, [\![S_\Omega]\!]_\mathsf{C})$
10. $\mathsf{result} \leftarrow \mathsf{ComputeResult}(\theta, \xi, S_\Omega)$
11. $\mathsf{transcript} \leftarrow (\mathsf{pk}_\mathsf{S}, \boldsymbol{C}, \mathsf{pk}_\mathsf{C}, \beta, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \mathfrak{T}, \widetilde{\boldsymbol{\beta}}, \widetilde{\mathfrak{I}}^\mathsf{E}, [\![S_\Omega]\!]_\mathsf{C}, S_\Omega, \mathsf{result})$

Game G_1. For the first game, the client simply returns the expected S_Ω to the server instead of genuinely decrypting the received value. This value is indeed known by the simulator who can retrieve S_Ω from the inputs $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ and $\mathcal{M}$. Therefore, the simulator can add the random value θ to S_Ω and multiply the result by ξ (from the random coins) in order to retrieve S_Ω. We recall that the number of successful paths is a tolerated leakage. As previously, such a change avoids the call to the decryption procedure, and thus the need of the decryption key sk_C.

$\mathsf{DecryptScore}(\mathsf{sk}_\mathsf{C}, [\![S_\Omega]\!]_\mathsf{D})$	$\mathsf{DecryptScore}'(S_\Omega)$
$S_\Omega \leftarrow \mathsf{Dec}(\mathsf{sk}_\mathsf{C}, [\![S_\Omega]\!]_\mathsf{C})$ **return** S_Ω	 **return** S_Ω

In this game, $\mathcal{A}$ does not gain any advantage as the output is exactly the same with both functions.

Game G_2. In this game, we introduce InitializeGC' who acts as InitializeGC except that the transition table outputs contains encryptions of random values with simulated NIZKs.

$\mathsf{InitializeGC}(\mathsf{pk}_\mathsf{C})$	$\mathsf{InitializeGC}'(\mathsf{pk}_\mathsf{C})$
for $i \in [1 \,..\, P]$: $\alpha_i \leftarrow\!\!\$\, \mathbb{Z}_p$ $\alpha_{input} \leftarrow (\alpha + \delta - 1, \alpha + 2\delta - 1)$ $(\mathfrak{C}_i, \mathfrak{I}^\mathsf{G}_{\alpha_i}, \mathfrak{I}^\mathsf{E}_i, \mathfrak{T}_i) \leftarrow \mathsf{GC.Generate}(\mathsf{pk}_\mathsf{C}, \mathcal{C}_{EQ}, \alpha_i)$ **return** $(\boldsymbol{\alpha}, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \mathfrak{I}^\mathsf{E}, \mathfrak{T})$	**for** $i \in [1 \,..\, P]$: $\alpha_i \leftarrow\!\!\$\, \mathbb{Z}_p$ $\alpha_{input} \leftarrow (\alpha + \delta - 1, \alpha + 2\delta - 1)$ $(\mathfrak{C}_i, \mathfrak{I}^\mathsf{G}_{\alpha_i}, \mathfrak{I}^\mathsf{E}_i, \mathfrak{T}_i) \leftarrow \mathsf{GC.Generate}(\mathsf{pk}_\mathsf{C}, \mathcal{C}_{EQ}, \alpha_i)$ **for** $j \in \{1, 2\}$ $r_0, r_1 \leftarrow\!\!\$\, \mathbb{Z}_q$ $\widetilde{\mathfrak{T}}^j_i \leftarrow ((\mathfrak{O}^j_{i,0}, [\![r^j_0]\!]_\mathsf{C}), (\mathfrak{O}^j_{i,1}, [\![r^j_1]\!]_\mathsf{C}), \zeta^j_i)$ **return** $(\boldsymbol{\alpha}, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \mathfrak{I}^\mathsf{E}, \widetilde{\mathfrak{T}})$

The advantage of $\mathcal{A}$ in this game relies on the IND-CPA property of the AHE scheme, so this advantage is bounded by $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{E}}(t)$.

Game G_3. For this game, we use $\mathsf{ComputeOT}'$ who takes $\boldsymbol{\beta} = (\beta_i)_{i \in [P]}$ as input and replaces the labels not corresponding to the evaluator's input by a random label.

ComputeOT($\mathfrak{I}^{\mathsf{E}}, \widetilde{\boldsymbol{\beta}}$)	ComputeOT'($\mathfrak{I}^{\mathsf{E}}, \widetilde{\boldsymbol{\beta}}, \boldsymbol{\beta}$)
for $i \in [1 \,..\, P]$:	**for** $i \in [1 \,..\, P]$:
for $k \in [1 \,..\, q]$:	**for** $k \in [1 \,..\, q]$:
$\widetilde{\mathfrak{I}}^{\mathsf{E}}_{i,k} \leftarrow \mathsf{OT.Compute}(\mathfrak{I}^{\mathsf{E}}_i, \widetilde{\beta}^k_i)$	$\widetilde{\mathfrak{I}}^{\mathsf{E}}_{i,k} \leftarrow \mathsf{OT.Compute}(\mathfrak{I}^{\mathsf{E}}_i, \widetilde{\beta}^k_i)$
	$\mathfrak{I}^{\mathsf{E}}_{i,k,1-\beta_i[k]} \leftarrow\!\!\$\ \{0,1\}^\kappa$
return $\widetilde{\mathfrak{I}}^{\mathsf{E}}$	**return** $\widetilde{\mathfrak{I}}^{\mathsf{E}}$

As OT provides Sender Privacy against an honest-but-curious receiver, the advantage of $\mathcal{A}$ is bounded by $\mathsf{Adv}^{\mathsf{inds}}_{\mathsf{OT}}(\mathcal{A})$.

Game G_4. In this game, we introduce EvaluatePaths' which is quite similar to RandomizeScores' in section 3.1.3. As the score is no longer directly computed with the values $[\![S_i]\!]$, we can set all those values to 0.

EvaluatePaths($\mathsf{pk}_{\mathsf{S}}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{C}$)	EvaluatePaths'($\boldsymbol{\alpha}$)
for $i \in [1 \,..\, P]$:	**for** $i \in [1 \,..\, P]$:
for $j \in [1 \,..\, \delta]$: $[\![S_{i,j}]\!] = C_{i,j}^{x_{i,j}}$	$[\![S_i]\!] = [\![0]\!]$
$[\![\beta_i]\!]_{\mathsf{S}} = [\![\alpha_i]\!]_{\mathsf{S}} \boxplus \left(\boxplus_j \left([\![S_{i,j}]\!]_{\mathsf{S}}\right)\right)$	$[\![\beta_i]\!]_{\mathsf{S}} = [\![\alpha_i]\!]_{\mathsf{S}} \boxplus [\![S_{i,j}]\!]_{\mathsf{S}}$
$\beta \leftarrow ([\![\beta_i]\!]_{\mathsf{S}})_{i \in [P]}$	$\beta \leftarrow ([\![\beta_i]\!]_{\mathsf{S}})_{i \in [P]}$
return β	**return** β

When the server decrypts the values obtained when using EvaluatePaths, he retrieves $\alpha_i + \sum_j S_{i,j}$ which is perfectly indistinguishable from any $\alpha' \leftarrow \mathbb{Z}_p$. As a consequence, $\mathcal{A}$ gains no advantage by using EvaluatePaths'.

When using algorithms InitializeGC', EvaluatePaths', ComputeOT' and DecryptScore', the global advantage $\mathsf{Adv}^{\text{client-privacy}}(\mathcal{A})$ of the adversary $\mathcal{A}$ is therefore bounded by $\mathsf{Adv}^{\mathsf{inds}}_{\mathsf{OT}}(\mathcal{A}) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{E}}(t) < \mathsf{negl}(\kappa)$.

Server Privacy. Our main goal is the soundness against a malicious client that would try to get falsely accepted. Moreover, to support that a random guess of the path polarities is the best attack, we need to show that the adversary cannot learn anything that could help him to make a better guess than at random. Eventually, the client does not get back the decision, so if all the received information looks random, we have proven server privacy.

The first messages received by the client are the encryptions of the comparison gates. Under the indistinguishability of the public-key encryption scheme, they do not leak any information about these gates. Then, the server sends encoding for his inputs β_i and $\beta_i - \delta$, which are just keys for the Garbled Circuits. This does not reveal any information about them to the client. Eventually, the client receives the encryption of S_Ω, which is randomized by θ and ξ. This double randomization is to avoid him to cheat and to increase the probability to get accepted. But it also completely hides the real value of $\sum_i S_i$.

As a consequence, the view of the client does not contain any information about the model nor the outcome. The only information the client knows is which feature is used in which comparison, in order to use the appropriate x_i. But again, because of possible dummy comparisons, and the random permutations of the comparisons along a path, the client cannot know which gates are real gates, and which gate is the last critical gate.

This is the main critical security proof, against a malicious client (See Figure 3.12). The latter can thus do anything he wants as its random coins are not given to the simulator anymore. However, as in the honest-but-curious model, server privacy is first provided by the encryption of the model in $\boldsymbol{C}$. But we have to additionally show that no private information leaks during the OT's or the decryption phase. We still consider an adversary that chooses $\boldsymbol{x}$ and $(\mathcal{M}_0, \mathcal{M}_1) =$

$((P, T, \delta, \nu, \boldsymbol{p}_0, \boldsymbol{t}_0, \boldsymbol{v}_0, \tau_0), (P, T, \delta, \nu, \boldsymbol{p}_1, \boldsymbol{t}_1, \boldsymbol{v}_1, \tau_1))$ with the same constraints as in the honest-but-curious case. We stress that the client random coins are not given anymore. We denote the advantage of any adversary $\mathcal{A}$ by:

$$\mathsf{Adv}^{\text{server-privacy}}(\mathcal{A}) = \left|\Pr[\mathsf{Exp}^{\text{server-privacy}-1}(\kappa, \mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\text{server-privacy}-0}(\kappa, \mathcal{A}) = 1]\right|.$$

Experiment $\mathsf{Exp}^{\text{server-privacy}-b}_{\mathcal{A}}(\kappa, \mathcal{A})$	
1.	$((\boldsymbol{x}), (\mathcal{M}_0, \mathcal{M}_1)) \leftarrow \mathcal{A}.\mathsf{Find}()$
2.	$\text{transcript} \leftarrow \mathsf{Execute}((\boldsymbol{x}), \mathcal{M}_b)$
3.	$b' \leftarrow \mathcal{A}.\mathsf{Guess}(\text{transcript})$
4.	**if** $(\mathcal{M}_0(\boldsymbol{x}) \geq \tau_0) = (\mathcal{M}_1(\boldsymbol{x}) \geq \tau_1)$: **return** $(b' = b)$
5.	**else return** $\widetilde{b} \leftarrow\$ \{0, 1\}$

Figure 3.12: Server-Privacy Security Game in the Malicious Setting

Game G_0. The first game is the actual security game, with the execution of the protocol between the client with input $\boldsymbol{x}$ and the server with input $\mathcal{M}_b$. We thus detail the Execute procedure:

Execute	
1.	$(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S}) = \mathsf{KGen}(1^\kappa)$
2.	$\boldsymbol{C} \leftarrow \mathsf{EncodeModel}(\mathsf{pk}_\mathsf{S}, \mathcal{M}_b)$
3.	$(\mathsf{pk}_\mathsf{C}, \mathsf{sk}_\mathsf{C}) = \mathsf{KGen}(1^\kappa)$
4.	$(\boldsymbol{\alpha}, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \mathfrak{I}^\mathsf{E}, \mathfrak{T}) \leftarrow \mathsf{InitializeGC}(\mathsf{pk}_\mathsf{C})$
5.	$\beta \leftarrow \mathsf{EvaluatePaths}(\mathsf{pk}_\mathsf{S}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{C})$
6.	$\widetilde{\boldsymbol{\beta}} \leftarrow \mathsf{EncodeOT}(\mathfrak{T}, \mathsf{sk}_\mathsf{S}, \beta)$
7.	$\widetilde{\mathfrak{I}}^\mathsf{E} \leftarrow \mathsf{ComputeOT}(\mathfrak{I}^\mathsf{E}, \widetilde{\boldsymbol{\beta}})$
8.	$([\![S_\Omega]\!]_\mathsf{C}, \theta, \xi) \leftarrow \mathsf{ComputeScore}(\mathsf{pk}_\mathsf{C}, \mathcal{M}_b, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \widetilde{\mathfrak{I}}^\mathsf{E}, \mathfrak{T})$
9.	$S_\Omega \leftarrow \mathsf{DecryptScore}(\mathsf{sk}_\mathsf{C}, [\![S_\Omega]\!]_\mathsf{C})$
10.	$\text{result} \leftarrow \mathsf{ComputeResult}(\mathcal{M}_b, \theta, \xi, S_\Omega)$
11.	$\text{transcript} \leftarrow (\mathsf{pk}_\mathsf{S}, \boldsymbol{C}, \mathsf{pk}_\mathsf{C}, \beta, \mathfrak{C}, \mathfrak{I}^\mathsf{G}_{\boldsymbol{\alpha}}, \mathfrak{T}, \widetilde{\boldsymbol{\beta}}, \widetilde{\mathfrak{I}}^\mathsf{E}, [\![S_\Omega]\!]_\mathsf{C}, S_\Omega, \text{result})$

Game G_1. In order to prepare the next game, we use a variant ComputeResult' of ComputeResult, which only takes as input the final global outcome $\text{result} = (\mathcal{M}_0(\boldsymbol{x}) \geq \tau_0) = (\mathcal{M}_1(\boldsymbol{x}) \geq \tau_1)$ as well as the larger range $[0, P]$ instead of $[\tau_b, P]$ in order to output the final outcome.

$\mathsf{ComputeResult}(\mathcal{M}_b, \theta, \xi, S_\Omega)$	$\mathsf{ComputeResult}'(\theta, \xi, S_\Omega, \text{result})$
$S_\Omega \leftarrow \xi^{-1} \cdot S_\Omega - \theta$	$S_\Omega \leftarrow \xi^{-1} \cdot S_\Omega - \theta$
if $S_\Omega \in [\tau_b, P]$: result $= 1$	**if** $S_\Omega \in [0, P]$: result $= 1$
else result $= 0$	**else** result $= 0$

In case the adversary cheats during the decryption, because of the random values θ, ξ, the probability this change impacts the view of the adversary is less than τ_b/q, and thus less than P/q, which is negligible.

Game G_2. For this game, instead of genuinely run ComputeScore to compute $[\![S_\Omega]\!]_C$, the latter is set to an encryption of θ by the function ComputeScore'. If $\mathcal{A}$ decrypts both values, he won't be able to distinguish between them as both values are indistinguishable from random values.

ComputeScore$(\mathsf{pk}_C, \mathcal{M}_b, \mathfrak{C}, \mathfrak{I}^G_{\boldsymbol{\alpha}}, \widetilde{\mathfrak{I}}^E, \mathfrak{T})$	ComputeScore'()
$\theta \leftarrow\$ \mathbb{Z}_p$; $[\![S_\Omega]\!]_C \leftarrow [\![\theta]\!]_C$	$\theta \leftarrow\$ \mathbb{Z}_p$; $[\![S_\Omega]\!]_C \leftarrow [\![\theta]\!]_C$
for $i \in [1 \,..\, P]$:	
for $k \in [1 \,..\, q]$: $\mathfrak{I}^E_{\beta_i^k} \leftarrow$ OT.Decode$(\widetilde{\mathfrak{I}}^E_{i,k})$	
if GC.Eval$(\mathfrak{C}_i, \mathfrak{I}^G_{\alpha_i}, \mathfrak{I}^E_{\beta_i}, \mathfrak{T}_i) = \perp$:	
$\theta' \leftarrow\$ \mathbb{Z}_p$	
$[\![S_\Omega]\!]_C \leftarrow [\![\theta']\!]_C$	
else $(\sigma_0, \sigma_1) \leftarrow$ GC.Eval$(\mathfrak{C}_i, \mathfrak{I}^G_{\alpha_i}, \mathfrak{I}^E_{\beta_i}, \mathfrak{T}_i)$	
$[\![S_\Omega]\!]_C \leftarrow [\![S_\Omega]\!]_C \boxplus (2^{-1} p_i \boxdot (\sigma_0 \boxplus \sigma_1))$	
$\xi \leftarrow\$ \mathbb{Z}_p$	$\xi \leftarrow\$ \mathbb{Z}_p$
$[\![S_\Omega]\!]_C \leftarrow \xi \boxdot [\![S_\Omega]\!]_C$	$[\![S_\Omega]\!]_C \leftarrow \xi \boxdot [\![S_\Omega]\!]_C$
return $([\![S_\Omega]\!]_C, \theta, \xi)$	**return** $([\![S_\Omega]\!]_C, \theta, \xi)$

Game G_3. In order to prepare the next game, we use a variant EncodeOT' of EncodeOT, which outputs encodings of random values. Due to Oblivious Transfers privacy properties, this game is indistinguishable from the previous one.

EncodeOT$(\mathsf{sk}_S, [\![\boldsymbol{S}]\!]_S)$	EncodeOT'()
if ZKVf$(\mathfrak{T}) \neq 1$: **return** $\perp$	**if** ZKVf$(\mathfrak{T}) \neq 1$: **return** $\perp$
for $i \in [1 \,..\, P]$: $\beta_i =$ Dec$(\mathsf{sk}_S, [\![S_i]\!]_S)$	**for** $i \in [1 \,..\, P]$:
for $k \in [1 \,..\, q]$:	**for** $k \in [1 \,..\, q]$:
$\widetilde{\beta}_i^k =$ OT.Encode$(\overline{\beta_i[k]})$	$\widetilde{\beta}_i^k =$ OT.Encode$(b \leftarrow\$ \mathbb{Z}_2)$
return $\widetilde{\boldsymbol{\beta}}$	**return** $\widetilde{\boldsymbol{\beta}}$

As OT provides receiver privacy against an honest-but-curious sender, the advantage of $\mathcal{A}$ is bounded by $\mathsf{Adv}^{\mathsf{ind}_R}_{\mathsf{OT}}(\mathcal{A})$

Game G_4. This game is the same as the game G_1 for the server privacy in the honest-but-curious model. We can similarly show that advantage of $\mathcal{A}$ is bounded by $N \cdot \delta \cdot 2^\nu \times \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{E}}(t)$, where t is the running time of the adversary $\mathcal{A}$.

Finally, when using EncodeModel', EncodeOT', ComputeScore' and ComputeResult', the global advantage $\mathsf{Adv}^{\mathsf{server\text{-}privacy}}(\mathcal{A})$ of the adversary $\mathcal{A}$ is bounded by $(N \cdot \delta \cdot 2^\nu) \cdot \mathsf{Adv}^{\mathsf{IND}}_{\mathcal{E}}(t) + \mathsf{Adv}^{\mathsf{ind}_R}_{\mathsf{OT}}(\mathcal{A}) + N/p$.

3.3 Performances and applications

In this section, we detail the storage and communication costs of our protocols, and compare them with related works. We then show the accuracy of those protocols on a well-known database, namely spambase [HRFS99], and an authentication database.

3.3.1 Storage and bandwidth costs

In this section, we describe the storage and bandwidth cost of our protocols. Let $\lambda^R_{\mathsf{OT}} = |\tilde{b}| = |\mathsf{OT.Encode}(b)|$, $\lambda^S_{\mathsf{OT}} = |\mathsf{OT.Compute}(\tilde{b})|$ and $\lambda_{\mathsf{AHE}} = |[\![m]\!]_S| = |[\![m]\!]_C|$.

Offline storage cost. Encoding of the model consists in computing a ciphertext for the 2^ν possible values, for the δ comparisons of the P paths. Thus, the client stores $2^\nu \cdot \delta \cdot P$ ciphertexts, leading to $2^\nu \cdot \delta \cdot P$ bits to store. Table 3.1 presents the storage cost of $\boldsymbol{C}$ with respect to different model parameters. We use lifted ElGamal with Elliptic Curves and q over 256 bits as $\kappa = 128$, thus, $\lambda_{\mathsf{AHE}} = 512$. With inputs over $\nu \in [2, \ldots, 8]$ bit, depth $\delta \in [2, \ldots, 16]$, and P ranging from 10 to 100, the storage is between a few Kilobytes (KB) and a few Megabytes (MB).

P	10			25		
δ \ ν	2	4	8	2	4	8
2	5 KB	20 KB	320 KB	12.5 KB	50 KB	800 KB
4	10 KB	40 KB	640 KB	25 KB	100 KB	1.56 MB
8	20 KB	80 KB	1.28 MB	50 KB	200 KB	3.13 MB
16	40 KB	160 KB	2.56 MB	100 KB	400 KB	6.25 MB

P	50			100		
δ \ ν	2	4	8	2	4	8
2	25 KB	100 KB	1.56 MB	50 KB	200 KB	3.13 MB
4	50 KB	200 KB	3.13 MB	100 KB	400 KB	6.25 MB
8	100 KB	400 KB	6.25 MB	200 KB	800 KB	12.50 MB
16	200 KB	800 KB	12.50 MB	400 KB	1.60 MB	25 MB

Table 3.1: Communication and Storage during the Offline Phase

Online Storage Cost. When honest, the client sends P ciphertexts (one for each path score) to the server. Using the lifted ElGamal AHE, the message sent by the client is of size $P \cdot 512$ bits (see Table 3.2).

When malicious, the client sends P tuples, each formed with a ciphertext of size λ_{AHE} as the path score along with the AND-gate ciphertexts. As seen in Figure 3.9, a double comparison circuit with ℓ bits length inputs consists in $2(\ell - 1)$ AND gates. To reduce communication costs, we use a hash function on the Garbled Circuits inputs to be compared, with output length λ_{GC}. Thus, the client will send $2(\lambda_{\mathsf{GC}} - 1)$ pairs of κ_H-bit hash values (using HalfGates optimization), the $2\lambda_{\mathsf{GC}}$ input labels (where each label has size κ_H) and the transition table consisting in 4 output labels, 4 AHE ciphertexts, and $2\lambda_{\mathsf{ZK}}$-bit length zero knowledge proofs.

This results for each path in the following bit-length:

$$\underbrace{\lambda_{\mathsf{AHE}}}_{|[\![\beta_i]\!]_{\mathsf{C}}|} + \underbrace{2\kappa_H \cdot 2(\lambda_{\mathsf{GC}} - 1)}_{|\mathfrak{C}_i|} + \underbrace{2 \cdot \lambda_{\mathsf{GC}} \cdot \kappa_H}_{|\mathfrak{I}^{\mathsf{G}}_{\alpha_i}|} + \underbrace{4 \cdot (\kappa_H + \lambda_{\mathsf{AHE}}) + 2\lambda_{\mathsf{ZK}}}_{|\mathfrak{T}_i|}$$

which is $5\lambda_{\mathsf{AHE}} + 6\lambda_{\mathsf{GC}} \cdot \kappa_H + 2\lambda\mathsf{ZK}$. With $\kappa_H = 256$, $\lambda_{\mathsf{GC}} = 64$, $\lambda_{\mathsf{AHE}} = 512$ and $\lambda_{\mathsf{ZK}} = 128$, the client first message can be expressed as 101120 bits, i.e. ≈ 12.3 KB per path. During the Oblivious Transfers, the server encodes $P \times |\beta_i| = P \cdot \lambda_{\mathsf{GC}}$ bits. As a consequence, he sends to the client a message of $P \cdot \lambda_{\mathsf{GC}} \cdot \lambda^{\mathsf{R}}_{\mathsf{OT}}$ bits. Using classical ElGamal as PKE in the Oblivious Transfer, one has $\lambda^{\mathsf{R}}_{\mathsf{OT}} = 512$ and $\lambda^{\mathsf{S}}_{\mathsf{OT}} = 1024$ resulting in a 4 KB long message for each path. The client responds with a $P \cdot \lambda_{\mathsf{GC}} \cdot \lambda^{\mathsf{S}}_{\mathsf{OT}}$ bit long message corresponding to 8 KB per path. Then, the server sends the encrypted randomized score to the client who returns the plaintext value he retrieves when decrypting. Table 3.2 shows the total bandwidth cost for each party in both protocols, according to the number P of paths.

Table 3.2: Communications During the Online Phase.

P	50	100	150	200
Honest-but-Curious				
Client	3.13 KB	6.25 KB	9.38 KB	12.5 KB
Server	0 bit			
Malicious				
Client	1 MB	2 MB	3 MB	4 MB
Server	200 KB	400 KB	600KB	800KB

3.3.2 Comparison with related works

Several approaches have been proposed in order to securely evaluate decision trees [BPSW07, BFK+09, BPTG15, WFNL16, TMZC17, DDH+16, TKK19]. However, most of these constructions are either only secure in the honest-but-curious setting or tailored for decision tree evaluation rather than decision forest evaluation. As such, except [WFNL16] and [TMZC17] which we consider hereafter, the aforementioned works cannot be compared to our protocol meaningfully. Similarly to our protocol, both [WFNL16] and [TMZC17] rely on Additive Homomorphic Encryption (AHE) and Oblivious Transfers (OT). In addition, our protocol also uses Garbled Circuits (GC) in the malicious setting.

A major difference between our protocol and existing constructions is that we rely on the server sending its encrypted model to the client rather than the client sending its encrypted data to the server. This strongly impacts the design of our protocol and allows us to introduce a preprocessing phase that can be performed offline and leveraged later during the online phrase. This is advantageous for several use-cases such as the continuous authentication one as it offers a trade-off between the number of rounds required to execute the protocol and the required bandwidth.

All existing constructions leak some information with respect to the model structure (*i.e.*, on the server side) whether it be its total number of nodes M, the total number of comparison nodes m or the maximal depth δ of the trees. In our protocol, the client may also learn which features will be used to evaluate the trees. We consider this as a privacy leak, but as our protocols will have a complexity independent of the depth of the trees (or more precisely, the length of the paths down to the leaves), we will be able to add dummy comparisons with dummy features, which will completely hide which are the actually used features. We thus propose two variants of our protocols in Table 3.3, without (version 1) or with dummy comparisons (version 2). They only differ during the offline step as the number of features in the comparisons impacts the communication and the storage: δ is the number of real features per path, whereas χ is the number of real and dummy features. We stress that this modification does not impact the online step of our protocols: communication only depends on the number P of paths, and not their length. This thus allows us to use as many dummy features as we want to hide the trees and the forest structure. This will be a crucial property for the privacy of the model. In addition, our constructions also feature some leakage on the client side. When used to evaluate decision forests, our protocol leaks the number of successful paths within the forest which constitutes a tolerated leakage in our targeted application. Indeed, it corresponds to the number of accepting trees which allows to compute a confidence level associated to the result.

One can note half rounds in our constructions, which mean one-way flows, from the sender to the recipient. Indeed, most of our schemes are actually non-interactive. Since our goal is to provide the result to the server, in the malicious setting, we get a 5-flow protocol.

Scheme	Rounds	Tools	Bandwidth
Honest-but-Curious model			
[WFNL16]	6	AHE+OT	$\mathcal{O}(m)$
[TMZC17]	4	AHE+OT	$\mathcal{O}(m)$
Section 3.1 (Offline) – 1	0.5	AHE	$\mathcal{O}(m \cdot \delta \cdot 2^{\nu})$
Section 3.1 (Offline) – 2			$\mathcal{O}(m \cdot \chi \cdot 2^{\nu})$
Section 3.1 (Online)	0.5		$\mathcal{O}(P)$
Malicious model			
[WFNL16]	2	AHE+OT	$\mathcal{O}(M)$
[TMZC17]	4	AHE+OT	$\mathcal{O}(m)$
Section 3.2 (Offline) – 1	0.5	AHE	$\mathcal{O}(m \cdot \delta \cdot 2^{\nu})$
Section 3.2 (Offline) – 2			$\mathcal{O}(m \cdot \chi \cdot 2^{\nu})$
Section 3.2 (Online)	2.5	AHE+GC+OT	$\mathcal{O}(P)$

Table 3.3: Comparison to state-of-the-art. Protocol 1 offers a weaker feature protection compared to Protocol 2 that hides features using dummy comparisons.

3.4 Application to continuous authentication and spam filtering

We implemented our solution and run our tests in Python with the Scikit-learn library [PVG+11], using 75% of the dataset as the training set and the remaining 25% as the testing set. We optimized the training using the Orthogonal Matching Pursuit Algorithm [MZ93]: we generate 100 times more trees than expected, and we then apply the OMP algorithm on the global set, such that the outcome is the best linear combination with the expected number of trees.

We first deal with our initial use case, namely the continuous authentication. We use a database of 20531 samples split in 35 profiles built with 222 features. In this context, low *False Positive Rate* (FPR) is privileged to low *False Negative Rate* (FNR), since it is preferable to ask the client to use a second authentication factor rather than being impersonated. Moreover, high accuracy for each test is not required, as multiple tests will amplify the quality. Table 3.4 shows the mean results on the 35 profiles (where, for a given profile, all other profiles are considered as imposter), depending on the number of paths P and the depth of the model δ, while ν is set to 6, leading to 64 ciphertexts for each comparison, stored by the client. Also, we consider several values for the acceptation threshold (τ) (which equals 50% by default, for the simple majority).

We compute the FPR and FNR, then the accuracy is defined as the F1-score (defined by $(1 - \mathsf{FPR})/(1 + (\mathsf{FNR} - \mathsf{FPR})/2)$). Random decision would lead to an accuracy of $1/2$, and perfect filter should have F1-score equal to 1. We determine the best accuracy depending on those parameters in Table 3.4. For the honest-but-curious security setting, there is no constraint on the threshold, while against malicious clients, the higher the threshold is, the higher the security level is against active impersonation attempts.

Secondly, we consider on the spambase database [HRFS99] (with 4601 samples $\times$ 57 features) which determines if an email should be considered as spam or not. Results are shown in Table 3.5.

τ		50%			55%		
T	δ	FPR	FNR	F1 Score	FPR	FNR	F1 Score
10	6	0.02	0.15	0.92	0.02	0.16	0.91
	8	0.01	0.19	0.91	0.01	0.19	0.91
25	6	0.04	0.13	0.92	0.03	0.15	0.92
	8	0.02	0.14	0.92	0.02	0.17	0.91

τ		60%			65%		
T	δ	FPR	FNR	F1 Score	FPR	FNR	F1 Score
10	6	0.01	0.23	0.89	0.01	0.23	0.89
	8	0.01	0.25	0.88	0.01	0.26	0.88
25	6	0.02	0.2	0.9	0.01	0.23	0.89
	8	0.01	0.24	0.89	0.01	0.26	0.88

Table 3.4: Accuracy on our continuous authentication Database

τ		50%			55%		
T	δ	FPR	FNR	F1 Score	FPR	FNR	F1 Score
10	2	0.02	0.24	0.88	0.01	0.28	0.87
	4	0.03	0.20	0.89	0.04	0.18	0.90
25	2	0.03	0.24	0.88	0.02	0.25	0.88
	4	0.04	0.19	0.89	0.04	0.21	0.88

τ		60%			65%		
T	δ	FPR	FNR	F1 Score	FPR	FNR	F1 Score
10	2	0.02	0.24	0.88	0.01	0.45	0.81
	4	0.02	0.27	0.87	0.01	0.30	0.86
25	2	0.01	0.47	0.80	0.00	0.56	0.78
	4	0.02	0.23	0.89	0.01	0.34	0.85

Table 3.5: Accuracy on the spambase Database

Chapter 4

Post-quantum Oblivious Transfer from Smooth Projective Hash Functions with gray zone

Smooth Projective Hash Functions (SPHF), or Hash Proof System as introduced by Cramer and Shoup in [CS02], are a cryptographic primitive initially designed to provide IND-CCA encryption schemes. Over the years, SPHFs have been used for many applications such as Password-Authenticated Key Exchange [GL03, ACP09, KV11, BBC+13a], Zero-Knowledge Proofs [JR12, BBC+13b] or Witness Encryption [DS15]. Since their introduction, SPHFs have been developed over classical hard problems such as discrete logarithm or factorization. However, post-quantum cryptography does not seem to be as easily compliant with SPHF. In [KV09], Katz *et al.* introduced *Approximate Smooth Projective Hash Functions*. The correctness property of an SPHF claims that the hash value and the projective hash value are required to be equal on words in an NP-language, when knowing a witness, while the smoothness property expects them to be independent when no witness exists. Approximate SPHF uses an approximate correctness, that allows those values to be close, relatively to a given distance. Furthermore, languages relying on code-based or lattice-based ciphertexts result in a gap between the set of valid ciphertexts of a given value μ, and the values that decrypt into μ. As mentioned in [BBB+21a], an adversary could maliciously generate one of those ciphertexts and open the door for practical attacks. The presence of this gap can also be problematic when expecting to work in the Universal Composability framework [Can01]. In this chapter, we define a new primitive named Smooth Projective Hash Function with Gray Zone (SPHFwGZ) which can be seen as a relaxation of the classical Smooth Projective Hash Functions, with a subset of the words for which one cannot claim correctness nor smoothness. We then rely on this new primitive to build Oblivious Transfer (OT) protocols. Interestingly, our new paradigm features a security analysis in the Universal Composability (UC) framework and may be instantiated from post-quantum primitives. We then provide two instantiations based on the Diffie-Hellman and the Learning With Errors (LWE) problems.

★ Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Post-Quantum and UC-secure Oblivious Transfer from SPHF with Grey Zone. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022.[1]

[1] in this thesis, we use 'gray' rather than 'grey', as it is written is American English

Contents

Notations & Acronyms

Smooth Projective Hash Functions	
SPHF	Smooth Projective Hash Function(s)
SPHFwGZ	Smooth Projective Hash Function(s) with Gray Zone
$\mathcal{X}$	Space of words
$\mathcal{L}, \mathcal{L}'$	Languages
x	Word
w	Witness of a word in a language
hk	Hash key
hp	Projection key
H_{hk}	Hash value
H_{hp}	Projection hash
$\mathfrak{R}_{\mathcal{L}}$	Witness Relation function for the language $\mathcal{L}$

Oblivious Transfer	
OT	Oblivious Transfer
$\Phi_{\mathsf{OT}}^{\mathsf{SPHFwGZ}}$	our OT protocol based on SPHFwGZ

Security Proof	
UC	Universal Composability
CRS	Common Reference String
$\mathcal{S}_0, \mathcal{S}_1$	Setup sets for σ
$\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}'_1$	Setup sets for ρ
$\mathsf{td}_\sigma, td_\rho$	Trapdoors
$\mathcal{F}_{\mathsf{CRS}}$	Ideal Functionality for a CRS
$\mathcal{F}_{\mathsf{OT}}$	Ideal Functionality for an Oblivious Transfer

Previous Works. In code-based cryptography, the first proposition was made by Persichetti in [Per13]. The SPHF proposed there uses a weaker smoothness definition, called universality. Strictly speaking, this is not a drawback as we can transform an SPHF with universality property to a word-dependent SPHF with smoothness property. However, the main issue with this candidate is that the proof is done on random keys, rather than the whole keys. This has for consequence that an adversary can exploit some well-chosen keys resulting in a failure of the proof. A second construction was designed in [BBB+21a]. As said before, when working with lattices and codes, languages based on ciphertexts present a Gray Zone. In this work, Bettaieb *et al.* withdraw this gap using a zero-knowledge proof asserting if two different ciphertexts of the same message are valid, reducing the SPHF on the set of valid ciphertexts, resulting in the first *gapless* post-quantum SPHF. A solution based on codes is also given in [SA21], but their solution offers an Approximate SPHF with computational smoothness, while real SPHF expects statistical/perfect smoothness. In lattice-based cryptography, the first construction was given in [KV09] where Katz *et al.* introduced the notion of Approximate SPHF. Their language not being exactly defined as the valid LWE-ciphertexts, decoding procedure was expensive, as detailed in [BBDQ18]. This latter article, motivated by this issue, offers the first non-approximated SPHF based on lattices. While the two previous constructions are in the standard model, Zha *et al.* [ZY17] propose a SPHF requiring access to a random oracle. Indeed, their language relies on simulation-sound non-interactive zero-knowledge proofs, that we are not able to construct efficiently without random oracles.

4.1 Smooth Projective Hash Functions with gray zone

We first give a definition of our formalization of Smooth Projective Hash Functions with Gray Zone (SPHFwGZ) which is a relaxation of the classical SPHF in which one cannot claim correctness nor smoothness for a subset of the words. Later, we provide a quantum-resistant SPHFwGZ based on lattices. With this new definition, we have two disjoint languages $\mathcal{L}, \mathcal{L}' \subset \mathcal{X}$ that does not necessarily partition the superset $\mathcal{X}$: the remaining subset $\mathcal{X} \setminus (\mathcal{L} \cup \mathcal{L}')$ is defined as the Gray Zone.

Definition 4.1 - Smooth Projective Hash Functon with Gray Zone

An SPHFwGZ is defined with a tuple of algorithms:

- $\mathsf{Setup}(1^\kappa)$ generates the parameters param from κ, the security parameter, or an explicit random tape (σ, ρ) in $\mathcal{S}_0 \times \mathcal{R}_0$. param includes a description of $\mathcal{L}, \mathcal{L}', \mathcal{X}$, where $(\mathcal{L} \cup \mathcal{L}') \subset \mathcal{X}$ and $\mathcal{L} \cap \mathcal{L}' = \emptyset$, and $\mathcal{L}$ is a language hard to decide in $\mathcal{X}$. Returns param.
- $\mathsf{HashKG}(\mathsf{param})$ generates a random hash key hk. Returns hk.
- $\mathsf{ProjKG}(\mathsf{hk}, x)$ derives the projection key hp (it may need x as input). Returs hp.
- $\mathsf{Hash}(\mathsf{hk}, x)$ computes the hash value $H_{\mathsf{hk}} \in \mathcal{V}$, where $\mathcal{V}$ is the set of hash values, associated to the word x. Returns H_{hk}.
- $\mathsf{ProjHash}(\mathsf{hp}, x, w)$ computes $H_{\mathsf{hp}} \in \mathcal{V}$ using a witness w linked to the word x. Returns H_{hp}.

As the classical SPHF, our SPHFwGZ verifies the following statistical properties, for any setup execution that provides param, defining $\mathcal{L}, \mathcal{L}', \mathcal{X}$:

- **Correctness:** For any $x \in \mathcal{L}$, $H_{\mathsf{hk}} = H_{\mathsf{hp}}$, where $\mathsf{hk} \leftarrow \mathsf{HashKG}(\mathsf{param})$, $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, x)$, $H_{\mathsf{hk}} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$, and $H_{\mathsf{hp}} \leftarrow \mathsf{ProjHash}(\mathsf{hp}, x, w)$ for the witness w of $x \in \mathcal{L}$;
- **Smoothness:** For any $x \in \mathcal{L}'$, the distributions of $(\mathsf{hp}, H_{\mathsf{hk}})$ and (hp, v) are indistinguishable, where $\mathsf{hk} \leftarrow \mathsf{HashKG}(\mathsf{param})$, $\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, x)$, $H_{\mathsf{hk}} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$, and $v \leftarrow\!\!\$\ \mathcal{V}$;

The algorithms and properties described above are the basic algorithms for SPHFwGZ. For a later use, we need to define several additional properties.

4.1.1 Word indistinguishability and trapdoor

First, we assume languages $\mathcal{L}$ and $\mathcal{L}'$ in $\mathcal{X}$ are defined according to a random tape (σ, ρ) sampled in a set $\mathcal{S}_0 \times \mathcal{R}_0$ (i.e. from $\mathsf{param} \leftarrow \mathsf{Setup}(\sigma, \rho)$). The samplable set $\mathcal{S}_0$ is defined together with its twin set $\mathcal{S}_1$ such that when $\sigma \in \mathcal{S}_1$, and $\mathsf{param} \leftarrow \mathsf{Setup}(\sigma, \rho)$, there exists a trapdoor td_σ that allows to test if a given word $x \in \mathcal{X}$ is in $\mathcal{L}'$ or not. We then also need the following algorithms:

- $\mathsf{WordGen}\mathcal{L}(\mathsf{param})$: Samples and returns $x \leftarrow\!\!\$\ \mathcal{L}$, together with its witness w;
- $\mathsf{WordTest}(\mathsf{td}_\sigma, x)$, using the trapdoor td_σ, tests if $x \in \mathcal{L}'$.

As we assumed $\mathcal{L}$ to be a hard subset of $\mathcal{X}$ when $\sigma \in \mathcal{S}_0$, we have the **Word-Indistinguishability Property**: An adversary can not distinguish between random words in $\mathcal{L}$ and random words in $\mathcal{X}$, for any $\sigma \in \mathcal{S}_0$, with more than a negligible advantage.

The string σ can be seen as a CRS, that admits a trapdoor when sampled from $\mathcal{S}_1$. The normal use is with $\sigma \leftarrow\!\!\$\ \mathcal{S}_0$, which needs to be efficiently samplable. When $\sigma \in \mathcal{S}_1$, the trapdoor td_σ must be easy to compute from σ.

4.1.2 Decomposition intractability and trapdoor

We also define the alternate sets $\mathcal{R}_1$ and $\mathcal{R}'_1$ for $\mathcal{R}_0$. During normal use, ρ is sampled from $\mathcal{R}_0$, which needs to be efficiently samplable. When $\rho \in \mathcal{R}_1$, and $\mathsf{param} \leftarrow \mathsf{Setup}(\sigma, \rho)$, there exists a trapdoor $\mathsf{td}_\rho = (x, x', w, w')$, that must be easy to compute from ρ. When $\rho \in \mathcal{R}'_1$, and $\mathsf{param} \leftarrow \mathsf{Setup}(\sigma, \rho)$, there exists a trapdoor $\mathsf{td}_\rho = (x, x')$, that must be easy to compute from ρ. Let us define the complement algorithm, for any $\rho \in \mathcal{R}_0 \cup \mathcal{R}_1 \cup \mathcal{R}'_1$:

- $\mathsf{ComplementWord}(\mathsf{param}, \rho, x)$: from any word $x \in \mathcal{X}$, it outputs x';

From this complement algorithm, we expect the following statistical property, for any $\sigma \in \mathcal{S}_0 \cup \mathcal{S}_1$ but ρ only in $\mathcal{R}_0$:

- **Complement:** for any $x \in \mathcal{X}$, if $x' \leftarrow \mathsf{ComplementWord}(\mathsf{param}, \rho, x)$, we have $x = \mathsf{ComplementWord}(\mathsf{param}, \rho, x')$;

In addition to those algorithms, we also need a computational assumption: the **Decomposition Intractability**. This assumption states that, given random $(\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_1 \times \mathcal{R}_0$, no adversary can generate, with non-negligible probability, $x, y \notin \mathcal{L}'$ such that $y = \mathsf{ComplementWord}(\mathsf{param}, \rho, x)$, and so even with the knowledge of the trapdoor td_σ.

On the other hand, when $\rho \in \mathcal{R}_1$, the trapdoor $\mathsf{td}_\rho = (x, x', w, w')$ satisfies x and x' are uniformly random in $\mathcal{L}$ with witnesses w, w', and $x' = \mathsf{ComplementWord}(\mathsf{param}, \rho, x)$. And when $\rho \in \mathcal{R}'_1$, the trapdoor $\mathsf{td}_\rho = (x, x')$ satisfies x and x' are uniformly random in $\mathcal{L}'$, and $x' = \mathsf{ComplementWord}(\mathsf{param}, \rho, x)$.

Again, the string ρ can be seen as a CRS, that admits a trapdoor when sampled from $\mathcal{R}_1$ or $\mathcal{R}'_1$. The normal use is with $\rho \leftarrow\!\!\$\ \mathcal{R}_0$, which needs to be efficiently samplable. When $\rho \in \mathcal{R}_1$ or $\rho \in \mathcal{R}'_1$, the trapdoor td_ρ must be easy to compute from ρ.

Eventually, for the security proof to go through, we make use of the **CRS Indistinguishability**: An adversary can not distinguish between $\mathcal{R}_0$, $\mathcal{R}_1$ and $\mathcal{R}'_1$, and between $\mathcal{S}_0$ and $\mathcal{S}_1$, with more than a negligible advantage.

Note that we independently consider the choices between $\mathcal{S}_0$ and $\mathcal{S}_1$ and between $\mathcal{R}_0$, $\mathcal{R}_1$ and $\mathcal{R}'_1$, but the latter choice could depend on the former choice. So the global CRS is the pair $\mathsf{crs} = (\sigma, \rho)$.

4.2 Oblivious Transfer from SPHFwGZ

In this section we first present our construction of Oblivious Transfers based on Smooth Projective Hash Functions with Gray Zone, and then provide a security proof of our Oblivious Transfer in the Universal Composability framework

4.2.1 Construction of Oblivious Transfer

Our Oblivious Transfer uses a crs $= (\sigma, \rho) \in \mathcal{S}_0 \times \mathcal{R}_0$ as defined above, where we assume $\mathcal{S}_0 \times \mathcal{R}_0 \approx \mathcal{S}_1 \times \mathcal{R}_0 \approx \mathcal{S}_0 \times \mathcal{R}_1 \approx \mathcal{S}_0 \times \mathcal{R}'_1$. The idea of our Oblivious Transfer is quite straightforward. The receiver generates a word x_b for the sender's input she wants to retrieve, and extract a second word x_{1-b} using ρ and x_b. Then, the sender computes a One-Time Pad encryption of his messages m_b and m_{1-b} with the hash values of x_b and x_{1-b} respectively.

We depict in Figure 4.1 the OT protocol $\Phi_{\mathsf{OT}}^{\mathsf{SPHFwGZ}}$.

Receiver		Sender
with input $b \in \{0,1\}$		with input $m_0, m_1 \in \mathcal{M}$
for sid and crs $= (\sigma, \rho)$ from $\mathcal{F}_{\mathsf{CRS}}$		for sid and crs $= (\sigma, \rho)$ from $\mathcal{F}_{\mathsf{CRS}}$
param $\leftarrow \mathsf{Setup}(\sigma, \rho)$		param $\leftarrow \mathsf{Setup}(\sigma, \rho)$
$(x_b, w) \leftarrow \mathsf{WordGen}\mathcal{L}(\mathsf{param})$		
$x_{1-b} \leftarrow \mathsf{ComplementWord}(\rho, x_b)$	$\xrightarrow{x_0}$	$x_1 \leftarrow \mathsf{ComplementWord}(\rho, x_0)$
		for i in $\{0,1\}$:
		$\mathsf{hk}_i \leftarrow \mathsf{HashKG}(\mathsf{param})$
		$\mathsf{hp}_i \leftarrow \mathsf{ProjKG}(\mathsf{hk}_i, x_i)$
		$H_i \leftarrow \mathsf{Hash}(\mathsf{hk}_i, x_i)$
$H' \leftarrow \mathsf{ProjHash}(c_{b,1}, x_b, w)$	$\xleftarrow{(c_0, c_1)}$	$c_i = (H_i \oplus m_i, \mathsf{hp}_i)$
$m = H' \oplus c_{b,0}$		

Figure 4.1: General description of the protocol $\Phi_{\mathsf{OT}}^{\mathsf{SPHFwGZ}}$

The protocol $\Phi_{\mathsf{OT}}^{\mathsf{SPHFwGZ}}$ provides **Correctness**. Indeed, with the honest generation $(x, w) \leftarrow \mathsf{WordGen}\mathcal{L}(\mathsf{param})$ we have $c = (H \oplus m, \mathsf{hp})$. Then, $m = H \oplus m \oplus \mathsf{ProjHash}(c_1, x, w)$ if and only if $H = \mathsf{ProjHash}(c_1, x, w)$ which is ensured due to the correctness property of the SPHFwGZ. Moreover, the **complement** property ensures the value x_1 computed by the sender is always the same as the value x_1 computed by the receiver.

We now prove the privacy of our Oblivious Transfer in the Universal Composability framework.

4.2.2 Security analysis

Our Oblivious Transfer protocol will be proven in the CRS-hybrid model (as in [PVW08]), with the functionality $\mathcal{F}_{\mathsf{CRS}}$, where the two players get the same random crs from the sid. In practice, as we assumed $\mathcal{S}_0$ and $\mathcal{R}_0$ efficiently samplable, (σ, ρ) can be derived from $\mathcal{H}(\mathsf{sid})$. Then, in Figure 4.2, we recall the ideal functionality $\mathcal{F}_{\mathsf{OT}}$ for a secure Oblivious Transfer, where there are two first messages from the sender with input (m_0, m_1) and from the receiver with input bit b, to initialize the process, and the final request message by the sender that decides when the receiver can get m_b.

$\mathcal{F}_{\mathsf{OT}}$ interacts with a sender S and a receiver R:

- Upon receiving a message (sid,`sender`, m_0, m_1) from S, store (sid, m_0, m_1);
- Upon receiving a message (sid,`receiver`, b) from R, store (sid, b);
- Upon receiving a message (sid,`answer`) from the adversary, check if both records (sid, m_0, m_1) and (sid, b) exist for sid. If yes, send (sid,m_b) to R, and sid to the adversary and halt. If not, send nothing but continue running.

Figure 4.2: Functionality $\mathcal{F}_{\mathsf{OT}}$

Theorem 4.1

The protocol $\Phi_{\mathsf{OT}}^{\mathsf{SPHFwGZ}}$ UC-realizes $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model in the static-corruption setting, from any SPHFwGZ providing the decomposition intractability.

We stress that we consider static corruptions only, where the corrupted players are known when each protocol execution starts.

Game G_0. This is the real game, where $\mathcal{F}_{\mathsf{CRS}}$ samples crs in $\mathcal{S}_0 \times \mathcal{R}_0$.

Game G_1. In this game, the simulator $\mathcal{S}$ simulates itself the sampling of $\mathsf{crs} = (\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}_0$, and generates correctly every flow from the honest players, as they would do themselves, knowing the inputs (m_0, m_1) and b sent by the environment to the sender and the receiver, respectively.

Game G_2. In this game, we deal with **corrupted receivers**. Instead of sampling $\mathsf{crs} = (\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}_0$, the simulator $\mathcal{S}$ samples $\mathsf{crs} = (\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_1 \times \mathcal{R}_0$, and therefore with the trapdoor td_σ. This game is indistinguishable from the previous one due to the *CRS Indistinguishability*.

Game G_3. In this game, the simulator $\mathcal{S}$ uses the trapdoor td_σ to get $t_i = \mathsf{WordTest}(x_i, \mathsf{td}_\sigma)$ for $i \in \{0, 1\}$. If $t_0 = t_1 = 0$ (none of the words are in $\mathcal{L}'$), $\mathcal{S}$ aborts. This game is indistinguishable from the previous one, under the *Decomposition Intractability*, as $(\sigma, \rho) \in \mathcal{S}_1 \times \mathcal{R}_0$.

Game G_4. If $t_0 = t_1 = 0$, we still abort. If $t_0 = t_1 = 1$ we set $b = 0$, otherwise, we set b such that $t_b = 0$. Next, the simulator $\mathcal{S}$ proceeds on m_b with x_b and on a random message with x_{1-b}. Under the smoothness of the SPHFwGZ, as $x_{1-b} \in \mathcal{L}'$, and the *One-Time Pad Semantic Security*, this game is statistically indistinguishable from the previous one.

Game G_5. In this game, we deal with **corrupted senders**. Instead of sampling $\mathsf{crs} = (\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}_0$, the simulator $\mathcal{S}$ samples $\mathsf{crs} = (\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}_1$, and therefore with the trapdoor $\mathsf{td}_\rho = (x, x', w, w')$. This game is indistinguishable from the previous one due to the *CRS indistinguishability*.

Game G_6. In this game, the simulator $\mathcal{S}$ respectively sets (x_0, w_0, x_1, w_1) as (x, w, x', w') from td_ρ. It can then retrieve both m_0 and m_1. This game is indistinguishable from the previous one due to the *Word Indistinguishability*, and the uniform distribution of the trapdoor.

Game G_7. We now deal with **honest players**. Instead of sampling $\mathsf{crs} = (\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}_0$, the simulator $\mathcal{S}$ samples $\mathsf{crs} = (\sigma, \rho) \leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}'_1$, and therefore with the trapdoor $\mathsf{td}_\rho = (x, x')$, and simulates the flows with random $m_0, m_1 \leftarrow\!\!\$\ \mathcal{M}$ and random $b \leftarrow\!\!\$\ \{0, 1\}$. Under the *CRS Indistinguishability* and the smoothness of the SPHFwGZ, as both $x, x' \in \mathcal{L}'$, coupled with the *One-Time Pad Semantic Security*, this game is indistinguishable from the previous one.

Game G_8. This is the ideal game We can now make use of the functionality $\mathcal{F}_{\mathsf{OT}}$ which leads to the following simulator:

- If no participant is corrupted, one uses crs $\leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}'_1$, and the simulator $\mathcal{S}$ simply uses random inputs for the sender and the receiver;
- If the receiver is corrupted, one uses crs $\leftarrow\!\!\$\ \mathcal{S}_1 \times \mathcal{R}_0$, and the simulator $\mathcal{S}$ extracts b using the trapdoor td_σ, and sends $(\mathsf{sid}, \mathsf{receiver}, b)$ to $\mathcal{F}_{\mathsf{OT}}$;
- If the sender is corrupted, one uses crs $\leftarrow\!\!\$\ \mathcal{S}_0 \times \mathcal{R}_1$, and the simulator $\mathcal{S}$ extracts m_0, m_1 using the trapdoor td_ρ, and sends $(\mathsf{sid}, \mathsf{sender}, m_0, m_1)$ to $\mathcal{F}_{\mathsf{OT}}$;
- The adversary sends $(\mathsf{sid}, \mathsf{answer})$ when it decides to deliver the result to the receiver.

4.2.3 Noisy homomorphic encryption setup

We now define a general setup leading to an instantiation of our Oblivious Transfer from many Homomorphic Encryption. This setup includes encryption schemes with potential decryption failure, as shown with our lattice-based instantiation, thanks to some amplification. We denote the group law $*$ on plaintexts and $\circledast$ on the ciphertexts.

We consider an encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with possible decryption failures. Thus, we set $\mathcal{X}$ as the ciphertext space of Π, whereas

$$\mathcal{L} = \{\mathsf{Enc}(\mathsf{pk}, 0; r)\} \subset \mathcal{X} \text{ and } \mathcal{L}' = \{c \in \mathcal{X}, \mathsf{Dec}(\mathsf{sk}, c) \neq 0\} \subset \mathcal{X}.$$

Sets $\mathcal{S}_0$ and $\mathcal{S}_1$ can both be seen as public keys pk generated from $\mathsf{KGen}(1^\kappa)$ except that when $\sigma \in \mathcal{S}_1$, the secret key sk is known and defines the trapdoor td_σ. Hence, σ (which defines the public key pk) defines the sets $\mathcal{L}$ and $\mathcal{L}'$ in $\mathcal{X}$. On the other hand, we can define $\mathcal{R}_0, \mathcal{R}_1$ and $\mathcal{R}'_1$ as follows:

- $\mathcal{R}_0 = \mathcal{X}$, the set of all the ciphertexts, or a superset, with uniform distribution;
- $\mathcal{R}_1 = \{c_0 \circledast c_1\}$, for two ciphertexts c_0, c_1 in $\mathcal{L}$, following the distribution of the encryption algorithm, on plaintext 0, and according the distribution of the randomness r_0, r_1, which allows to define the trapdoor td_ρ as (c_0, c_1, r_0, r_1);
- $\mathcal{R}'_1 = \{c_0 \circledast c_1\}$, for two ciphertexts c_0, c_1 in $\mathcal{L}'$, following the distribution of the encryption algorithm, on non-zero plaintexts, which allows to define the trapdoor td_ρ as (c_0, c_1).

The setup defined above verifies both basic assumptions required to make the Oblivious Transfer Universally Composable:

- *CRS Indistinguishability*: Under the *semantic security* of the encryption scheme Π, $\mathcal{L}$, $\mathcal{L}'$, and $\mathcal{X}$ are indistinguishable. The homomorphic property implies that $\{x \circledast x' | (x, x') \in \mathcal{X}^2\} = \mathcal{X}$. As a consequence, we have indistinguishability between $\mathcal{R}_0 = \mathcal{X} = \{x \circledast x' | (x, x') \in \mathcal{X}^2\}$, $\mathcal{R}_1 = \{x \circledast x' | (x, x') \in \mathcal{L}^2\}$, and $\mathcal{R}'_1 = \{x \circledast x' | (x, x') \in \mathcal{L}'^2\}$. Furthermore, as $\mathcal{S}_0 = \mathcal{S}_1$, they are perfectly indistinguishable;
- *Word Indistinguishability*: Under the *semantic security* of the encryption scheme Π, one can not distinguish between $c_0 \in \mathcal{L}$, an encryption of 0 and $c_1 \in \mathcal{X}$, an encryption of a random value.
- **Complement:** for any $x \in \mathcal{X}$, if $x' \leftarrow \mathsf{ComplementWord}(\mathsf{param}, \rho, x) = \rho \circledast x^{-1}$, hence we have $\mathsf{ComplementWord}(\mathsf{param}, \rho, x') = \rho \circledast (\rho \circledast x^{-1})^{-1} = x$;

Additional properties will depend on concrete instantiations.

4.3 Concrete instantiations of SPHFwGZ

We now provide two concrete instantiations of SPHFwGZ based on the Diffie-Hellman and Learning With Errors problems. As both constructions rely on a homomorphic encryption scheme, we can already consider the basic properties shown in Section 4.2.3.

4.3.1 Instantiation from the Diffie-Hellman problem

In this section, we focus on elliptic curve based cryptography, using the Decisional Diffie-Hellman assumption in a prime-order group.

4.3.1.1 SPHFwGZ from ElGamal encryption.

From the IND-CPA ElGamal encryption scheme in a group $\mathbb{G}$, using multiplicative notation, of prime order p, with generator g.

We set $\mathcal{S}_0 = \mathcal{S}_1 = \{\sigma = h = g^{\mathsf{td}_\sigma}; \mathsf{td}_\sigma \leftarrow\$ \mathbb{Z}_p\}$. Then, $\mathcal{R}_0$ is defined as $\mathbb{G}^2 = \{\rho = (\hat{g}, \hat{h}) \leftarrow\$ \mathbb{G}^2\}$, $\mathcal{R}_1$ as $\{\rho = (\hat{g} \leftarrow g^{r_0} \cdot g^{r_1}, \hat{h} \leftarrow h^{r_0} \cdot h^{r_1}); (r_0, r_1) \leftarrow\$ \mathbb{Z}_p\}$ and $\mathcal{R}'_1$ as $\{c_0 \circledast c_1; (c_0, c_1) \in \mathbb{G}^{2\times 2}\}$. The crs is set as (σ, ρ). One can note that witnesses only exist when $\rho \in \mathcal{R}_1$, then $\mathsf{td}_\rho = (c_0 = (g^{r_0}, h^{r_0}), c_1 = (g^{r_1}, h^{r_1}), r_0, r_1)$. Hence, c_0 and c_1 are encryptions of $M = g^0$, with respective randomness r_0 and r_1. Moreover, while td_σ always exists, it is not necessarily known.

From the above generic construction, we have $\mathcal{X} = \{(g^r, h^r \cdot M); M \in \mathbb{G}\} = \mathbb{G}^2$ and $\mathcal{L} = \{(g^r, h^r)\}$, which are indistinguishable under the Decisional Diffie-Hellman assumption. With $\mathsf{param} = (g, \sigma = h)$, which determines all the sets (specified by the Setup algorithm), we can define:

- $\mathsf{hk} = \mathsf{HashKG}(\mathsf{param}) = (\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$;
- $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}) = g^\alpha h^\beta$;
- $H = \mathsf{Hash}(\mathsf{hk}, x = (u, v)) = u^\alpha v^\beta \in \mathbb{G}$;
- $H' = \mathsf{ProjHash}(\mathsf{hp}, x, w = r) = \mathsf{hp}^r$, if $x = (g^r, h^r) \in \mathcal{L}$.
- $x' = (u', v') = \mathsf{ComplementWord}(\rho = (\hat{g}, \hat{h}), x = (u, v)) = (\hat{g} \cdot u^{-1}, \hat{h} \cdot v^{-1})$

This is a word-independent SPHFwGZ. And we can show the expected properties:

- **Correctness:** When $x = (u, v) = (g^r, h^r) \in \mathcal{L}$, with witness r, $H = u^\alpha v^\beta = (g^\alpha h^\beta)^r = \mathsf{hp}^r = H'$;
- **Smoothness:** When $x = (u, v) = (g^r, h^{r'}) \notin \mathcal{L}$, then $r' = r + r''$ with $r'' \neq 0$: $H = u^\alpha v^\beta = (g^\alpha h^\beta)^r \times g^{r''\beta} = \mathsf{hp}^r \times g^{r''\beta} = H' \times g^{r''\beta}$. But β is perfectly hidden in hp, and $g^{r''\beta}$ is perfectly unpredictable;
- **Decomposition Intractability:** In ElGamal encryption there is no decryption failure: all the ciphertexts can be covered by the encryption algorithm, and the decryption perfectly inverts the encryption process. So we have $\mathcal{L}' = \mathcal{X} \backslash \mathcal{L}$. A random ciphertext ρ encrypts an $M \neq 1$ with overwhelming probability. Then, when it encrypts $M \neq 1$, from the homomorphic property, this is impossible to have two encryptions of 1 whose product is ρ. Hence, the *decomposition intractability* is statistical: the probability of existence of the decomposition is bounded by $1/p$, on ρ, even knowing the decryption key, and thus the trapdoor td_σ.

Note that this construction exactly corresponds to the one from [CS98].

4.3.2 Instantiation from the Learning With Errors problem

In this section, we focus on lattice-based cryptography. We are going to show how to instantiate the various required components from the Learning With Errors problem.

Regev Encryption. Regev [Reg05] showed that for $\chi = D_{\mathbb{Z},\sigma}$, a Gaussian centered distribution in $\mathbb{Z}$ for any standard deviation $\sigma \geq 2\sqrt{n}$, and q such that $q/\sigma = \text{poly}(n)$, $\text{LWE}_{\chi,q}$ is at least as hard as solving worst-case SIVP for polynomial approximation factors, which is assumed to be hard to solve, even for quantum computers.

Trapdoor for LWE. Throughout this section, we will use the trapdoors introduced in [MP12] to build our public matrix $\mathbf{A}$. Define $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{As} + \mathbf{e}$, the gadget matrix $\mathbf{G}$ as $\mathbf{G}^t = \mathbf{I}_n \otimes \mathbf{g}^t$, where $\mathbf{g}^t = [1, 2, \ldots, 2^k]$ and $k = \lceil \log q \rceil - 1$, and let $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ be invertible. The notation $[\mathbf{A} \,|\, \mathbf{B}]$ is for horizontal concatenation, while $[\mathbf{A}\,;\,\mathbf{B}]$ is for vertical concatenation.

Lemma 1 ([MP12, Theorems 5.1 and 5.4]) *There exist two PPT algorithms TrapGen and $g_{(\cdot)}^{-1}$ with the following properties assuming $q \geq 2$ and $m \geq \Theta(n \log q)$:*

- *$\text{TrapGen}(1^n, 1^m, q)$ outputs $(\mathbf{T}, \mathbf{A}_0)$, where the distribution of the matrix $\mathbf{A}_0$ is at negligible statistical distance from uniform in $\mathbb{Z}_q^{m \times n}$, and such that $\mathbf{TA}_0 = \mathbf{0}$, where $s_1(\mathbf{T}) \leq O(\sqrt{m})$ and where $s_1(\mathbf{T})$ is the operator norm of $\mathbf{T}$, which is defined as $\max_{\mathbf{x} \neq 0} \|\mathbf{Tx}\|/\|\mathbf{x}\|$.*[2]
- *Let $(\mathbf{T}, \mathbf{A}_0) \leftarrow \text{TrapGen}(1^n, 1^m, q)$. Let $\mathbf{A}_{\mathsf{H}} = \mathbf{A}_0 + [\mathbf{0}\,;\,\mathbf{GH}]$ for some invertible matrix $\mathbf{H}$ called a* tag*. Then, we have $\mathbf{TA}_{\mathsf{H}} = \mathbf{GH}$. Furthermore, if $\mathbf{x} \in \mathbb{Z}_q^m$ can be written as $\mathbf{A}_{\mathsf{H}}\mathbf{s} + \mathbf{e}$, with $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{e} \in \mathbb{Z}_q^m$ where $\|\mathbf{e}\| \leq B' := q/\Theta(\sqrt{m})$, then $g_{\mathbf{A}_{\mathsf{H}}}^{-1}(\mathbf{T}, \mathbf{x}, \mathbf{H})$ outputs $(\mathbf{s}, \mathbf{e})$.*

More precisely, to sample $(\mathbf{T}, \mathbf{A}_0)$ with TrapGen, we sample a uniform $\bar{\mathbf{A}} \in \mathbb{Z}_q^{\bar{m} \times n}$ where $\bar{m} = m - nk = \Theta(n \log q)$, and some $\mathbf{R} \leftarrow \mathcal{D}^{nk \times \bar{m}}$, where the distribution $\mathcal{D}^{nk \times \bar{m}}$ assigns probability $1/2$ to 0, and $1/4$ to ± 1. We output $\mathbf{T} = [-\mathbf{R} \,|\, \mathbf{I}_{nk}]$ along with $\mathbf{A}_0 = [\bar{\mathbf{A}}\,;\,\mathbf{R}\bar{\mathbf{A}}]$. Then, given a tag $\mathbf{H}$, with $\mathbf{A}_{\mathsf{H}} = \mathbf{A}_0 + [\mathbf{0}\,;\,\mathbf{GH}]$, we have: $\mathbf{TA}_{\mathsf{H}} = \mathbf{GH}$.

We will only consider a fixed tag $\mathbf{H} = \mathbf{I}$, for the Micciancio-Peikert encryption [MP12]. Our construction only requires CPA encryption so we don't need several tags, but we need to be able to reject improperly computed ciphertexts, and the gadget matrix is here, to allow this extra control during the decryption.

LWE Encryption à la Micciancio-Peikert. For this scheme, we assume q to be an odd prime. We set an encoding function for messages $\text{Encode}(\mu \in \{0,1\}) = \mu \cdot (0, \ldots 0, \lceil q/2 \rceil)^t$. Note that $2 \cdot \text{Encode}(\mu) = (0, \ldots, 0, \mu)^t \bmod q$, as $\lceil q/2 \rceil$ is the inverse of 2 mod q, for such an odd q.

Let $(\mathbf{T}, \mathbf{A}_0) \leftarrow \text{TrapGen}(1^n, 1^m, q)$. The public encryption key is $\mathsf{pk} = \mathbf{A}_0$, and the secret decryption key is $\mathsf{sk} = \mathbf{T}$.

- $\text{Enc}(\mathsf{pk} = \mathbf{A}_0,\ \mu \in \{0,1\})$ encrypts the message μ under the public key pk as follows: Let $\mathbf{A} = \mathbf{A}_0 + [\mathbf{0}\,;\,\mathbf{G}]$. Pick $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow D_{\mathbb{Z},t}^m$ where $t = \sigma\sqrt{m} \cdot \omega(\sqrt{\log n})$. Restart if $\|\mathbf{e}\| > B$, where $B := 2t\sqrt{m}$.[3] Output the ciphertext:

$$\mathbf{c} = \mathbf{As} + \mathbf{e} + \text{Encode}(\mu) \bmod q \ .$$

- $\text{Dec}(\mathsf{sk} = \mathbf{T},\ \mathbf{c} \in \mathbb{Z}_q^m)$ decrypts the ciphertext $\mathbf{c}$ using the decryption key sk as follows: With $B'' := q/2\Theta(\sqrt{m})$, output

$$\begin{cases} \mu & \text{if } g_{\mathbf{A}}^{-1}(\mathbf{T}, 2\mathbf{c}, \mathbf{I}) = (2\mathbf{s}, 2\mathbf{e} + (0, \ldots, 0, \mu)) \text{ where } \mathbf{s} \in \mathbb{Z}_q^n, \mathbf{e} \in \mathbb{Z}^m \text{ and } \|\mathbf{e}\| \leq B'' \ , \\ \perp & \text{otherwise.}[4] \end{cases}$$

[2] The bound on $\mathbf{s}_1(\mathbf{T})$ holds except with probability at most 2^{-n} in the original construction, but we assume the algorithm restarts if it does not hold.

[3] This happens only with exponentially small probability $2^{-\Theta(n)}$.

With $\Lambda(A) = \{\mathbf{As} | \mathbf{s} \in \mathbb{Z}_q^n\}$, honestly generated ciphertext $\mathbf{c}$ are such that $d(\mathbf{c} - \mathsf{Encode}(\mu), \Lambda(\mathbf{A})) \leq B$, while the decryption procedure is guaranteed not to return μ as soon as $d(\mathbf{c} - \mathsf{Encode}(\mu), \Lambda(\mathbf{A})) > B''$. From the decryption procedure, we have:

$$\mu' := \mathsf{Dec}(\mathbf{T}, \mathbf{c}) \neq \perp \quad \iff \quad d(\mathbf{c} - \mathsf{Encode}(\mu'), \Lambda(\mathbf{A})) < B'' \ .$$

Suppose that $m \geq \Theta(n \log q)$. The scheme is correct as long as $B \leq B''$, or equivalently $2\sigma m^{3/2} \cdot \omega(\sqrt{\log n}) \leq q$.

Theorem 4.2

Assume $m \geq \Theta(n \log q)$. The above scheme is IND-CPA assuming the hardness of the $\mathsf{LWE}_{\chi,q}$ problem for $\chi = D_{\mathbb{Z},\sigma}$.

Furthermore, this encryption scheme is homomorphic for plaintexts in $(\mathbb{Z}_2, +)$, and ciphertexts in $\mathbb{Z}_q^m$ with component-wise addition.

Bit-SPHFwGZ from LWE Encryption Scheme. We consider, an LWE encryption scheme defined with a superpolynomial modulus. More precisely, we set $m = n\log(q)$, $t = \sqrt{mn}.\omega(\sqrt{\log(n)})$, $k = \Theta(n), s \geq \Theta(\sqrt{n}) \wedge s/q = \mathsf{negl}(n), s = \Omega(mk^2q^{2/3})$. We also set R to be a *probabilistic* rounding function from $[0, 1]$ to $\{0, 1\}$, such that $R(x) = 1$ with probability $0.5 \cdot \cos(\frac{2\pi x}{q})$ and 0 otherwise.

We set $\mathcal{S}_0 = \mathcal{S}_1 = \{\sigma = \mathbf{A} = \mathbf{A}_0 + [\mathbf{0}; \mathbf{G}] | (\mathbf{T}, \mathbf{A}_0) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)\}$, td_σ being $\mathbf{T}$. Then, $\mathcal{R}_0$ is defined as $\{\rho = \mathbf{v} \in \mathbb{Z}_q^m\}$ and $\mathcal{R}_1$ is the set composed of all the sums of two honest encryptions of 0, in other words as $\{\rho = \mathbf{A}(\mathbf{s} + \mathbf{s}') + \mathbf{e} + \mathbf{e}' \bmod q \ \ | \mathbf{s}, \mathbf{s}' \leftarrow\$ \mathbb{Z}_q^n, \ \mathbf{e}, \mathbf{e}' \leftarrow D_{\mathbb{Z},t}^m$ with $\|\mathbf{e}\| \leq B$ and $\|\mathbf{e}'\| \leq B\}$, more simply $\mathcal{R}_1$ can be written as $\{\rho = \mathbf{A}(\mathbf{s}+\mathbf{s}')+\mathbf{e}+\mathbf{e}' \bmod q \ \ | \mathbf{s}, \mathbf{s}' \in \mathbb{Z}_q^n, \ \mathbf{e}, \mathbf{e}' \leftarrow D_{\mathbb{Z},t}^m \wedge \|\mathbf{e}\| \leq B \wedge \|\mathbf{e}'\| \leq B\}$ with $\mathsf{td}_\rho = (\mathbf{As} + \mathbf{e}, \mathbf{As}' + \mathbf{e}', (\mathbf{s}, \mathbf{e}), (\mathbf{s}', \mathbf{e}'))$.

With $\mathcal{X} = \{\mathbf{c} \leftarrow\$ \mathbb{Z}_q^m\}$, $\mathcal{L} = \{\mathbf{c} | \exists \mathbf{s}, \mathbf{e}, \mathbf{c} = \mathsf{Enc}(\mathbf{A}_0, 0; \mathbf{s}, \mathbf{e})\}$ defined following the description above, and $\mathcal{L}' = \{\mathbf{c} \in \mathcal{X} | \mathsf{Dec}(\mathbf{T}, \mathbf{c}) \neq 0\}$, note that $\mathbf{s}$ could be enough as a witness for $\mathbf{c} = \mathbf{As} + \mathbf{e} \in \mathcal{L}$, as one can check $\mathbf{e} = \mathbf{c} - \mathbf{As}$ is small enough. This defines the Setup algorithm, and we have:

Definition 4.2 - Bit-SPHFwGZ over Micciancio-Peikert like Ciphertexts [ ]

For $k = \Theta(n)$, and picking $s \geq \Theta(\sqrt{n})$, and $s = \Omega(mk^2q^{2/3})$, we can define:

- $\mathsf{HashKG}(\mathsf{param}) = \mathsf{hk} = \mathbf{h} \leftarrow\$ D_{\mathbb{Z},s}^m$
- $\mathsf{ProjKG}(\mathsf{hk}) = \mathsf{hp} = \mathbf{A}^t\mathbf{h}$
- $\mathsf{Hash}(\mathsf{hk}, \mathbf{c}) = R(\langle \mathsf{hk}, \mathbf{c} \rangle) = R(\langle \mathbf{h}, \mathbf{c} \rangle) \in \{0, 1\}$
- $\mathsf{ProjHash}(\mathsf{hp}, \mathbf{c}, w = \mathbf{s}) = R(\langle \mathsf{hp}, \mathbf{s} \rangle) = R(\langle \mathbf{A}^t\mathbf{h}, \mathbf{s} \rangle)$

For a word $\mathbf{c} = \mathbf{As} + \mathbf{e}$ in the language $\mathcal{L}$, $\langle \mathbf{h}, \mathbf{c} \rangle = \mathbf{h}^t\mathbf{As} + \mathbf{h}^t\mathbf{e} = \langle \mathbf{A}^t\mathbf{h}, \mathbf{s} \rangle + \mathbf{h}^t\mathbf{e}$. And by construction $\mathbf{h}^t\mathbf{e}$ is small. The choice of the rounding function $R(x)$, characterized by a coin flip where the outcome 1 is weighted by $0.5 \cdot \cos(\frac{2\pi x}{q})$, is such that it allows to cancel out this small noise most of the time, while providing smoothness for words outside the language (ensuring that $R(\langle \mathsf{hk}, \mathbf{c} \rangle)$ is random when given only hp)

It was shown in [BBDQ18], that for this choice of random function, such bit-SPHFwGZ achieves negligible-universality, thanks to the rounding function, but $(3/4 + o(1))$-correctness for the chosen set of parameters.

[4] Note that the inversion algorithm $g_{(\cdot)}^{-1}$ can succeed even if $\|\mathbf{e}\| > B''/2$, depending on the randomness of the trapdoor. It is crucial to reject decryption nevertheless when $\|\mathbf{e}\| > B''$ to ensure security.

Full-Fledged SPHFwGZ from LWE. The previous construction has limitations as it is neither perfectly correct, nor smooth, we need to apply a transformation to reach those goals. This transformation is explained below, first informally, then in more details:

- It is a bit-function meaning the final hash value lives in $\{0,1\}$, while one needs a larger mask. To solve this issue, one has to run it in parallel a linear number of times, to have an output string long enough.
- The correctness is imperfect. The output bit only matches with probability $3/4 + o(1)$. As such, applications running $\mathsf{Enc}(\mathsf{pk}, m; r)$ should encryption a redundant version of m, with an error-correcting code, $\mathsf{ECC}(m)$. Such transformation makes the SPHF word-dependent (i.e. the projection key is dependent on the user/receiver input), however in our scenario, such a word-dependent function is enough.
- $\forall i \in [\ell]$, the verifier generates $\mathbf{h}_i \leftarrow\!\!\$\ D^m_{\mathbb{Z},s}$, $\mathsf{hp}_i = \mathbf{A}^t\mathbf{h}_i$, $H_i = R(\langle \mathsf{hk}, \mathbf{c}\rangle)$
- It then picks a random value $K \leftarrow \{0,1\}^\kappa$, and compute $T = \mathsf{ECC}(K) \oplus S$ where $S = \|_i H_i$, and sends $(\mathsf{hp}_i)_{i\in[\ell]}, S$ to the prover.
- The prover can then compute, $\forall i \in [\ell]$, $H'_i = R(\langle \mathsf{hp}, \mathbf{s}\rangle)$, then $S' = \|_i H'_i$, and finally recover $K' = \mathsf{ECC}^{-1}(T \oplus S')$. As the SPHFwGZ has a correctness of $3/4+o(1)$, the error correcting code can correct the small differences between S and S', and so $K' = K$.

To describe this in terms of SPHF, we use capital letters for our new construction:

- $\mathsf{SETUP}(1^\kappa)$: Outputs the result from $\mathsf{Setup}(1^\kappa)$
- $\mathsf{HASHKG}(\mathsf{param})$: Picks a random $K \leftarrow \{0,1\}^\kappa$, and $\forall i \in [\ell]$, gets $\mathsf{hk}_i = \mathsf{HashKG}(\mathsf{param})$, and set $\mathsf{HK} = (\{\mathsf{hk}_i\}, K)$;
- $\mathsf{ProjKG}(\mathsf{HK}, \mathbf{c})$: $\forall i \in [\ell]$, gets $\mathsf{hp}_i = \mathsf{ProjKG}(\mathsf{hk}_i), H_i = \mathsf{Hash}(\mathsf{hk}_i, \mathbf{c})$. It then computes $T = \mathsf{ECC}(K) \oplus S$ where $S = (H_i)_{i\in[\ell]}$, and outputs $\mathsf{HP} = ((\mathsf{hp}_i)_{i\in[\ell]}, T)$;
- $\mathsf{HASH}(\mathsf{HK}, \mathbf{c})$: Returns K, from HK;
- $\mathsf{PROJHASH}(\mathsf{HP}, \mathbf{c}, w = \mathbf{s})$: $\forall i \in [\ell]$, computes $H'_i = \mathsf{ProjHash}(\mathsf{hp}_i, \mathbf{c}, \mathbf{s})$. Then computes $S' = (H'_i)_{i\in[\ell]}$, and finally $K' = \mathsf{ECC}^{-1}(T \oplus S')$.

Such transformation allows to achieve *smoothness* which can be proven with an hybrid argument, handling intermediate distributions where the first H_i values are random. The *correctness* is simply inherited from the correcting-code capacity, while the number of errors to be corrected can be estimated thanks to the Hoeffding's bound [Hoe63]. We can guarantee the expected properties:

- **Correctness:** When $x = \mathbf{c} \in \mathcal{L}$, with the above conversion, we have $K = K'$ with overwhelming probability, thanks to the error-correcting code;
- **Smoothness:** When $x = \mathbf{c} \notin \mathcal{L}$, then the value K is random from an adversary point of view, as the parallelization technique allows to transform the negligible-universality to a classical smoothness (at the cost of a word-dependent SPHF);
- **Half Decomposition Intractability:** A random vector ρ should not be split into two ciphertexts that could be decrypted to 0, or at least not too often. We first deal with *half* decomposition intractability, when at most half of the random vectors can be split.

  To get a lower-bound on the number of vectors like such ρ, we can remark that a vector verifies this property as soon as $d(\rho, \Lambda(\mathbf{A}))$ is greater than 2 times the decryption bound.

  This is the reason why we took a conservative value $B'' = B'/2$ in the encryption compared to classical Micciancio-Peikert encryption. By halving the decryption radius, we ensured that adding two elements that still decrypt within this bound will fall on classically decryptable ciphertexts. As such, at least half the elements cannot be reached (those that classically decrypted to 1). Hence, $\Pr_{\rho\in\mathbb{Z}_q^m}[\exists \mathbf{c}, \mathbf{d} | \rho = \mathbf{c} + \mathbf{d} \wedge \mathsf{Dec}(\mathsf{sk}, \mathbf{c}) = \mathsf{Dec}(\mathsf{sk}, \mathbf{d}) = 0] \leq 1/2$. This is a statistical bound, that holds even when knowing the decryption key.

Another amplification is required to make *full-fledged decomposition intractability*, by working on ciphertexts $(\mathbf{c}_j)_{j\in[k]}$, with k parallel executions of the SPHFwGZ, with a final XOR of all the outputs, so that the smoothness for one word is enough to get the smoothness for the vector of words, but the correctness on all the words leads to the global correctness. The acceptable language, for correctness is then:

$$\tilde{\mathcal{L}} = \mathcal{L}^k = \{(\mathbf{c}_j)_{j\in[k]} | (\forall j \in [k]), \exists(\mathbf{s}_j, \mathbf{e}_j), \mathbf{c} = \mathsf{Enc}(\mathbf{A}_0, 0; \mathbf{s}_j, \mathbf{e}_j)\} \subset \mathcal{X}^k$$

whereas the language for the smoothness becomes:

$$\tilde{\mathcal{L}}' = \mathcal{L}'^k = \{(\mathbf{c}_j)_{j\in[k]} | (\exists j \in [k]), \mathsf{Dec}(\mathbf{T}, \mathbf{c}_j) \neq 0\} \subset \mathcal{X}^k$$

Then, for random $(\rho_j)_{j\in[k]} \leftarrow\!\!\$\ \mathcal{X}^k$, a decomposition would be a list of pairs $(\mathbf{c}_j, \mathbf{d}_j)_{j\in[k]} \in (\mathcal{X}\times\mathcal{X})^k$ such that for all j, $\rho_j = \mathbf{c}_j + \mathbf{d}_j$ and $\mathsf{Dec}(\mathbf{T}, \mathbf{c}_j) = \mathsf{Dec}(\mathbf{T}, \mathbf{d}_j) = 0$, which only exists with probability less than $1/2^k$. We thus have achieved all the security properties required for our applications.

Chapter 5

Computational security analysis of the EDHOC protocol

A key agreement is under analysis by IETF [SMP21], under the name Ephemeral Diffie-Hellman Over COSE (EDHOC). EDHOC aims at being a very compact and lightweight authenticated Diffie-Hellman key exchange with ephemeral keys. This protocol is deeply inspired from SIGMA [Kra03] and OPTLS [KW16] and targets constrained devices over low-power IoT radio communication technologies. For this reason, very aggressive parameters are proposed to minimize the communications. Nonetheless, it is still expected to provide mutual authentication, forward secrecy, and identity protection, with a 128-bit security level even with those aggressive parameters. A formal analysis has already been proposed at SECRYPT'21 [NSB21], on a former version, leading to some improvements, in the ongoing evaluation process by IETF. Unfortunately, while formal analysis can detect some misconceptions in the protocol, it cannot evaluate the actual security level. In this chapter, we analyze the computational security of the last version of EDHOC.

★ Baptiste Cottier and David Pointcheval. Security Analysis of Improved EDHOC Protocol. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022

Contents

Notations & Acronyms

Protocol Description	
CBOR	Concise Binary Object Representation
COSE	CBOR object signing and encryption
EDHOC	Ephemeral Diffie-Hellman over COSE
$\mathcal{H}$	Hash function SHA-256 (256 bits digest)
X_e, x_e	Initiator Ephemeral DH Public and Secret Key
X_s, x_s	Initiator Static DH Public and Secret Key
Y_e, y_e	Responder Ephemeral DH Public and Secret Key
Y_s, y_s	Responder Static DH Public and Secret Key
HKDF	HMAC-based Key Derivation Function
PRK	Pseudorandom Key
$\mathcal{E}, \mathcal{D}$	One-time Encryption and Decryption
$\mathcal{E}', \mathcal{D}'$	Authenticated Encryption and Decryption with Associated Data
IV	Initialization Vector
EAD	External Authorized Data
$\perp$	Protocol abortion
TH	Transcript Hash
t_2, t_3	MAC tags
$\mathsf{C_R}, \mathsf{C_I}$	Connection Identifiers

Security Analysis	
ℓ_{MAC}	MAC output length
$\ell_{\mathsf{plaintext}}$	Plaintext length of the first message send by the responder
ℓ_{hash}	Hash length
$\ell_{\mathsf{key}}, \ell_{\mathsf{IV}}$	Key and IV length

Figure 5.1: Notations

We first give a description of the EDHOC protocol, then prove key privacy, explicit authentication and identity protection. We close the chapter by giving some improvements to the actual protocol.

5.1 Protocol description

Ephemeral Diffie-Hellman over COSE [Sch16] (EDHOC) aims to provide a public-key exchange of two parties, denoted Initiator (that initiates the key exchange) and Responder, potentially running on constrained devices over low-power IoT radio communication technologies. EDHOC protocol can be instantiated with several settings:

- *Authentication Method*: Each party (Initiator and Responder) can use an authentication method: either by Signature or by using a Static DH Key.

Value	Initiator	Responder	Denotation
0	Signature	Signature	`SIG-SIG`
1	Signature	Static DH	`SIG-STAT`
2	Static DH	Signature	`STAT-SIG`
3	Static DH	Static DH	`STAT-STAT`

- *Cipher Suites*: Ordered set of protocol security settings. Initial paper offers many possible suites, including ones with either 8 or 16 bytes MAC, but we focus on the cipher suites with agressive MAC length set to 8 bytes, i.e. Cipher Suites 0 and 2 which share the following parameters:

(Application) AEAD	Hash	MAC len
AES-CCM-16-64-128	SHA-256	8

  The difference between Cipher Suites 0 and 2 is the Elliptic Curve used: X25519 in suite 0 and P-256 in suite 2.
- *Connection Identifiers.* Data that may be used to correlate between messages and facilitate retrieval of protocol state in EDHOC and application.
- *Credentials and Identifiers.* They are used to identify and optionally transport the authentication keys of the Initiator and the Responder.

As this is the setup in which we are doing the security analysis, we can assume both the Initiator and the Responder agreed to use the authentication method 3, where both use Static Diffie-Hellman keys. Also, as the difference between Cipher Suite 0 and Cipher Suite 2 is the choice of the Elliptic Curve, both providing the same security, we do not include the Cipher Suite ID `Suites_I` (as it appears in [SMP21]) in the first message of the protocol.

Extract and Expand. In the EDHOC Key-Schedule defined below, the pseudorandom keys (PRK) are derived using an extraction function. In our context, as we study cipher suites 0 and 2, our hash algorithm is set as SHA-256. Therefore Extract(salt, IKM) =HKDF -Extract (salt, IKM) is defined with SHA-256, where IKM holds for Input Keying Material (in our context, this will be some Diffie-Hellman keys) and Expand(PRK, info, len) =HKDF -Expand (PRK, info, len) where info contains the transcript hash (TH_2, TH_3 or TH_4), the name of the derived key and some context, while len denotes the output length.

Key Schedule. During the key exchange, several cryptographic computations occur. The keying material, including MAC Keys (as we study `STAT-STAT` method), Encryption Keys and Initialisation Vectors result from a key schedule, adapted from Norrman et al. [NSB21]. In Figure 5.2, purple boxes denote Diffie-Hellman shared secrets ($g^{x_e y_e}$, $g^{y_s x_e}$, $g^{x_s y_e}$), where x_e and y_e denote the ephemeral keys, while x_s and y_s are the long-term static DH keys. From those Diffie-Hellman shared secrets, we extract the pseudo random keys PRK_{2e}, PRK_{3e2m} and PRK_{4e3m}, in light blue. Those keys are then expanded to compute the encryption material (keys and IV), in red, and the authentication tags t_2 and t_3, in yellow. Keys PRK_{2e} and PRK_{3e2m} are also used to compute the salts salt_{3e2m} and salt_{4e3m} respectively, in gray, used in the HKDF -Extract function. The final session key $\mathsf{PRK}_{\mathsf{out}}$, backgrounded in green, is computed calling HKDF -Expand on PRK_{4e3m}. Transcript hashes, denoted TH_i, are used as input to the HKDF -Expand function. More precisely, with SHA-256 as $\mathcal{H}$, we have:

$$\mathsf{TH}_2 = \mathcal{H}(Y_e, \mathsf{C_R}, \mathcal{H}(m_1)) \qquad \mathsf{TH}_3 = \mathcal{H}(\mathsf{TH}_2, m_2) \qquad \mathsf{TH}_4 = \mathcal{H}(\mathsf{TH}_3, m_3)$$

where m_1 is the first message sent by the Initiator, m_2 and m_3 respectively are the plaintexts respectively encrypted in the message 2 and message 3.

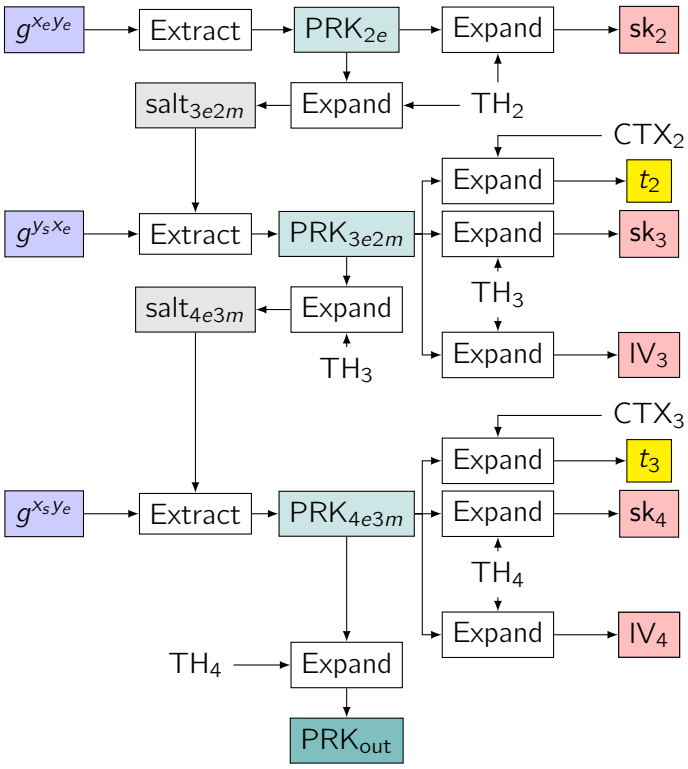


Figure 5.2: Key Derivation (for the `STAT-STAT` Method).

Authenticated Encryption with Associated Data (AEAD). As said above, the cipher suites we work on both use AES-CCM-16-64-128. We detail this Authenticated Encryption scheme:

- AES-CCM: CCM, for Counter with CBC-MAC is an AES mode providing both encryption and authentication. CCM mode combines the CBC-MAC and the CTR (counter) mode of encryption. The first step consists in calculating a tag T, then, encrypt the message and the tag using the counter mode. Hence, the authenticated encryption is done in the Mac-then-Encrypt approach.
- 16: messages length is limited to 2^{16} bytes long (64KiB). Therefore, the nonce is 13 bytes long allowing $2^{13\times 8}$ possibles values of the nonce without repeating

- 64: Tag is 64 bits long.
- 128: Key is 128 bits long.

Connection Identifiers (from [SMP21]). Connection identifiers (C_I and C_R) may be used to correlate EDHOC messages and facilitate the retrieval of protocol state during EDHOC protocol execution or in a subsequent application protocol. The connection identifiers do not have any cryptographic purpose in EDHOC.

EDHOC-Exporter and EDHOC-KeyUpdate. At the end of the protocol, the Initiator and the Responder compute TH_4. This value can be used in case an application need to export the EDHOC session key. Also, in case the key needs to be updated, the Initiator and the Responder rerun HKDF -Extract, with PRK_{4e3m} as input, together with a random nonce agreed upon by the Initiator and the Responder.

Protocol. The detailed description of the protocol is given in Fig 5.3. The final session key is $SK = PRK_{out}$

Security Goals. The security goals of an authenticated key exchange protocol are:

- *Key Privacy*: Equivalent to Implicit Authentication. At most both participants know the final session key. With additional *Perfect Forward Secrecy*, by compromising the long-term credential of either peer, an attacker shall not be able to compute past session keys. In our context, this will rely on a Diffie-Hellman assumption.
- *Mutual Authentication*: Equivalent to explicit authentication. Exactly both participants have the material to compute the final session key.
- *Identity Protection*: At most both participants know the identity of the Initiator and the Responder.

5.2 Key privacy

An authenticated key exchange protocol AKE can be defined using three algorithms:

- KGen(ID) takes an identity ID as input and samples a long term pair of keys $(\mathsf{pk}, \mathsf{sk})$. Key pairs are associated to that user with identity ID, and the public key is added to the list `peerpk`.
- Activate(ID, `role`) takes as input a user identity ID and its `role` $\in$ {initiator, responder}. Activate returns a state st and a message m'.
- Run(ID, sk, `peerpk`, m) delivers m to the session of user ID with secret key sk and state `st`. Run update the state `st` and returns the response message m'

Algorithm Run takes as implicit argument a state `st`, that contains some information, denoted in `typewriter font`, about the actual session. A value `peerid` $\in \mathbb{N}$ used to identify the intended partner identity of the session, a `role`, either initiator or responder, defining the role played by the session. The state also contains the `status` of the actual session. Either the session `status` is **running**, meaning the session has been activated, **accepted**, meaning that a party has all the material to compute the session key or **terminated** when the session key is computed and the protocol is ended. We also consider a **rejected** flag in case the session meets a mistake in the verification phase. The final session key is stored in SK and set as $\perp$ until defined by the key schedule. Finally, `sid` stores the session identifier used to define partnered session in the security model.

Initiator		Responder
$X_s = g^{x_s}$	static keys	$Y_s = g^{y_s}$
$\underline{\text{InitRun1}(\text{ID}_\text{I})}$		
$x_e \leftarrow\$ \mathbb{Z}_p, X_e \leftarrow g^{x_e}$		
$\text{C}_\text{I} \leftarrow\$ \{0,1\}^{nl}$		
$m_1 \leftarrow (X_e \Vert \text{C}_\text{I} \Vert \text{EAD}_1)$	$\xrightarrow{m_1}$	$\underline{\text{RespRun1}(\text{ID}_\text{R}, y_s, m_1)}$
		Parse m_1 as $(X_e \Vert c \Vert \text{EAD}_1)$
		$y_e \leftarrow\$ \mathbb{Z}_p, Y_e \leftarrow g^{y_e}$
		$\text{C}_\text{R} \leftarrow\$ \{0,1\}^{nl}$
		$\text{sid} \leftarrow (\text{C}_\text{I}, \text{C}_\text{R}, X_e, Y_e)$
		$\text{PRK}_{2e} \leftarrow \text{HKDF-Extract}(\text{""}, {X_e}^{y_e})$
		$\text{TH}_2 \leftarrow \mathcal{H}(Y_e, \text{C}_\text{R}, \mathcal{H}(m_1))$
		$\text{sk}_2 \leftarrow \text{HKDF-Expand}(\text{PRK}_{2e}, 0, \text{TH}_2, \ell_{\text{plaintext}})$
		$\text{salt}_{3e2m} \leftarrow \text{HKDF-Expand}(\text{PRK}_{2e}, 1, \text{TH}_2, \ell_{\text{hash}})$
		$\text{PRK}_{3e2m} \leftarrow \text{HKDF-Extract}(\text{salt}_{3e2m}, {X_e}^{y_s})$
		$\text{CTX}_2 \leftarrow (\text{ID}_\text{R} \Vert \text{TH}_2 \Vert Y_s \Vert \text{EAD}_2)$
		$t_2 \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 2, \text{CTX}_2, \ell_{\text{MAC}})$
		$m_2 \leftarrow (\text{ID}_\text{R} \Vert t_2 \Vert \text{EAD}_2)$
$\underline{\text{InitRun2}(\text{ID}_\text{I}, x_s, Y_s, (Y_e, c_2, \text{C}_\text{R}))}$	$\xleftarrow{Y_e, c_2, \text{C}_\text{R}}$	$c_2 \leftarrow \mathcal{E}(\text{sk}_2, m_2)$
$\text{PRK}_{2e} \leftarrow \text{HKDF-Extract}(\text{""}, {Y_e}^{x_e})$		
$\text{TH}_2 \leftarrow \mathcal{H}(Y_e, \text{C}_\text{R}, \mathcal{H}(m_1))$		
$\text{sk}_2 \leftarrow \text{HKDF-Expand}(\text{PRK}_{2e}, 0, \text{TH}_2, \ell_{\text{plaintext}})$		
set $m_2 \leftarrow \mathcal{D}(\text{sk}_2, c_2)$		
parse m_2 as $(\text{ID}_\text{R} \Vert t_2 \Vert \text{EAD}_2)$		
$\text{CTX}_2 \leftarrow (\text{ID}_\text{R} \Vert \text{TH}_2 \Vert Y_s \Vert \text{EAD}_2)$		
$\text{salt}_{3e2m} \leftarrow \text{HKDF-Expand}(\text{PRK}_{2e}, 1, \text{TH}_2, \ell_{\text{hash}})$		
$\text{PRK}_{3e2m} \leftarrow \text{HKDF-Extract}(\text{salt}_{3e2m}, {Y_s}^{x_e})$		
$t_2' \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 2, \text{CTX}_2, \ell_{\text{MAC}})$		
if $t_2' \neq t_2$: **return** $\perp$		
$\text{TH}_3 \leftarrow \mathcal{H}(\text{TH}_2, m_2)$		
$\text{sk}_3 \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 3, \text{TH}_3, \ell_{\text{key}})$		
$\text{IV}_3 \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 4, \text{TH}_3, \ell_{\text{IV}})$		
$\text{salt}_{4e3m} \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 5, \text{TH}_3, \ell_{\text{hash}})$		
$\text{PRK}_{4e3m} \leftarrow \text{HKDF-Extract}(\text{salt}_{4e3m}, {Y_e}^{x_s})$		
accepted $\leftarrow 1$		
$\text{CTX}_3 \leftarrow (\text{ID}_\text{I} \Vert \text{TH}_3 \Vert X_s \Vert \text{EAD}_3)$		
$t_3 \leftarrow \text{HKDF-Expand}(\text{PRK}_{4e3m}, 6, \text{CTX}_3, \ell_{\text{MAC}})$		
$m_3 \leftarrow (\text{ID}_I \Vert t_3 \Vert \text{EAD}_3)$		
$c_3 \leftarrow \mathcal{E}'(\text{sk}_3, \text{IV}_3, m_3)$	$\xrightarrow{c_3}$	$\underline{\text{RespRun2}(\text{ID}, \texttt{st}, \texttt{peerpk}, c_3)}$
		$\text{TH}_3 \leftarrow \mathcal{H}(\text{TH}_2, m_2)$
		$\text{sk}_3 \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 3, \text{TH}_3, \ell_{\text{key}})$
		$\text{IV}_3 \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 4, \text{TH}_3, \ell_{\text{IV}})$
		set $m_3 \leftarrow \mathcal{D}'(\text{sk}_3, \text{IV}_2, c_3)$
		if $m_3 = \perp$: **return** $\perp$
		parse m_3 as $(\text{ID}_\text{I} \Vert t_3 \Vert \text{EAD}_3)$
		$X_s \leftarrow \texttt{peerpk}[\text{ID}_\text{I}]$
		$\text{salt}_{4e3m} \leftarrow \text{HKDF-Expand}(\text{PRK}_{3e2m}, 5, \text{TH}_3, \ell_{\text{hash}})$
		$\text{PRK}_{4e3m} \leftarrow \text{HKDF-Extract}(\text{salt}_{4e3m}, {X_s}^{y_e})$
		accepted $\leftarrow 1$
		$\text{CTX}_3 \leftarrow (\text{ID}_\text{I} \Vert \text{TH}_3 \Vert X_s \Vert \text{EAD}_3)$
		$t_3' \leftarrow \text{HKDF-Expand}(\text{PRK}_{4e3m}, 6, \text{CTX}_3, \ell_{\text{MAC}})$
		if $t_3' \neq t_3$: **return** $\perp$
$\text{TH}_4 \leftarrow \mathcal{H}(\text{TH}_3, m_3)$		$\text{TH}_4 \leftarrow \mathcal{H}(\text{TH}_3, m_3)$
$\text{PRK}_{\text{out}} \leftarrow \text{HKDF-Expand}(\text{PRK}_{4e3m}, 7, \text{TH}_4, \ell_{\text{hash}})$		$\text{PRK}_{\text{out}} \leftarrow \text{HKDF-Expand}(\text{PRK}_{4e3m}, 7, \text{TH}_4, \ell_{\text{hash}})$
terminated $\leftarrow 1$		**terminated** $\leftarrow 1$
$\text{SK} \leftarrow \text{PRK}_{\text{out}}$		$\text{SK} \leftarrow \text{PRK}_{\text{out}}$

Figure 5.3: EDHOC Protocol in the Static-Static Method

We describe in Figure 5.4 the security game introduced in [DG20] following the framework by Bellare *et al.* [BR06]. After initializing the game, the adversary $\mathcal{A}$ is given multiple access to the following queries:

- NewUser: Generates a new user by generating a new pair of keys.
- Send: Controls activation and message processing of sessions
- SessionKeyReveal: Reveals the session key of a terminated session.
- LongTermKeyReveal: Corrupts a user and reveals its long term secret key.
- Test: Provides a real-or-random challenge on the session key of the queried session.

Then, the adversary makes a single call to the Finalize algorithm, which returns the result of the predicate $[b' = b]$, where b' is the guess of $\mathcal{A}$ and b is the challenge bit, after succeeding through the following predicates:

Sound: ensures at most two sessions share the same `sid`. Once a couple of sessions is detected, the predicate checks if both of them have their `status` accepted, one session user ID is the `peerid` of the other session, with different `role` and the same final session key SK. If one of these property is not verified, the adversary breaks the soundness.

Fresh: detects trivially attacked sessions. First, it ensures that neither the session key is revealed nor any of the peers of the session is corrupted before the acceptance time t_{acc}. In a second time, the Fresh predicate ensures that the partnered session is neither tested nor revealed. If such a session is detected, we set the answer bit b' as 0.

The advantage of an adversary $\mathcal{A}$ against the key privacy is its bias in guessing b, from the random choice: $\mathsf{Adv}^{\mathsf{kp-ake}}(\mathcal{A}) = \Pr[b' = b] - 1/2$. Therefore, in Figure 5.5 we give a formalized description of the EDHOC protocol compliant with the security game made in Figure 5.4. The protocol is analyzed in the random oracle model, therefore, HKDF can be substituted by respective random oracles.

Theorem 5.1

The above EDHOC protocol satisfies the key privacy property under the Gap Diffie-Hellman problem in the Random Oracle model, and the injectivity of $(\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\mathsf{Adv}^{\mathsf{kp-ake}}_{\mathsf{EDHOC}}(t; q_{\mathsf{RO}}, n_\sigma, N)$ upper-bounded by

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \frac{{q_{\mathsf{RO}}}^2 + 4}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_0. This game is the key privacy security game $G^{\mathsf{kp-ake}}_{\mathsf{AKE},\mathcal{A}}$ (defined in Figure 5.4) played by $\mathcal{A}$ using the KGen, Activate and Run algorithms (defined in Figure 5.5). The KGen algorithm generates a long term pair of key, calling Activate with a user with identity u, $\mathcal{A}$ creates its i-th session with u, denoted π^i_u.

$$\Pr[\mathsf{Succ}_0] = \Pr[G^{\mathsf{kp-ake}}_{\mathsf{AKE},\mathcal{A}}],$$

where the event Succ means $b' = b$.

We stress that in this security model, with Perfect Forward Secrecy, we use the weak definition of corruption, meaning that a query to LongTermKeyReveal only reveals the long-term key, while the ephemeral key remains unrevealed. We say a party/session is non-corrupted if no query to LongTermKeyReveal has been made before the time of acceptance t_{acc}, where we consider each block (InitRun1, InitRun2, RespRun1, RespRun2) as atomic. Then corruptions can only happen between two calls to simulated players.

Initialize ()

1. time $\leftarrow 0$
2. users $\leftarrow 0$
3. $b \leftarrow\$ \{0,1\}$

NewUser ()

1. users $\leftarrow$ users $+ 1$
2. $(\mathsf{pk}_{\mathsf{users}}, \mathsf{sk}_{\mathsf{users}}) \leftarrow\$$ KGen
3. $\mathsf{revltk}_{\mathsf{users}} \leftarrow \infty$
4. $\mathtt{peerpk}[\mathsf{users}] \leftarrow \mathsf{pk}_{\mathsf{users}}$
5. **return** $\mathsf{pk}_{\mathsf{users}}$

LongTermKeyReveal(u)

1. time $\leftarrow$ time $+ 1$
2. $\mathsf{revltk}_u \leftarrow$ time
3. **return** sk_u

SessionKeyReveal(u, i)

1. **if** $\pi_u^i = \perp$ **or**
   $\pi_u^i.\mathtt{status} \neq$ **accepted** :
2. **return** $\perp$
3. $\pi_u^i.\mathsf{revealed} \leftarrow$ **true**
4. **return** $\pi_u^i.\mathsf{SK}$

Finalize(b')

1. **if** $\neg$Sound :
2. **return** 1
3. **if** $\neg$Fresh :
4. $b' \leftarrow 0$
5. **return** $[b = b']$

Send(u, i, m)

1. **if** $\pi_u^i = \perp$:
2. $(\mathtt{peerid}, \mathtt{role}) \leftarrow m$
3. $(\pi_u^i, m') \leftarrow\$$ Activate$(u, \mathsf{sk}_u, \mathtt{peerid}, \mathtt{peerpk}, \mathtt{role})$
4. $\pi_u^i.t_{acc} \leftarrow 0$
5. **else** :
6. $(\pi_u^i, m') \leftarrow\$$ Run$(u, \mathsf{sk}_u, \pi_u^i, \mathtt{peerpk}, \mathtt{role})$
7. **if** $\pi_u^i.\mathtt{status} =$ **accepted** :
8. time $\leftarrow$ time $+ 1$
9. $\pi_u^i.t_{acc} \leftarrow$ time
10. **return** m'

Test(u, i)

1. **if** $\pi_u^i = \perp$ **or**
   $\pi_u^i.\mathtt{status} \neq$ **acceptedor** $\pi_u^i.\mathsf{tested}$
2. **return** $\perp$
3. $\pi_u^i.\mathsf{tested} \leftarrow$ **true**
4. $T \leftarrow T \cup \{\pi_u^i\}$
5. $k_0 \leftarrow \pi_u^i.\mathsf{SK}$
6. $k_1 \leftarrow\$$ KE.KS
7. **return** k_b

Sound

1. **if** $\exists$ distinct $\pi_u^i, \pi_v^j, \pi_w^k$ with $\pi_u^i.\mathsf{sid} = \pi_v^j.\mathsf{sid} = \pi_w^k.\mathsf{sid}$:
2. **return false**
3. **if** $\exists \pi_u^i, \pi_v^j$ with
4. $\pi_u^i.\mathtt{status} = \pi_v^j.\mathtt{status} =$ **accepted**
5. **and** $\pi_u^i.\mathsf{sid} = \pi_v^j.\mathsf{sid}$
6. **and** $\pi_u^i.\mathtt{peerid} = u$ **and** $\pi_v^j.\mathtt{peerid} = v$
7. **and** $\pi_u^i.\mathtt{role} \neq \pi_v^j.\mathtt{role}$, but $\pi_u^i.\mathsf{SK} \neq \pi_v^j.\mathsf{SK}$
8. **return false**
9. **return true**

Fresh

1. $\forall \pi_u^i \in T$
2. **if** $\pi_u^i.\mathsf{revealed}$
   or $\mathsf{revltk}_{\pi_u^i.\mathtt{peerid}} < \pi_u^i.t_{acc}$:
3. **return false**
4. **if** $\exists \pi_v^j \neq \pi_u^i$**s.t.**
   $\pi_u^i.\mathsf{sid} = \pi_u^i.\mathsf{sid}$ **and**
   $(\pi_v^j.\mathsf{tested}$ **or** $\pi_v^j.\mathsf{revealed})$:
5. **return false**
6. **return true**

Figure 5.4: Authenticated Key Exchange Key Privacy Security Game $G^{\mathsf{kp-ake}}_{\mathsf{AKE},\mathcal{A}}$

KGen()

1. $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$
2. $\mathsf{pk} \leftarrow g^{\mathbb{Z}_p}$
3. **return** $(\mathsf{pk}, \mathsf{sk})$

Activate(ID, $role$)

1. $\texttt{role} \leftarrow role$
2. $\texttt{status} \leftarrow$ **running**
3. **if** $role =$ initiator :
4. $\quad m' \leftarrow \mathsf{InitRun1}(\mathsf{ID})$
5. **else** $m' \leftarrow \perp$
6. **return** m

InitRun1(ID_I)

1. $x_e \leftarrow\!\!\$\ \mathbb{Z}_p$
2. $X_e \leftarrow g^{x_e}$
3. $\mathsf{C}_\mathsf{I} \leftarrow\!\!\$\ \{0,1\}^{nI}$
4. $\mathsf{st} \leftarrow (\mathsf{C}_\mathsf{I}, X_e, x_e)$
5. $m_1 \leftarrow (\mathsf{C}_\mathsf{I} \| X_e \| \mathsf{EAD}_1)$
6. **return** m_1

InitRun2($\mathsf{ID}_\mathsf{I}, x_s, (Y_e, c_2, \mathsf{C}_\mathsf{R})$)

1. $\mathsf{PRK}_{2e} \leftarrow \mathsf{RO}_T(\text{''''}, {Y_e}^{x_e})$
2. $\mathsf{TH}_2 \leftarrow \mathcal{H}(Y_e, \mathsf{C}_\mathsf{R}, \mathcal{H}(m_1))$
3. $\mathsf{sk}_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_{\mathsf{plaintext}})$
4. $m_2 \leftarrow \mathcal{D}(\mathsf{sk}_2, c_2)$
5. $(\mathsf{ID}_\mathsf{R} \| t_2 \| \mathsf{EAD}_2) \leftarrow m_2$
6. $\mathsf{CTX}_2 \leftarrow (\mathsf{ID}_\mathsf{R} \| \mathsf{TH}_2 \| Y_s \| \mathsf{EAD}_2)$
7. $\mathsf{salt}_{3e2m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
8. $\mathsf{PRK}_{3e2m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{3e2m}, {Y_s}^{x_e})$
9. $t_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{MAC}})$
10. **if** $t_2' \neq t_2$:
11. $\quad \texttt{status} \leftarrow$ **rejected**
12. $\quad$ **return** $\perp$
13. $\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2)$
14. $\mathsf{sk}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
15. $\mathsf{IV}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{IV}})$
16. $\mathsf{salt}_{4e3m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
17. $\mathsf{PRK}_{4e3m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{4e3m}, {Y_e}^{x_s})$
18. $\texttt{status} \leftarrow$ **accepted**
19. $\mathsf{CTX}_3 \leftarrow (\mathsf{ID}_\mathsf{I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3)$
20. $t_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{MAC}})$
21. $m_3 \leftarrow (\mathsf{ID}_\mathsf{I} \| t_3 \| \mathsf{EAD}_3)$
22. $c_3 \leftarrow \mathcal{E}'(\mathsf{sk}_3, \mathsf{IV}_3, m_3)$
23. $\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3)$
24. $\mathsf{PRK}_{\mathsf{out}} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$
25. $\texttt{status} \leftarrow$ **terminated**
26. $\mathsf{SK} \leftarrow \mathsf{PRK}_{\mathsf{out}}$
27. **return** c_3

Run(ID, sk, peerpk, m)

1. **if** $\texttt{status} \neq$ **running** :
2. $\quad$ **return** $\perp$
3. **if** $\texttt{role} =$ initiator :
4. $\quad m' \leftarrow \mathsf{InitRun2}(\mathsf{ID}, \mathsf{sk}, m)$
5. **elseif** $\mathsf{sid} = \perp$:
6. $\quad m' \leftarrow \mathsf{RespRun1}(\mathsf{ID}, \mathsf{sk}, m)$
7. **else** :
8. $\quad m' \leftarrow \mathsf{RespRun2}(\mathsf{ID}, \texttt{peerpk}, m)$
9. **return** m

RespRun1($\mathsf{ID}_\mathsf{R}, y_s, m_1 = (X_e, \mathsf{C}_\mathsf{I}, \mathsf{EAD}_1)$)

1. $y_e \leftarrow\!\!\$\ \mathbb{Z}_p$
2. $Y_e \leftarrow g^{y_e}$
3. $\mathsf{C}_\mathsf{R} \leftarrow\!\!\$\ \{0,1\}^{nI}$
4. $\mathsf{sid} \leftarrow (\mathsf{C}_\mathsf{I}, \mathsf{C}_\mathsf{R}, X_e, Y_e)$
5. $\mathsf{PRK}_{2e} \leftarrow \mathsf{RO}_T(\text{''''}, {X_e}^{y_e})$
6. $\mathsf{sk}_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_{\mathsf{plaintext}})$
7. $\mathsf{salt}_{3e2m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
8. $\mathsf{PRK}_{3e2m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{3e2m}, {X_e}^{y_s})$
9. $\mathsf{TH}_2 \leftarrow \mathcal{H}(Y_e, \mathsf{C}_\mathsf{R}, \mathcal{H}(m_1))$
10. $\mathsf{CTX}_2 \leftarrow (\mathsf{ID}_\mathsf{R} \| \mathsf{TH}_2 \| Y_s \| \mathsf{EAD}_2)$
11. $t_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{MAC}})$
12. $m_2 \leftarrow (\mathsf{ID}_\mathsf{R} \| t_2 \| \mathsf{EAD}_2)$
13. $c_2 \leftarrow \mathcal{E}(\mathsf{sk}_2, m_2)$
14. **return** $(Y_e, c_2, \mathsf{C}_\mathsf{R})$

RespRun2(ID_R, peerpk, c_3)

1. $\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2)$
2. $\mathsf{sk}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
3. $\mathsf{IV}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{IV}})$
4. $m_3 \leftarrow \mathcal{D}'(\mathsf{sk}_3, \mathsf{IV}_3, c_3)$
5. **if** $m_3 = \perp$:
6. $\quad \texttt{status} \leftarrow$ **rejected**
7. $\quad$ **return** $\perp$
8. $(\mathsf{ID}_\mathsf{I} \| t_3 \| \mathsf{EAD}_3) \leftarrow m_3$
9. $X_s \leftarrow \texttt{peerpk}[\mathsf{ID}_\mathsf{I}]$
10. $\mathsf{salt}_{4e3m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
11. $\mathsf{PRK}_{4e3m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{4e3m}, {X_s}^{y_e})$
12. $\texttt{status} \leftarrow$ **accepted**
13. $\mathsf{CTX}_3 \leftarrow (\mathsf{ID}_\mathsf{I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3)$
14. $t_3' \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{MAC}})$
15. **if** $t_3' \neq t_3$:
16. $\quad \texttt{status} \leftarrow$ **rejected**
17. $\quad$ **return** $\perp$
18. $\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3)$
19. $\mathsf{PRK}_{\mathsf{out}} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$
20. $\texttt{status} \leftarrow$ **terminated**
21. $\mathsf{SK} \leftarrow \mathsf{PRK}_{\mathsf{out}}$
22. **return** $\perp$

Figure 5.5: Formalized description of the EDHOC protocol

Game G_1. In this game, we simulate the random oracles by lists that are empty at the beginning of the game. As $\mathrm{RO}_{\mathcal{T}}$ and $\mathcal{H}$ always return a digest of size ℓ_{hash}, we simply use the simulation oracle $\mathrm{SO}_{\mathcal{T}}$ and $\mathrm{SO}_{\mathcal{H}}$ respectively. However, RO_P may return values of several lengths: $\ell_{\mathsf{plaintext}}$ for the one-time key encrypting the responder first message, ℓ_{hash} for the salt values and the session key, ℓ_{key} and ℓ_{IV} for the AEAD key length and Initialisation Vector respectively, and ℓ_{MAC} for the tags. We thus define a simulation oracle by digest size: $\mathrm{SO}_P^{\mathsf{size}}$, for size in $\{\ell_{\mathsf{plaintext}}, \ell_{\mathsf{hash}}, \ell_{\mathsf{key}}, \ell_{\mathsf{IV}}, \ell_{\mathsf{MAC}}\}$

The simulation oracles SO_P and $\mathrm{SO}_{\mathcal{H}}$ work as the usual way of simulating the answer with a new random answer for any new query, and the same answer if the same query is asked again. For the simulation oracles $\mathrm{SO}_{\mathcal{T}}$, the oracle consists in a list that contains elements of the form $(\mathsf{str}, Z, (X, Y); \lambda)$, where when first set, either Z or (X, Y) is non-empty. Indeed, when making a call to a random oracle, the official query is of the form (str, Z), where str is any bit string, that can be empty or a pseudo-random key, and Z is a Diffie-Hellman value. Then, the simulator checks in the list for an entry matching with $(\mathsf{str}, Z, *; \lambda)$. If such an element is found, one outputs λ, otherwise one randomly set $\lambda \leftarrow\$ \{0,1\}^{\kappa}$ and append $(\mathsf{str}, Z, \perp; \lambda)$ to the list. But later, the simulator will also ask queries of the form $(\mathsf{str}, (X, Y))$, where (X, Y) is a pair of group elements. Then one checks in the list for an entry matching with either $(\mathsf{str}, *, (X, Y); \lambda)$ or $(\mathsf{str}, Z, *; \lambda)$ such that $\mathsf{DDH}(g, X, Y, Z) = 1$. If such an element is found, one outputs λ, otherwise one randomly set $\lambda \leftarrow\$ \{0,1\}^{\kappa}$ and append $(\mathsf{str}, \perp, (X, Y); \lambda)$ to the list. When such new kinds of elements exist in the list, for the first kind of queries (str, Z), one checks in the list for an entry matching with either $(\mathsf{str}, Z, *; \lambda)$ as before, or $(\mathsf{str}, *, (X, Y); \lambda)$ such that $\mathsf{DDH}(g, X, Y, Z) = 1$. We detail in Figure 5.6 the functioning of those oracles, and the modifications made to the simulation.

Thanks to the DDH oracle, this simulation is perfect, and is thus indistinguishable to the adversary:

$$\Pr[\mathsf{Succ}_1] = \Pr[\mathsf{Succ}_0]$$

Game G_2. In order to prevent collisions in the future PRK generation, we modify the simulation oracles $\mathrm{SO}_{\mathcal{T}}, \mathrm{SO}_P^{\ell_{\mathsf{hash}}}$ and $\mathrm{SO}_{\mathcal{H}}$, such that if a collision occurs, the simulator stops. We therefore need to determine the probability of a collision, to bound the probability for an adversary to distinguish this game from the previous one. To do so, we rely on the birthday paradox. By denoting $q_{\mathrm{SO}_{\mathcal{T}}}, q_{\mathrm{SO}_P^{\ell_{\mathsf{hash}}}}, q_{\mathrm{SO}_{\mathcal{H}}}$ the amount of queries made to oracles $\mathrm{SO}_{\mathcal{T}}, \mathrm{SO}_P^{\ell_{\mathsf{hash}}}, \mathrm{SO}_{\mathcal{H}}$ respectively, the birthday paradox bound gives:

$$\Pr[\mathsf{Succ}_2] - \Pr[\mathsf{Succ}_1] \leq \frac{{q_{\mathrm{SO}_{\mathcal{T}}}}^2 + {q_{\mathrm{SO}_P^{\ell_{\mathsf{hash}}}}}^2 + {q_{\mathrm{SO}_{\mathcal{H}}}}^2}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_3. One can note that thanks to the above simulation of the random oracles, the simulator does not need anymore to compute Diffie-Hellman values. Then, for every simulated player, the simulator generates X_e or Y_e at random in the group, and the simulation is still performed as in the previous game. As corruption queries only reveal long-term secret, still known to the simulator, the view of the adversary is perfectly indistinguishable of the previous game and we have:

$$\Pr[\mathsf{Succ}_3] = \Pr[\mathsf{Succ}_2]$$

Game G_4. In this game, when simulating any **initiator** receiving a forged tuple $(Y_e, c_2, \mathsf{C_R})$ from the adversary in the name of a **non-corrupted user**, one simulates PRK_{3e2m} thanks to a private oracle $\mathrm{SO}_{\mathrm{PRK}_{3e2m}}$, which makes it perfectly unpredictable to the adversary. If the pair $(Y_e, \mathsf{C_R})$ is forged, TH_2 and salt_{3e2m} are different from the values obtained by a possibly simulated responder, thanks to the absence of collisions as they are respectively computed using $\mathrm{SO}_{\mathcal{H}}$ and $\mathrm{SO}_P^{\ell_{\mathsf{hash}}}$. Otherwise, sk_2 is not modified. So if the ciphertext c_2 is forged,

$\mathsf{SO}_{\mathcal{T}}(\mathsf{str}, \mathsf{input})$

1. **if** $\mathsf{len}(\mathsf{input}) = 1$:
2. $\quad Z \leftarrow \mathsf{input}$
3. $\quad$ **if** $\exists(\mathsf{str}, Z, *; \lambda) \in \mathsf{SO}_{\mathcal{T}}$:
4. $\quad\quad$ **return** λ
5. $\quad$ **else** :
6. $\quad\quad$ **if** $\exists(\mathsf{str}, \bot, (X, Y); \lambda) \in \mathsf{SO}_{\mathcal{T}}$ s.t. $\mathsf{DDH}(X, Y, Z) = 1$:
$\quad\quad\quad$ // update $\mathsf{SO}_{\mathcal{T}}$
7. $\quad\quad\quad \mathsf{SO}_{\mathcal{T}}{}^{-1}[\lambda] \leftarrow (\mathsf{str}, Z, (X, Y); \lambda)$
8. $\quad\quad\quad$ **return** λ
9. $\quad\quad$ **else** :
10. $\quad\quad\quad \lambda \leftarrow\$ \{0,1\}^{\kappa}$
11. $\quad\quad\quad \mathsf{SO}_{\mathcal{T}} \leftarrow \mathsf{SO}_{\mathcal{T}} \cup \{(\mathsf{str}, Z, \bot; \lambda)\}$
12. $\quad\quad\quad$ **return** λ
13. **else**
$\quad$ // input $= (X, Y)$, only by the simulator
14. $\quad (X, Y) \leftarrow \mathsf{input}$
15. $\quad$ **if** $\exists(\mathsf{str}, *, (X, Y); \lambda) \in \mathsf{SO}_{\mathcal{T}}$:
16. $\quad\quad$ **return** λ
17. $\quad$ **else** :
18. $\quad\quad$ **if** $\exists(\mathsf{str}, Z, \bot; \lambda) \in \mathsf{SO}_{\mathcal{T}}$ s.t. $\mathsf{DDH}(X, Y, Z) = 1$:
19. $\quad\quad\quad \mathsf{SO}_{\mathcal{T}}{}^{-1}[\lambda] \leftarrow (\mathsf{str}, Z, (X, Y); \lambda)$
20. $\quad\quad\quad$ **return** λ
21. $\quad\quad$ **else** :
22. $\quad\quad\quad \lambda \leftarrow\$ \{0,1\}^{\kappa}$
23. $\quad\quad\quad \mathsf{SO}_{\mathcal{T}} \leftarrow \mathsf{SO}_{\mathcal{T}} \cup \{(\mathsf{str}, \bot, (X, Y); \lambda)\}$
24. $\quad\quad\quad$ **return** λ

$\mathsf{RespRun1}(\mathsf{ID}_{\mathsf{R}}, y_s, m)$

5. $\mathsf{PRK}_{2e} \leftarrow \mathsf{SO}_{\mathcal{T}}(\text{''''}, {X_e}^{y_e})$
6. $\mathsf{sk}_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_{\mathsf{plaintext}})$
7. $\mathsf{salt}_{3e2m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
8. $\mathsf{PRK}_{3e2m} \leftarrow \mathsf{SO}_{\mathcal{T}}(\mathsf{salt}_{3e2m}, {X_e}^{y_s})$
9. $\mathsf{TH}_2 \leftarrow \mathsf{SO}_{\mathcal{H}}(Y_e, \mathsf{C}_{\mathsf{R}}, \mathsf{SO}_{\mathcal{H}}(m_1))$
$\quad$:
11. $t_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{MAC}})$

$\mathsf{InitRun2}(\mathsf{ID}_{\mathsf{I}}, x_s, m)$

1. $\mathsf{PRK}_{2e} \leftarrow \mathsf{SO}_{\mathcal{T}}(\text{''''}, {Y_e}^{x_e})$
2. $\mathsf{TH}_2 \leftarrow \mathsf{SO}_{\mathcal{H}}(Y_e, \mathsf{C}_{\mathsf{R}}, \mathsf{SO}_{\mathcal{H}}(m_1))$
3. $\mathsf{sk}_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_{\mathsf{plaintext}})$
$\quad$:
7. $\mathsf{salt}_{3e2m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
8. $\mathsf{PRK}_{3e2m} \leftarrow \mathsf{SO}_{\mathcal{T}}(\mathsf{salt}_{3e2m}, {Y_s}^{x_e})$
9. $t_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{MAC}})$
$\quad$:
13. $\mathsf{TH}_3 \leftarrow \mathsf{SO}_{\mathcal{H}}(\mathsf{TH}_2, m_2)$
14. $\mathsf{sk}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
15. $\mathsf{IV}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{IV}})$
16. $\mathsf{salt}_{4e3m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
17. $\mathsf{PRK}_{4e3m} \leftarrow \mathsf{SO}_{\mathcal{T}}(\mathsf{salt}_{4e3m}, {Y_e}^{x_s})$
$\quad$:
20. $t_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{MAC}})$
$\quad$:
23. $\mathsf{TH}_4 \leftarrow \mathsf{SO}_{\mathcal{H}}(\mathsf{TH}_3, m_3)$
24. $\mathsf{PRK}_{\mathsf{out}} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$

$\mathsf{RespRun2}(\mathsf{ID}_{\mathsf{R}}, \mathtt{peerpk}, c_3)$

1. $\mathsf{TH}_3 \leftarrow \mathsf{SO}_{\mathcal{H}}(\mathsf{TH}_2, m_2)$
2. $\mathsf{sk}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
3. $\mathsf{IV}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{IV}})$
$\quad$:
10. $\mathsf{salt}_{4e3m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
11. $\mathsf{PRK}_{4e3m} \leftarrow \mathsf{SO}_{\mathcal{T}}(\mathsf{salt}_{4e3m}, \mathtt{peerpk}[\mathsf{ID}_{\mathsf{I}}]^{y_e})$
$\quad$:
14. $t'_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{MAC}})$
$\quad$:
18. $\mathsf{TH}_4 \leftarrow \mathsf{SO}_{\mathcal{H}}(\mathsf{TH}_3, m_3)$
19. $\mathsf{PRK}_{\mathsf{out}} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$

Figure 5.6: Description of $\mathsf{SO}_{\mathcal{T}}$ list queries and modifications to the simulation

thanks to the injective property of the encryption scheme $(\mathcal{E}, \mathcal{D})$ when the key is fixed, m_2, and by consequent TH_3 and salt_{4e3m} are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{3e2m} with respect to the public oracle answer, the adversary must have asked $\mathsf{SO}_{\mathcal{T}}$ on the correct Diffie-Hellman value ${X_e}^{y_s}$. We denote the event F_1, that query ${X_e}^{y_s}$ is asked whereas y_s is the long-term secret key of a non-corrupted user and X_e has been generated by the simulator. If this event happens (which can easily be checked as the simulator knows y_s), one stops the simulation:

$$|\Pr[\mathsf{Succ}_4] - \Pr[\mathsf{Succ}_3]| \leq \Pr[F_1].$$

Game $G_{4'}$. We now provide an upper-bound on $\Pr[F_1]$: given a GDH challenge $(X = g^x, Y = g^y)$, one simulates all the X_e as $X_e = X \cdot g^r$, for random $r \leftarrow\!\!\$\ \mathbb{Z}_p$, but chooses one user to set $Y_s = Y$. Even if y_s is therefore not known, simulation is still feasible as the simulator can make query to the $\mathsf{SO}_{\mathcal{T}}$ oracle with input (X_e, Y_s). Then, one can still answer all the corruption queries, excepted for that user. But anyway, if F_1 happens on that user, this user must be non-corrupted at that time: one has solved the GDH problem, and one can stop the simulation. If the guess on the user is incorrect, one can also stop the simulation: $\Pr[F_1] \leq N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}})$, where N is the number of users in the system.

Game G_5. In this game, when simulating any **responder** receiving a forged message m_1 from the adversary in the name of a **non-corrupted user**, still non-corrupted when sending c_3 to RespRun2, one simulates PRK_{4e3m} thanks to a private oracle $\mathsf{SO}_{\mathsf{PRK}_{4e3m}}$, which makes it perfectly unpredictable to the adversary. Since m_1 is forged, thanks to the absence of collisions, TH_2,TH_3, and salt_{4e3m} are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{4e3m} with respect to the public oracle answer, the adversary must have asked $\mathsf{SO}_{\mathcal{T}}$ on the correct Diffie-Hellman value ${Y_e}^{x_s}$. We denote the event F_2, that query ${Y_e}^{x_s}$ is asked whereas x_s is the long-term secret key of a non-corrupted user and Y_e has been generated by the simulator. If this event happens (which can easily be checked as the simulator knows x_s), one stops the simulation:

$$|\Pr[\mathsf{Succ}_5] - \Pr[\mathsf{Succ}_4]| \leq \Pr[F_2].$$

Game $G_{5'}$. We now provide an upper-bound on $\Pr[F_2]$: given a GDH challenge $(X = g^x, Y = g^y)$, one simulates all the Y_e as $Y_e = Y \cdot g^{r'}$, for random $r' \leftarrow\!\!\$\ \mathbb{Z}_p$, but chooses one user to set $X_s = X$. Then, one can still answer all the corruption queries, excepted for that user. But anyway, if F_2 happens on that user, this user must be non-corrupted at that time: one has solved the GDH problem, and one can stop the simulation. If the guess on the user is incorrect, one can also stop the simulation: $\Pr[F_2] \leq N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}})$.

Game G_6. In this game, we simulate the key generation of PRK_{2e}, for all the passive sessions (m_1 received by a simulated responder comes from a simulated initiator, or $(Y_e, c_2, \mathsf{C_R})$ received by a simulated initiator comes from a simulated responder, and both used the same m_1 as first message), thanks to a private oracle $\mathsf{SO}_{\mathsf{PRK}_{2e}}$, acting in the same vein as $\mathsf{SO}_{\mathcal{T}}$, but not available to the adversary. This makes a difference with the previous game if the key PRK_{2e} has also been generated by asking $\mathsf{SO}_{\mathcal{T}}$ on the correct Diffie-Hellman value $Z = g^{x_e y_e}$. We denote by F_3 the latter event, and stop the simulation in such a case:

$$|\Pr[\mathsf{Succ}_6] - \Pr[\mathsf{Succ}_5]| \leq \Pr[F_3]$$

Game $G_{6'}$. We now provide an upper-bound on $\Pr[F_3]$. Given a GDH challenge $(X = g^x, Y = g^y)$, one simulates all the X_e as $X_e = X \cdot g^r$, for random $r \leftarrow\!\!\$\ \mathbb{Z}_p$, and all the Y_e as $Y_e = Y \cdot g^{r'}$, for random $r' \leftarrow\!\!\$\ \mathbb{Z}_p$. As the key PRK_{2e} does not depend on the session

context, any query Z to the SO_T oracle can make F_3 occurs on any of the passive session pairs $(X_e = X \cdot g^r, Y_e = Y \cdot g^{r'})$, we upper-bound the number by n_σ. Hence, q_{RO} DDH-oracle queries might be useful to detect F_3 on an input $Z = \mathrm{CDH}(X_e, Y_e) = g^{xy} \cdot X^{r'} \cdot Y^r \cdot g^{rr'}$, solving the GDH challenge (X, Y):

$$\Pr[F_3] \leq \mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathrm{RO}}).$$

Game G_7. In this game, when simulating any **initiator** receiving the second message $(Y_e, c_2, \mathsf{C_R})$, from the adversary in the name of a **non-corrupted user**, one simulates PRK_{3e2m} thanks to a private oracle $\mathrm{SO}_{\mathrm{PRK}_{3e2m}}$. This makes a difference with the previous game only if this is a passive session, in which case PRK_{2e} is unpredictable, and thus different from the public one excepted with probability $2^{-\ell_{\mathrm{hash}}}$. As there are no collision, salt_{3e2m} is different from the value obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{3e2m} with respect to the public oracle answer, the adversary must have asked SO_T on the correct Diffie-Hellman value $X_e^{y_s}$, which is not possible as event F_1 stops the simulation:

$$|\Pr[\mathrm{Succ}_7] - \Pr[\mathrm{Succ}_6]| \leq \frac{1}{2^{\ell_{\mathrm{hash}}}}.$$

Game G_8. In this game, when simulating any **initiator** receiving the second message $(Y_e, c_2, \mathsf{C_R})$, from the adversary in the name of a **non-corrupted user**, one simulates PRK_{4e3m} thanks to a private oracle $\mathrm{SO}_{\mathrm{PRK}_{4e3m}}$. In this case, PRK_{3e2m} is unpredictable, as well as salt_{4e3m} and PRK_{4e3m}:

$$\Pr[\mathrm{Succ}_8] = \Pr[\mathrm{Succ}_7].$$

Game G_9. In this game, when simulating any **responder** receiving c_3, from the adversary in the name of a **non-corrupted user**, one simulates PRK_{4e3m} thanks to the private oracle $\mathrm{SO}_{\mathrm{PRK}_{4e3m}}$. This makes a difference with the previous game only if this is not a passive session, in which case PRK_{2e} is unpredictable, and thus different from the public one excepted with probability $2^{-\ell_{\mathrm{hash}}}$. As there are no collision, salt_{3e2m}, PRK_{3e2m}, and salt_{4e3m} are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{4e3m} with respect to the public oracle answer, the adversary must have asked SO_T on the correct Diffie-Hellman value $Y_e^{x_s}$, which is not possible as event F_2 stops the simulation:

$$|\Pr[\mathrm{Succ}_9] - \Pr[\mathrm{Succ}_8]| \leq \frac{1}{2^{\ell_{\mathrm{hash}}}}.$$

Game G_{10}. In this game, for any fresh session, one simulates $\mathrm{PRK}_{\mathrm{out}}$ thanks to the private oracle $\mathrm{SO}_{\mathrm{PRK}_{\mathrm{out}}}$. A session being fresh means that no corruption of the party or of the partner occurred before the time of acceptance: the initiator is not corrupted before receiving $(Y_e, c_2, \mathsf{C_R})$ and the responder is not corrupted before receiving c_3. By consequent, they are not corrupted before PRK_{4e3m} was computed. We have seen above that in those cases, the key PRK_{4e3m} is generated using the private oracle $\mathrm{SO}_{\mathrm{PRK}_{4e3m}}$: it is unpredictable. The use of the private oracle $\mathrm{SO}_{\mathrm{PRK}_{\mathrm{out}}}$ can only be detected if the query PRK_{4e3m} is asked to SO_P:

$$|\Pr[\mathrm{Succ}_{10}] - \Pr[\mathrm{Succ}_9]| \leq \frac{q_{\mathrm{SO}_P^{\ell_{\mathrm{hash}}}}}{2^{\ell_{\mathrm{hash}}}}.$$

Globally, one can note that the gap between the initial and the last games is upper-bounded by

$$\begin{aligned}
&\mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathrm{RO}}) + 2N \cdot \mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, q_{\mathrm{RO}}) \\
&\qquad + \frac{{q_{\mathrm{SO}_T}}^2 + {q_{\mathrm{SO}_P^{\ell_{\mathrm{hash}}}}}^2 + {q_{\mathrm{SO}_{\mathcal{H}}}}^2}{2^{\ell_{\mathrm{hash}}+1}} + \frac{2 + q_{\mathrm{SO}_P^{\ell_{\mathrm{hash}}}}}{2^{\ell_{\mathrm{hash}}}} \\
&\leq \mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathrm{RO}}) + 2N \cdot \mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, q_{\mathrm{RO}}) + \frac{{q_{\mathrm{RO}}}^2 + 4}{2^{\ell_{\mathrm{hash}}+1}}
\end{aligned}$$

Eventually, for all the fresh sessions, in the real case ($b = 0$), the private oracle is used, and outputs a random key, while in the random case ($b = 1$), the session key is random too:

$$\Pr[\mathsf{Succ}_{10}] = \frac{1}{2}.$$

This concludes the proof.

5.3 Explicit authentication

Explicit authentication (or mutual authentication) aims to ensure each participant has the material to compute the final session key (accepts) when the partner terminates. In the EDHOC protocol, this means the responder (resp. the initiator) owns the private long-term key y_s (resp x_s) associated to the long-term public key Y_s (resp. X_s), and the private ephemeral keys, when the partner terminates.

Finalize

1. **return** :

$$\forall \pi_u^i \text{ s.t. } \begin{cases} \pi_u^i.\mathtt{status} = \textbf{terminated} \\ \pi_u^i.t_{acc} < \mathsf{revltk}_{\pi_u^i.\mathtt{peerid}} \end{cases}, \exists \pi_v^j \text{ s.t. } \begin{cases} \pi_u^i.\mathtt{peerid} = v \\ \pi_v^j.\mathtt{peerid} = u \\ \pi_u^i.\mathsf{sid} = \pi_v^j.\mathsf{sid} \\ \pi_u^i.\mathtt{role} \neq \pi_v^j.\mathtt{role} \\ \pi_v^j.\mathtt{status} = \textbf{accepted} \end{cases}$$

Figure 5.7: **Finalize** Function for the Explicit Authentication Security Game

To do so, the responder uses y_s in **RespRun1** to compute PRK_{3e2m} used for the tag t_2 and the key sk_3. In the same way, the initiator uses x_s to compute PRK_{4e3m}, used for the tag t_3. Furthermore, they both have to use their ephemeral keys to compute PRK_{2e}, used for sk_2.

Responder Authentication. Consider a simulated **initiator** receiving a forged message $(Y_e, c_2, \mathsf{C_R})$ from the adversary in the name of a **non-corrupted user**. In such a case, consider the modifications made in the key privacy proof up to the game G_7. Hence, we have replaced the generation of PRK_{3e2m} with a private oracle. Then the advantage of the adversary in breaking the explicit authentication of the responder in this game is bounded by $\frac{1}{2^{\ell_{\mathsf{MAC}}}}$, added to the gap induced by the modifications made up to the game G_7. This leads to the following theorem:

Theorem 5.2

The above EDHOC protocol satisfies the responder authentication under the Gap Diffie-Hellman problem in the Random Oracle model, and the injectivity of $(\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\mathsf{Adv}^{\mathsf{auth-resp}}_{\mathsf{EDHOC}}(t; q_{\mathsf{RO}}, n_\sigma, N)$ upper-bounded by

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \frac{{q_{\mathsf{RO}}}^2 + 2}{2^{\ell_{\mathsf{hash}}+1}} + \frac{1}{2^{\ell_{\mathsf{MAC}}}}$$

Optimal Reduction. One cannot expect more after these three flows, as the adversary can play the role of the responder with known y_e. Without knowing y_s, it just gets stuck to compute PRK_{3e2m} and thus t_2. But it can guess it (with probability $2^{-\ell_{\mathsf{MAC}}}$), breaking authentication. But it will not know SK.

Initiator Authentication. We now consider any **responder** receiving a forged message c_3 from the adversary in the name of a **non-corrupted user**. As above, considering the modifications made in the key privacy proof up to the game G_8, we have replaced the generation of PRK_{4e3m} with a private oracle. Then the advantage of the adversary in breaking the explicit authentication of the initiator in this game is bounded by $\frac{1}{2^{\ell_{\mathsf{MAC}}}}$, added to the gap induced by the modifications made up to the game G_7. This leads to the following theorem:

Theorem 5.3

The above EDHOC protocol satisfies the initiator authentication under the Gap Diffie-Hellman problem in the Random Oracle model, and the injectivity of $(\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\mathsf{Adv}^{\mathsf{auth-init}}_{\mathsf{EDHOC}}(t; q_{\mathsf{RO}}, n_\sigma, N)$ upper-bounded by

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \frac{{q_{\mathsf{RO}}}^2 + 4}{2^{\ell_{\mathsf{hash}}+1}} + \frac{1}{2^{\ell_{\mathsf{MAC}}}}$$

Optimal Reduction. One cannot expect more after these three flows, as the adversary can play the role of the initiator with known x_e. Without knowing x_s, it just gets stuck to compute PRK_{4e3m} and thus t_3. But it can guess it (with probability $2^{-\ell_{\mathsf{MAC}}}$) and encrypt it, as it knows sk_3, breaking authentication. But it will not know SK.

5.4 Identity protection

Let us now consider anonymity, with identity protection. More precisely, we want to show that the initiator's identity ($\mathsf{ID_I}$) is protected against active adversaries, while responder's identity ($\mathsf{ID_R}$) is protected only against passive adversaries.

The values $\mathsf{ID_I}$ and $\mathsf{ID_R}$ are the authentication credentials containing the public authentication keys of I and R, respectively.

Responder's Identity Protection. The value $\mathsf{ID_R}$ is used in the computation of CTX_2 itself used to compute t_2, which together with $\mathsf{ID_R}$ constitute the first part of $m_2 = (\mathsf{ID_R} \| t_2 \| \mathsf{EAD}_2)$ whose encryption is c_2 under sk_2. For the sake of clarity, we set $\mathsf{EAD}_2 =$"" as EAD_2 is independent from the identity of the responder and has no cryptographic purpose. As a responder, the passive adversary can only learn information about $\mathsf{ID_R}$ using the ciphertext c_2. We thus define the responder identity protection experiment as follows:

$\mathbf{Exp}^{\mathsf{ID-resp}-b}_{\mathsf{EDHOC}}(\mathcal{A})$

1. $\mathsf{ID}_{\mathsf{R}_0}, \mathsf{ID}_{\mathsf{R}_1} \leftarrow \mathcal{A}(\texttt{peerid})$
2. $m_1 \leftarrow \mathcal{A}(\mathsf{InitRun1}(.))$
3. $b \leftarrow \{0, 1\}$
4. $\mathsf{ID_R} \leftarrow \mathsf{ID}_{\mathsf{R}_b}$
5. $y_s \leftarrow \mathsf{sk}_{\mathsf{ID_R}}$
6. $(Y_e, c_2, \mathsf{C_R}) \leftarrow \mathsf{RespRun1}(\mathsf{ID_R}, y_s, m_1)$
7. $b' \leftarrow \mathcal{A}(c_2)$
8. **return** $b = b'$

We define the advantage $\mathsf{Adv}^{\mathsf{ID-resp}-b}_{\mathsf{EDHOC}}$ of the adversary in breaking the responder mutual authentication of EDHOC by:

$$\mathsf{Adv}^{\mathsf{ID-resp}-b}_{\mathsf{EDHOC}}(t) = \max_{\mathcal{A}} \left(\left| \Pr[\mathbf{Exp}^{\mathsf{ID-resp}-0}_{\mathsf{EDHOC}}(\mathcal{A}) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ID-resp}-1}_{\mathsf{EDHOC}}(\mathcal{A}) = 1] \right| \right)$$

Theorem 5.4

The above EDHOC protocol protects Responder's Identity under the Gap Diffie-Hellman problem in the Random Oracle model, the injectivity and the semantic security of $\Pi = (\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\mathsf{Adv}^{\mathsf{ID-resp}-b}_{\mathsf{EDHOC}}(t; q_{\mathsf{RO}}, n_\sigma, N)$ upper-bounded by

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \mathsf{Adv}^{\mathsf{IND}}_{\Pi}(t) + \frac{{q_{\mathsf{RO}}}^2 + 2}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_0. This game is $\mathbf{Exp}^{\mathsf{ID-resp}-0}_{\mathsf{EDHOC}}$. The simulated initiator follows the protocol, computes $c_2 = \mathcal{E}(\mathsf{sk}_2, (\mathsf{ID_R} \| t_2))$ and sends Y_e, c_2, C_R to the adversary:

$$\Pr[\mathsf{Succ}_0] = \Pr[\mathbf{Exp}^{\mathsf{ID-resp}-0}_{\mathsf{EDHOC}} = 1]$$

Game G_1. In this game, we applied the modification made from G_0 up to G_6 in the key privacy proof.

$$|\Pr[\mathsf{Succ}_1] - \Pr[\mathsf{Succ}_0]| \leq \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \frac{{q_{\mathsf{RO}}}^2}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_2. In this game, one simulates sk_2 thanks to a private oracle, which makes a difference only if the random PRK_{2e} is asked to the public oracle:

$$|\Pr[\mathsf{Succ}_2] - \Pr[\mathsf{Succ}_1]| \leq \frac{1}{2^{\ell_{\mathsf{hash}}}}$$

Game G_3. In this game, we replace the line 4 of the experiment by $\mathsf{ID_R} \leftarrow \mathsf{ID}_{\mathsf{R}_{1-b}}$, leading to the instanciation of $\mathbf{Exp}^{\mathsf{ID-resp}-1}_{\mathsf{EDHOC}}$. As sk_2 is chosen at random, using the semantic security of the encryption scheme $(\mathcal{E}, \mathcal{D})$, we thus have

$$|\Pr[\mathsf{Succ}_3] - \Pr[\mathsf{Succ}_2]| \leq \mathsf{Adv}^{\mathsf{IND}}_{\Pi}(t)$$

Initiator's Identity Protection. In this case, we expect an active security: we consider the simulation of an initiator interacting with an adversary playing in the name of a non-corrupted user with public long-term key Y_s. We have simulated PRK_{3e2m} with a private oracle, which leads to a private random key sk_3, unless the query has been asked, with the same argument as above.

The value $\mathsf{ID_I}$ is used in the computation of CTX_3 itself used to compute t_3, which together with $\mathsf{ID_I}$ constitute the first part of the message $m_3 = (\mathsf{ID_R} \| t_2 \| \mathsf{EAD}_2)$ whose encryption is c_3 under sk_3. As above, for the sake of generality, we set $\mathsf{EAD}_3 =$"". One can note that the first message m_1 sent by the initiator is independent of $\mathsf{ID_R}$. We therefore start the experiment after the adversary sent his first message $(Y_e, c_2, \mathsf{C_R})$:

$\mathbf{Exp}^{\mathsf{ID-init}-b}_{\mathsf{EDHOC}}(\mathcal{A})$

1. $\mathsf{ID}_{\mathsf{I}_0}, \mathsf{ID}_{\mathsf{I}_1} \leftarrow \mathcal{A}(\texttt{peerid})$
2. $(Y_e, c_2, \mathsf{C_R}) \leftarrow \mathcal{A}(\mathsf{RespRun1}(.))$
3. $b \leftarrow \{0, 1\}$
4. $\mathsf{ID_I} \leftarrow \mathsf{ID}_{\mathsf{I}_b}$
5. $x_s \leftarrow \mathsf{sk}_{\mathsf{ID_I}}$
6. $Y_s \leftarrow \texttt{peerpk}[\mathsf{ID_I}]$
7. $c_3 \leftarrow \mathsf{InitRun2}(\mathsf{ID_I}, x_s, Y_s, (Y_e, c_2, \mathsf{C_R}))$
8. $b' \leftarrow \mathsf{Adv}(c_3)$
9. **return** $b = b'$

We define the advantage $\mathsf{Adv}^{\mathsf{ID-init}-b}_{\mathsf{EDHOC}}$ of the adversary in breaking the responder mutual authentication of EDHOC by:

$$\mathsf{Adv}^{\mathsf{ID-init}-b}_{\mathsf{EDHOC}}(t) = \max_{\mathcal{A}} \left(|\Pr[\mathbf{Exp}^{\mathsf{ID-init}-0}_{\mathsf{EDHOC}}(\mathcal{A}) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ID-init}-1}_{\mathsf{EDHOC}}(\mathcal{A}) = 1]| \right)$$

Theorem 5.5

The above EDHOC protocol protects Initiator's Identity under the Gap Diffie-Hellman problem in the Random Oracle model, the injectivity of $(\mathcal{E}, \mathcal{D})$ and the semantic security of $\Pi' = (\mathcal{E}', \mathcal{D}')$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\mathsf{Adv}^{\mathsf{ID-init}-b}_{\mathsf{EDHOC}}(t; q_{\mathsf{RO}}, n_\sigma, N)$ upper-bounded by

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \mathsf{Adv}^{\mathsf{IND}}_{\Pi'}(t) + \frac{{q_{\mathsf{RO}}}^2 + 2}{2^{\ell_{\mathsf{hash}}+1}}$$

5.5 Improvements

In this section, we present improvements to increase both key privacy and mutual authentication. First, we explain how one can improve mutual authentication with just a few more flows, but without modifying the protocol. Then, we propose a modification of the key schedule that improves both the key privacy security and the mutual authentication security.

5.5.1 Improved mutual authentication with more flows

As seen in Section 5.3, EDHOC protocol offers an explicit authentication security of ℓ_{MAC} bits. The lower setting for ℓ_{MAC} is 64 bits, giving a 64-bit security only. We describe in this section that by considering the fourth optional message sent from the responder to the initiator and a fifth message sent from the initiator to the responder, we can reach a 128-bit security.

With this updated protocol EDHOC, we first reconsider the responder authentication.

Updated Authentication. Consider a simulated **initiator** (resp. **responder**) receiving a forged message c_4 (resp. c_5) from the adversary in the name of a **non-corrupted user**. In such a case, we consider the modifications made in the key privacy proof up to the game G_9. Hence, in both cases, we have replaced the generation of PRK_{4e3m} with a private oracle. We can additionally replace the generation of sk_4 (resp. sk_5) by a private oracle, which makes a difference with probability bounded by $\frac{1}{2^{\ell_{\mathsf{hash}}}}$. Then the advantage of the adversary in passing the new verification step is bounded by $\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\Pi'}(\mathcal{A})$, in conjunction with the previous probability $\frac{1}{2^{\ell_{\mathsf{MAC}}}}$ for the tag:

Theorem 5.6

The above EDHOC protocol satisfies the responder/initiator authentication under the Gap Diffie-Hellman problem in the Random Oracle model, the injective property of $(\mathcal{E}, \mathcal{D})$, and the unforgeability of $\Pi' = (\mathcal{E}', \mathcal{D}')$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have both $\mathsf{Adv}^{\mathsf{auth-init}}_{\mathsf{EDHOC}}(t; q_{\mathsf{RO}}, n_\sigma, N)$ and $\mathsf{Adv}^{\mathsf{auth-resp}}_{\mathsf{EDHOC}}(t; q_{\mathsf{RO}}, n_\sigma, N)$ upper-bounded by

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \frac{1}{2^{\ell_{\mathsf{MAC}}}} \cdot \mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\Pi'}(t) + \frac{{q_{\mathsf{RO}}}^2 + 6}{2^{\ell_{\mathsf{hash}}+1}}$$

Note that one can expect a 128-bit security with $\frac{1}{2^{\ell_{\mathsf{MAC}}}} \cdot \mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\Pi'}$.

5.5.2 Improved EDHOC protocol

We present two modifications, and their motivations.

Non-Empty Nonce for PRK_{2e}. Looking at the games G_6 and $G_{6'}$ of the key-privacy proof, in which we simulate the generation of the key PRK_{2e}, we have the following advantage for the adversary to distinguish between the games G_5 and G_6:

$$|\Pr[\mathrm{Succ}_6] - \Pr[\mathrm{Succ}_5]| \leq \mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathrm{RO}}).$$

One can note that this advantage depends on the number of sessions n_σ. This is induced by the fact that, as the key PRK_{2e} does not depend on the session context, any query Z to the SO_T oracle can solve any of the n_σ GDH instances. Therefore, a first modification we suggest in the key schedule in Figure 5.8 would be to compute the key PRK_{2e} according to TH_2 (or any value depending on the actual session). This implies for the adversary to make a query to the SO_T oracle with a specific TH_2 linked to a single session, leading to the following advantage for the adversary $\mathcal{A}$:

$$|\Pr[\mathrm{Succ}_6] - \Pr[\mathrm{Succ}_5]| \leq \mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, q_{\mathrm{RO}}).$$

and the updated Theorem 5.1:

Theorem 5.1

he above EDHOC protocol satisfies the key privacy property under the Gap Diffie-Hellman problem in the Random Oracle model, and the injective property of $(\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, N the number of users, and ℓ_{hash} the hash digest length, we have $\mathrm{Adv}^{\mathrm{kp-ake}}_{\mathrm{EDHOC}}(t; q_{\mathrm{RO}}, n_\sigma, N)$ upper-bounded by

$$(2N+1) \cdot \mathrm{Adv}^{\mathrm{GDH}}_{\mathbb{G}}(t, q_{\mathrm{RO}}) + \frac{{q_{\mathrm{RO}}}^2 + 4}{2^{\ell_{\mathsf{hash}}+1}}$$

Therefore, this also impacts Theorems 5.2 and 5.3 (Explicit Authentication) and Theorems 5.4 and 5.5 (Identity Protection) as the game G_6 of the key privacy is used in those proofs.

Second message from the Initiator. The encryption key sk_3, used by the initiator to encrypt its second message m_3, is computed by calling HKDF -Expand on PRK_{3e2m}. However, even an adversary that plays in the name of a non-corrupted user, is able to compute PRK_{3e2m}, when knowing the Initiator ephemeral key x_e, as PRK_{3e2m} does not depend on x_s, the long term secret key of the Initiator. In order to break the Initiator authentication, with respect to a Responder, an adversary can play on behalf of any user as an Initiator. It will be able to compute sk_3, but not t_3, for which value it will need some luck, but this is only 64-bit long! Which is not enough for a 128-bit security.

To get around this issue, we suggest modifying the construction of Initiator's second message as follows: Initial message $m_3 = (\mathsf{ID}_\mathsf{I} \| t_3 || \mathsf{EAD}_3)$ is split as $m_3 \leftarrow (\mathsf{ID}_\mathsf{I})$ and $m'_3 \leftarrow (t_3 || \mathsf{EAD}_3)$. Thus, m_3 is encrypted using sk_3 (with a one-time pad encryption scheme $\Pi = (\mathcal{E}, \mathcal{D})$, under sk_3 still depending on PRK_{3e2m}) into c_3. Then m'_3 does not need to be encrypted. We introduce the value ℓ_{sec}, always set as the expected bit-security parameter, independently of the ℓ_{MAC} value. Then, we set the length of t_3 to be ℓ_{sec}, as it already authenticates $\mathsf{CTX}_3 = (\mathsf{ID}_\mathsf{I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3)$. Concretely, the second message sent by the initiator to the responder is: $c_3 \| m'_3$, where $c_3 = \mathcal{E}(\mathsf{sk}_3, m_3)$, $m'_3 = t_3 || \mathsf{EAD}_3$. Once the Responder receives (c_3, m'_3), he first decrypts c_3, retrieves X_s using m_3, computes PRK_{4e3m} and is then able to verify the tag t_3, allowing to check the authenticity of ID_I, as well as all the other values is $\mathsf{CTX}_3 = \mathsf{ID}_\mathsf{I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3$. The extra required length for the tag t_3 is perfectly compensated by the absence of the tag jointly sent when

using Authenticated Encryption, and the plaintext length of m_3 is the same as the encryption of m_3. Therefore, this does not impact the communication cost of the protocol, until $\ell_{\text{sec}} \leq 2 \times \ell_{\text{MAC}}$, but improves to ℓ_{sec}-bit security for Initiator-Authentication.

About the Responder-Authentication, t_2 also provides a 64-bit security level only: by guessing it, any active adversary can make the initiator terminate, and thus breaking the responder-authentication, if one does not wait for the fourth flow c_4, m'_4. However, with this fourth flow, we can show the $2 \times \ell_{\text{MAC}}$-bit security level is achieved.

In Figure 5.8, we show the updated Key Schedule, while in Figure 5.9, we describe the differences between the previous and the new version.

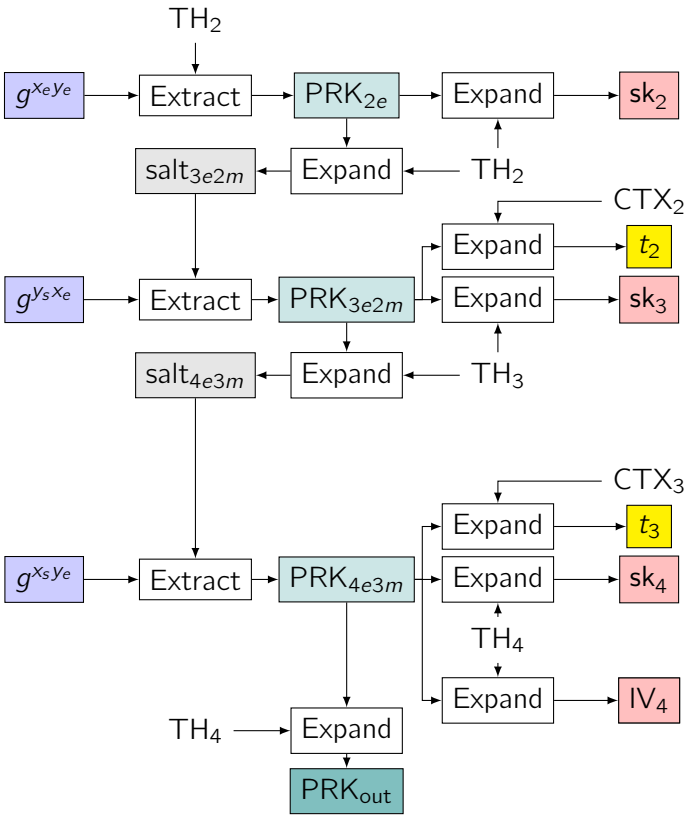


Figure 5.8: Updated Key Derivation after our improvements.

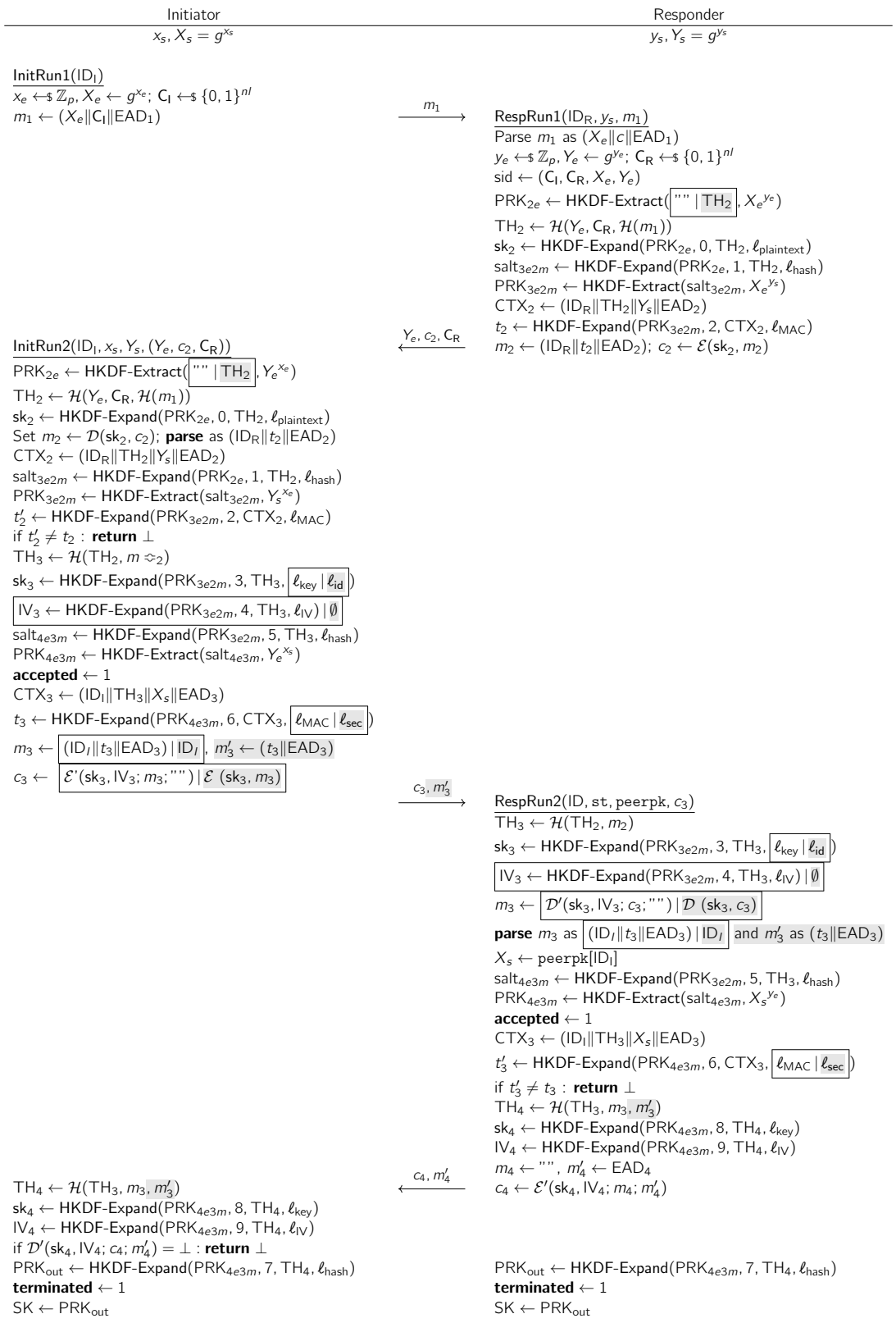


Figure 5.9: Optimized EDHOC with four messages in the STAT/STAT Authentication Method. Our modifications compared to [SMP21] (draft-ietf-lake-edhoc-15) are represented by previous | new and additions by gray highlights

Chapter 6

Conclusion

While initially explicitly used to secure human-to-human communications, cryptography is now implicitly used for device-to-device communications. With the growing concern about privacy jointly to the increasing amount of exchanged messages in our digital live, the spectrum of cryptography get wider. In this thesis, we have provided several contributions through various fields.

First, in chapter 3, we used cryptography as a tool at the service of Machine Learning, a field in continuous expansion. Indeed, whereas previous work consider evaluation on the server side, we showed that one can also securely evaluate a Decision Tree on the client device, opening the doors to use cases such as Continuous Authentication. Our protocol proven to be secure against Honest-but-Curious client only need a single flow to be completed. However, our protocol secure against malicious client requires Garbled Circuits, which use Oblivious Transfers and therefore increases communication cost, both in terms of bandwidth and flows. Even if secure evaluation of Machine Learning models is widely studied, a big remaining challenge is to find solutions to securely train on datasets and securely update encoded models.

Then, in chapter 4, we have explored the field of secure multiparty computation and post-quantum cryptography. We introduced Smooth Projective Hash Function with Gray Zone that are a relaxation of Smooth Projective Hash Functions to cases where correctness nor smoothness can not be claimed for every word. This relaxation is made by considering a new property, named Decomposition Intractability. We showed that if the Decomposition Intractability property was provided, it was enough to build an Oblivious Transfer proven to be secure in the Universal Composability Framework and instantiated such Oblivious Transfer with post-quantum cryptography using the LWE Encryption Scheme.

We may wonder if an instantiation can be done based on other post-quantum family, in particular error correcting codes. More generally, one may wonder if other properties on Smooth Projective Hash Functions with Gray Zone leading could lead to other constructions.

Finally, in chapter 5 we analyzed an authenticated key exchange protocol, EDHOC, specifically developed for lightweight cryptography. This led to improvements in order to reach sufficient security even in aggressive settings. Such protocols will take up more and more space as the Internet of Things grows considerably.

Bibliography

[AAB+22] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. [Cited on page 38]

[ABB+22] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. BIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. [Cited on page 38]

[ABC+22] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions. [Cited on page 38]

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 671–689. Springer, Heidelberg, August 2009. [Cited on page 78]

[AGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 138–154. Springer, Heidelberg, August 2010. [Cited on page 38]

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, Heidelberg, February 2007. [Cited on page 42]

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014. [Cited on pages 5 and 22]

[BBB+21a] Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Yann Connan, and Philippe Gaborit. A gapless code-based hash proof system based on RQC and its applications. Cryptology ePrint Archive, Report 2021/026, 2021. https://eprint.iacr.org/2021/026. [Cited on pages 78 and 80]

[BBB+21b] Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Secure decision forest evaluation. In *The 16th International Conference on Availability, Reliability and Security*, pages 1–12, 2021. [Not cited.]

[BBB+22a] Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Yann Connan, and Philippe Gaborit. A gapless code-based hash proof system based on rqc and its applications. *Designs, Codes and Cryptography*, pages 1–34, 2022. [Cited on pages 4, 5, and 21]

[BBB+22b] Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Post-Quantum Oblivious Transfer from Smooth Projective Hash Functions with grey[1]Zone. hal-03772089, 2022. https://hal.archives-ouvertes.fr/hal-03772089. [Not cited.]

[BBB+22c] Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Secure Decision Forest Evaluation. hal-03321368, 2022. https://hal.archives-ouvertes.fr/hal-03321368. [Not cited.]

[BBB+22d] Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Baptiste Cottier, and David Pointcheval. Post-Quantum and UC-secure Oblivious Transfer from SPHF with Grey Zone. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022. [Not cited.]

[BBC+13a] Fabrice Ben Hamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Efficient UC-secure authenticated key-exchange for algebraic languages. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 272–291. Springer, Heidelberg, February / March 2013. [Cited on page 78]

[BBC+13b] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 449–475. Springer, Heidelberg, August 2013. [Cited on page 78]

[BBDQ18] Fabrice Benhamouda, Olivier Blazy, Léo Ducas, and Willy Quach. Hash proof systems over lattices revisited. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 644–674. Springer, Heidelberg, March 2018. [Cited on pages 80 and 87]

[Ben16] Fabrice Benhamouda. *Diverse modules and zero-knowledge*. PhD thesis, Paris Sciences et Lettres (ComUE), 2016. [Cited on page 41]

[BFK+09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In Michael Backes and Peng Ning, editors, *ESORICS 2009*, volume 5789 of *LNCS*, pages 424–439. Springer, Heidelberg, September 2009. [Cited on page 74]

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, Heidelberg, February 2005. [Cited on page 38]

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012. [Cited on page 45]

[BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 547–557. Springer, Heidelberg, August 1990. [Cited on page 43]

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990. [Cited on pages 45 and 47]

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 498–507. ACM Press, October 2007. [Cited on page 74]

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS 2015*. The Internet Society, February 2015. [Cited on page 74]

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. [Cited on page 31]

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. [Cited on page 96]

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. [Cited on page 78]

[CD22] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). Cryptology ePrint Archive, Report 2022/975, 2022. https://eprint.iacr.org/2022/975. [Cited on page 38]

[CP22a] Baptiste Cottier and David Pointcheval. Security Analysis of Improved EDHOC Protocol. In *The 15th International Symposium on Foundations & Practice of Security*, pages 1–16, 2022. [Not cited.]

[CP22b] Baptiste Cottier and David Pointcheval. Security analysis of the EDHOC protocol. hal-03772082v2, 2022. https://hal.archives-ouvertes.fr/hal-03772082. [Not cited.]

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Heidelberg, August 1998. [Cited on page 85]

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002. [Cited on pages 40, 41, and 78]

[DDH+16] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson C. A. Nascimento, Stacey C. Newman, and Wing-Sea Poon. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. Cryptology ePrint Archive, Report 2016/736, 2016. https://eprint.iacr.org/2016/736. [Cited on page 74]

[DG20] Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029, 2020. https://eprint.iacr.org/2020/1029. [Cited on page 96]

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. [Cited on page 32]

[DPB18] Aurélien Dupin, David Pointcheval, and Christophe Bidan. On the leakage of corrupted garbled circuits. In Joonsang Baek, Willy Susilo, and Jongkil Kim, editors, *ProvSec 2018*, volume 11192 of *LNCS*, pages 3–21. Springer, Heidelberg, October 2018. [Cited on pages 5 and 22]

[DS15] David Derler and Daniel Slamanig. Practical witness encryption for algebraic languages and how to reply an unknown whistleblower. Cryptology ePrint Archive, Report 2015/1073, 2015. https://eprint.iacr.org/2015/1073. [Cited on page 78]

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 205–210. Plenum Press, New York, USA, 1982. [Cited on page 42]

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985. [Cited on page 38]

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. [Cited on pages 37 and 38]

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Heidelberg, May 2003. https://eprint.iacr.org/2003/032.ps.gz. [Cited on pages 41 and 78]

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982. [Cited on page 38]

[GMP+08] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS—Secure sessions with handshake and record layer protocols. Cryptology ePrint Archive, Report 2008/251, 2008. https://eprint.iacr.org/2008/251. [Cited on page 52]

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. [Cited on pages 42 and 45]

[HBD+20] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. [Cited on page 38]

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. [Cited on page 88]

[HRFS99] Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. *Spambase Data Set*, 1999. http://archive.ics.uci.edu/ml/datasets/Spambase/. [Cited on pages 72 and 75]

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003. [Cited on page 44]

[JAC+22] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. [Cited on page 38]

[JR12] Charanjit S. Jutla and Arnab Roy. Relatively-sound NIZKs and password-based key-exchange. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 485–503. Springer, Heidelberg, May 2012. [Cited on page 78]

[Kra03] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, Heidelberg, August 2003. [Cited on page 90]

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008. [Cited on page 47]

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, Heidelberg, December 2009. [Cited on pages 41, 78, and 80]

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Heidelberg, March 2011. [Cited on page 78]

[KW16] Hugo Krawczyk and Hoeteck Wee. The optls protocol and tls 1.3. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P '16)*, pages 81–96. IEEE Computer Society, 2016. https://eprint.iacr.org/2015/978. [Cited on page 90]

[LDK+20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. [Cited on page 38]

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Heidelberg, August 2013. [Cited on pages 5 and 22]

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, Heidelberg, May 2007. [Cited on pages 5, 22, and 51]

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, Heidelberg, August 2014. [Cited on pages 5 and 22]

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473. Springer, Heidelberg, April 2006. [Cited on pages 5 and 22]

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012. [Cited on page 86]

[MZ93] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993. [Cited on page 75]

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457, 2001. [Cited on page 43]

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999. [Cited on page 48]

[NSB21] Karl Norrman, Vaishnavi Sundararajan, and Alessandro Bruni. Formal analysis of edhoc key establishment for constrained iot devices. In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT '21)*, pages 210–221. INSTICC, SciTePress, 2021. https://arxiv.org/abs/2007.11427. [Cited on pages 90 and 93]

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999. [Cited on page 38]

[Per13] Edoardo Persichetti. Secure and anonymous hybrid encryption from coding theory. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 174–187. Springer, Heidelberg, June 2013. [Cited on page 80]

[PFH+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. [Cited on page 38]

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009. [Cited on page 48]

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011. [Cited on page 75]

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008. [Cited on page 82]

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. [Cited on pages 39 and 86]

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978. [Cited on pages 32 and 38]

[SA21] Masoumeh Koochak Shooshtari and Mohammad Reza Aref. Smooth projective hash function from codes and its applications. *IEEE Transactions on Services Computing*, pages 1–1, 2021. [Cited on page 80]

[SAB+20] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. [Cited on page 38]

[Sch16] Jim Schaad. Cbor object signing and encryption (cose), 2016. [Cited on page 92]

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994. [Cited on page 38]

[SMP21] Göran Selander, John Preuß Mattsson, and Francesca Palombini. Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-ietf-lake-edhoc-15, Internet Engineering Task Force, October 2021. Work in Progress. [Cited on pages 90, 92, 94, and 109]

[SRA81] Adi Shamir, Ronald L Rivest, and Leonard M Adleman. Mental poker. In *The mathematical gardner*, pages 37–43. Springer, 1981. [Cited on page 42]

[TKK19] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. *PoPETs*, 2019(1):266–286, January 2019. [Cited on page 74]

[TMZC17] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 494–512. Springer, Heidelberg, September 2017. [Cited on pages 74 and 75]

[WFNL16] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *PoPETs*, 2016(4):335–355, October 2016. [Cited on pages 74 and 75]

[WMK17] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster secure two-party computation in the single-execution setting. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 399–424. Springer, Heidelberg, April / May 2017. [Cited on pages 5 and 22]

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. [Cited on pages 42 and 45]

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015. [Cited on page 49]

[ZY17] Jiang Zhang and Yu Yu. Two-round PAKE from approximate SPH and instantiations from lattices. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 37–67. Springer, Heidelberg, December 2017. [Cited on page 80]

RÉSUMÉ

De nos jours, la cryptographie fait partie intégrante de notre quotidien. Initialement destinée à être utilisée pour les interactions entre humains, elle est aujourd'hui employée pour les interactions avec et entre des machines. A travers cette thèse, nous présenterons nos diverses contributions couvrant une partie du spectre de la cryptographie interactive.

Dans un premier temps, nous la verrons comme un outil au service de l'apprentissage automatique. Nous montrerons qu'elle peut permettre à un client et à un serveur d'interagir afin de faire évaluer les données du client par le modèle du serveur, sans qu'aucun d'eux n'apprenne d'informartion confidentielle au sujet de l'autre.

Ensuite, nous étendrons la notion déjà existante de *Smooth Projective Hash Functions* pour la rendre plus compatible avec la cryptographie post-quantique, et notamment les réseaux euclidiens. Cette extension servira de base à la construction de Transferts Inconscients, largement déployés aujourd'hui dans le cadre de Calculs Multipartites Securisés.

Finalement, nous nous intéresserons à la cryptographie à bas-coût, permettant à des appareils techniquement limités d'interagir entre eux tout en jouissant d'une sécurité suffisante. Plus précisément, nous étudierons la sécurité d'un protocole authentifié d'échange de clés optimisé pour de tels appareils, permettant d'établir une communication sécurisée entre eux.

MOTS CLÉS

Apprentissage automatique – Authentification – Calcul Multipartite Sécurisé – Cryptographie Post-quantique – Cryptographie à bas-coût

ABSTRACT

Nowadays, cryptography is an integral part of our daily lives. Originally intended to be used for human-to-human interactions, cryptography is now also used for with and between machines. In this thesis, we will present our contributions covering part of the spectrum of interactive cryptography.

First, we will use cryptography as a tool for Machine Learning. We will show that a client and a server can securely interact such that the client gets his data evaluated by a model owned by the server, without learning any confidential information about the other party's data.

Next, we will extend the already existing primitive of Smooth Projective Hash Functions to make it more compliant with post-quantum cryptography, and in particular with lattices. This extension will be used as a basis for the construction of Oblivious Transfers, widely used in the Secure Multiparty Context.

Finally, we will focus on lightweight cryptography, allowing technically limited devices to interact with each other while enjoying sufficient security. More precisely, we will perform a security analysis of an authenticated key exchange protocol optimized for such devices, allowing them to securely interact.

KEYWORDS

Machine Learning – Authentication – Secure Multiparty Computation – Post-Quantum Cryptography – Lightweight Cryptography