École Normale Supérieure
Département d'Informatique

Université Paris 7
Denis Diderot

Preuves de connaissance interactives et non-interactives

Thèse

présentée et soutenue publiquement le 27 septembre 2012 par

Olivier Blazy

pour l'obtention du

Doctorat de l'Université Paris Diderot
(spécialité informatique)

Devant le jury composé de :

Directeur de thèse :	David Pointcheval	(CNRS, École Normale Supérieure)
Rapporteurs :	Jean-Sébastien Coron	(Université du Luxembourg)
	Marc Fischlin	(Université de Darmstadt)
	Fabien Laguillaumie	(CNRS, LIP)
Examinateurs :	Michel Abdalla	(CNRS, École Normale Supérieure)
	Antoine Joux	(DGA, Université de Versailles)
	Eike Kiltz	(Université de la Ruhr, Bochum)
	Damien Vergnaud	(CNRS, École Normale Supérieure)

Travaux effectués au Laboratoire d'Informatique de l'École Normale Supérieure

Remerciements

Bien des gens ont contribué de près ou de loin à l'accomplissement de ce mémoire. Je commencerai tout d'abord par remercier mon directeur de thèse, David Pointcheval, qui m'a aidé et soutenu tout au long de ma thèse. Merci pour ta grande disponibilité, ta patience, tes précieux conseils, pour toutes ces discussions qui ont été très enrichissantes. Merci pour tes encouragements, ton optimisme. Ce fut un vrai plaisir d'être ton élève pendant ces années.

Il ne faut pas non plus oublier Jean-Sébastien Coron, Marc Fischlin et à Fabien Laguillaumie. Je leur suis extrêmement reconnaissant d'avoir accepté de sacrifier une partie de leur été pour rapporter ma thèse. Je remercie également Michel Abdalla, Antoine Joux, Eike Kiltz, et Damien Vergnaud de me faire l'honneur d'être dans le jury de ma soutenance.

Sans un cadre adéquat ce mémoire n'aurait pas vu le jour. Merci à toute l'équipe Crypto de l'École Normale Supérieure passée et présente. Merci à tous pour ces excellentes conditions de travail. Je n'oserai pas essayer de nommer tous les gens exceptionnels rencontrés au sein de l'équipe de peur d'involontairement en oublier. Merci à tous les permanents pour leur bonne humeur et leur conseil, merci aussi aux stagiaires, thésards, post-docs pour l'ambiance qu'ils ont su créer. Merci également à toute l'équipe administrative et au service informatique pour leur gentillesse et leur flexibilité ce qui a rendu ces années ici si agréable. Merci aussi aux anciens de m'avoir montré la voie, j'espère avoir l'occasion de pouvoir collaborer avec vous à nouveau.

On dit que les voyages forment la jeunesse, mais aussi les thésards. Je souhaite aussi remercier toutes les personnes que j'ai pu rencontrer au cours de mes stages que ce soit à Louvain-la-Neuve en Belgique, ou à Mountain View en Californie, leur bonne humeur et leurs conseils divers m'ont conforté dans l'idée de faire une thèse en cryptographie, et également tous les soutiens que j'ai pu trouver au cours des diverses conférences auxquelles j'ai pu participer.

Une ANR a aussi axé ma recherche. J'aimerai également exprimer ma gratitude à tout le reste de l'équipe PACE pour m'avoir intégré au projet et pour les divers échanges d'idées lors des réunions.

N'oublions le plus important, je tiens à remercier mes parents qui me soutiennent depuis le début même quand je me mets à leur parler de choses incompréhensibles.

On n'est pas constitué uniquement de 0 ou de 1. Merci à tous les amis qui m'ont accompagné pendant ces trois années, et qui m'ont supporté même quand je leur racontais mon sujet de thèse pour la énième fois, ou qu'au milieu de conversation je déviais subtilement la discussion vers quelque chose de nettement plus cryptique. Sans chercher à être exhaustif, je tiens à évoquer mes colocataires qui ont permis à ces trois années de se passer dans de bonnes conditions[1]. Merci à tous mes comparses normaliens pour les bons moments passés et toutes ces années de souvenirs. Merci aux anciens des Lazos pour leur soutien depuis notre arrivée sur Paris.

Une personne mérite une place toute particulière dans ces remerciements, sans son aide cette thèse serait dépourvue d'un certain je ne sais quoi : _____.

Rassurez-vous ces remerciements arrivent à leur fin. Il ne faut juste pas oublier celles et ceux qui ont relu cette thèse de gré ou de force ;)

Sigles, acronymes parsèment ce manuscrit, cependant certains n'ont pas trouvés places dans cette thèse, et je ne peux m'empêcher de les laisser à la sagacité du lecteur. Une journée type peut se résumer, aux ordre de multiplicités près, par MD, RER, ENS, DI, TS, SC. Il y a aussi des choses comme TBBT, DDR, HIMYM, GoT, COF, CDG, TSW. Sans oublier Team B2, TrTeam et les trop rares membres de Squirrel Corp.

[1] Merci surtout pour les discussions sur les articles à 3h du matin autour d'un verre de *soupline*

Contents

Je sers la science et c'est ma joie.

Prolégomènes

Pendant très longtemps la cryptologie, et plus particulièrement la cryptographie se restreignait au chiffrement, c'est à dire s'intéressait à la transmission d'informations entre deux utilisateurs par l'intermédiaire d'un canal non-sécurisé. Les toutes premières méthodes étaient relativement basiques et gravitaient autour de *substitutions* (ou *interversions*) et de *transpositions*. On a retrouvé de vieux hiéroglyphes, également des scytales utilisées par les Spartiates en 900 avant notre ère, et Suetonius rapporte que Jules César chiffrait ses messages, avec le célèbre chiffrement de César où toutes les lettres sont décalées de trois dans l'alphabet.

Au cours de l'histoire, il y a eu un duel entre les cryptographes et les cryptanalystes, chacun tentant de rendre le travail des autres caduque, en essayant de casser les chiffrements, ou au contraire d'en faire un plus perfectionné résistant aux attaques précédentes. Cette rivalité a transformé les buts de la cryptographie. Là où le but initial était de conserver la confidentialité d'un message au cours d'un échange, l'authenticité des données est devenu un problème d'une importance grandissante, puisqu'en plus de protéger le message, il fallait pouvoir garantir que son contenu n'avait pas été altéré, ou même qu'il venait bien de la bonne personne. Un des enjeux majeurs de la cryptographie moderne est de réaliser ces deux objectifs simultanément. Les utilisateurs désirent pouvoir accéder à leurs données à la volée en toute sécurité, sans pour autant révéler trop d'informations au serveur avec lequel ils interagissent.

Ces nouveaux enjeux ont grandement changé les attentes reposant sur la cryptographie, ce qui l'a conduite à passer d'une construction *ad hoc* où les gens essayaient de résoudre un problème dans sa globalité, en une science beaucoup plus *modulaire* où on considère des petites briques élémentaires que l'on prouve, avant de les combiner dans un schéma beaucoup plus complexe.

Il y a une certaine dualité entre ces deux approches, alors que la première donne souvent des schémas très efficaces avec une preuve de sécurité souvent très complexe, la seconde s'avère souvent moins optimale mais permet une analyse très fine de la sécurité du schéma résultat.

Au cours de cette thèse on va essayer de réconcilier ces deux approches en proposant des constructions modulaires qui s'avèrent particulièrement efficaces sans pour autant compromettre la protection des données des utilisateurs.

Ce travail peut être divisé en deux grandes parties. La première montre comment combiner les briques de bases usuelles que sont le chiffrement, la signature et les preuves de connaissance non-interactives à divulgation nulle de connaissance (NIZK), et même ensuite comment les faire commuter pour implémenter des primitives classiques comme les signatures de groupe, ou les signatures en blanc. Dans la seconde, on cherche à développer les capacités des preuves implicites de connaissance, grâce aux *Smooth Projective Hash Functions* et une fois cela fait, on utilise cette nouvelle brique pour construire plus efficacement des protocoles d'échanges de clé existants et définir une plus grande classe que l'on appellera LAKE (englobant les échanges de clés authentifiés par mot de passe, par accréditation ou encore les *poignées de mains* secrètes).

Grâce aux signatures digitales, aux chiffrements et aux preuves non-interactives à divulgation nulle de connaissance, il est aisé de construire plusieurs primitives cryptographiques permettant l'identification, l'authentification tout en préservant l'anonymat. On se concentrera en premier sur les signatures de groupe [Cv91]. Elles ont été introduites par Chaum et Van Heyst pour permettre aux membres d'un groupe, géré par une autorité indépendante, de signer anonymement au nom du groupe. Pour éviter des dérives, le modèle prévoit que cet anonymat puisse être levé par une autorité supplémentaire (possiblement différente de la précédente). Par la suite le modèle a été renforcé selon diverses approches, pour permettre de déléguer les capacités de levées d'anonymat de certains utilisateurs que ce soit en autorisant les utilisateurs à lever leur anonymat, à des sous-autorités de tracer un utilisateur spécifique, ou encore que n'importe qui puisse détecter si quelqu'un signe deux fois au cours d'une même période temps. Nous nous concentrerons aussi sur les signatures en blanc [Cha83] introduites par Chaum. Ici l'identité de l'utilisateur est connue, mais par contre le message signé demeure masqué. Elles ont été présentées dans

le contexte de la monnaie électronique et sont fréquemment utilisées dans le vote électronique où la valeur d'un bulletin doit demeurer secrète mais où il faut pouvoir vérifier qu'il provient d'une personne légitime.

La plupart des protocoles à deux participants peuvent être décrits en disant que soit A veut donner un message à B seulement si B remplit certaines conditions, soit A et B veulent établir un secret partagé si et seulement si chacun possède bien les informations que l'autre attend de lui. Beaucoup de protocoles rentrent dans ce cadre, certaines signatures en blanc permettent à A d'obtenir une signature de B sur un message, seulement s'il connait le message à signer avant de parler à B, dans les systèmes d'accréditations (anonymes) B peut accéder à une information seulement s'il a les droits pour, mais cependant il ne veut pas donner toutes ses accréditations à A, dans les protocoles de mises en accord de clé basés sur les mots de passe (PAKE) deux utilisateurs établissent une clé de session uniquement s'ils possèdent le même mot de passe, et dans les *poignées de mains* secrètes les deux utilisateurs ne tombent d'accord que s'ils sont dans la même organisation.

Une Brève Histoire de la Cryptographie

Antiquité

La plus vieille trace de cryptographie semble dater de l'Ancienne Égypte. Il y a près de 4000 ans, on a gravé sur la tombe de Khnumothp à Menet Khufu des instructions dans des hiéroglyphes inhabituels.

Elle a eu ensuite un usage plus militaire.

Autour de 900 avant Jésus Christ, des chiffrements par *transposition* étaient couramment utilisés, comme dans le cas des scytales spartiates. On enroulait une bandelette de papier autour d'un bâton, écrivait un message dessus, puis déroulait la bandelette. Pour retrouver le message initial, il fallait enrouler la bandelette autour d'un bâton de même rayon.

On utilisait également des chiffrements par *substitution*, le plus connu étant le chiffrement de César (cf [Sue21][2]) où chaque lettre était remplacée par celle 3 places plus loin dans l'alphabet. (crypto serait devenu fubswr). Il y eut aussi des usages plus religieux comme dans l'Ancien Testament (Atbash, Albam et Albah) qui opéraient des substitutions en prenant les lettres miroirs dans l'alphabet. En transposant leur principe dans un alphabet latin : le premier transformait a en z, b en y, ...; le second s'arrêtait au milieu de l'alphabet (a devenait m, et n devenait z), et le dernier était un peu plus compliqué mais fonctionnait sur la même idée que le chiffrement de César, mais en scindant l'alphabet, tous les trois étaient des convolutions; et à la même époque on pouvait trouver en Inde un des trois arts oratoires : le *mlecchita-vikalpa* qui était un chiffrement de César mais avec une transposition de 1 uniquement (Crypto serait devenu Dszqup).

Moyen-Age

Il y eut peu d'améliorations à cette époque. Al-Khindi cependant proposa la première vraie cryptanalyse, en utilisant une technique basée sur les analyses de fréquences des lettres en 800. Ce peu d'avancées s'explique aussi par le climat de l'époque où écrire dans une langue incompréhensible pouvait s'avérer plus dangereux que de dévoiler certains secrets. Seuls quelques religieux s'y sont essayés sans grands succès.

Tous les chiffrés connus demeurèrent vulnérables, jusqu'à l'introduction des chiffrés poly-alphabétiques, *précisés* par Leon Battista Alberti vers 1467; Johannes Trithemius proposa la tabula recta dans son ouvrage *Poligraphia*, grâce à laquelle Blaise de Vigenère définit le chiffré de Vigenère qui résista aux attaques pendant longtemps. En 1854 cependant, Charles Babbage parvint à le casser en devinant qu'il suffisait d'appliquer la méthode d'analyse des fréquences sur des blocs de taille multiple de la clé. En 1893, le Commandant Étienne Bazeries cassa également le *Grand Chiffre*, inventé par Rossignol fleuron de la cryptographie sous Louis XIV.

[2] "Exstant et ad Ciceronem, item ad familiares domesticis de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset; quae si qui investigare et persequi velit, quartam elementorum litteram, id est D pro A et perinde reliquas commutet."

Ère Moderne

C'est à cette époque que se précisa la route à prendre pour la cryptographie. En 1883, Auguste Kerckhoffs présenta une approche moderne de la cryptographie dans [Ker83] en disant :

1. Le système doit être matériellement, sinon mathématiquement, indéchiffrable.
2. Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber dans les mains de l'ennemi
3. La clé doit pouvoir en être communiquée et retenue sans le recours de notes écrites, et être changée et modifiée au gré des correspondants.
4. Il faut qu'il soit applicable à la correspondance télégraphique.
5. Il faut qu'il soit portatif, et que son maniement ou son fonctionnement n'exige pas le concours de plusieurs personnes.
6. Enfin il est nécessaire, vu les circonstances qui en commandent l'application, que le système soit d'un usage facile, ne demandant ni tension d'esprit, ni la connaissance d'une longue série de règles à observer.

Comme souligner dans son traité, les trois dernières règles semblent assez naturelles, et aujourd'hui encore on vise des systèmes simples d'utilisation et peu coûteux en terme de communication. Cependant, ce qui marqua l'évolution moderne de la cryptographie fut sa seconde règle. C'était la fin de la *Sécurité par l'Obscurité*, un système devait être sûr même si l'ennemi apprenait son principe de fonctionnement. Shannon l'a résumé plus tard en disant simplement : "L'ennemi connait le système". Ce n'est pas encore forcément le cas dans toutes les applications actuelles (certains chiffrement sur les DVD par exemple), mais les résultats modernes suivent plutôt cette idée. Et cela coïncide avec la transition récente de la cryptographie qui est passée d'un usage purement militaire / diplomatique à un usage beaucoup plus courant accessible à tous.

Deux autres étapes récentes dans la cryptographie tendent à la démocratiser un peu plus : la cryptographie à clés publiques de Diffie et Hellman [DH76] où aucun secret initial ne doit être partagé par les deux interlocuteurs; puis les preuves de connaissance à divulgation nulle de connaissance [GMR89] de Shafi Goldwasser, Silvio Micali, et Charles Rackoff qui permettent de prouver la connaissance d'un secret sans pour autant le révéler.

Dans le chapitre 2, page 15 nous préciserons ces notions, et nous nous en servirons pour construire de nouveaux protocoles.

Signatures Digitales et Preuves NIZK

Motivation

Signatures de Groupe

Le modèle BSZ est un excellent exemple appelant à une réalisation par le biais d'une construction modulaire. Ce modèle introduit par Bellare, Shi, et Zhang en 2005 [BSZ05] précise les signatures de groupe dynamiques. Dans ce modèle, on retrouve 3 types de participants, le certificateur, l'ouvreur et les membres du groupe. Pour devenir un membre du groupe, un utilisateur doit le *rejoindre* en interagissant avec le certificateur (ou du moins avec quelqu'un possédant une clé de certification), à la fin de l'interaction l'utilisateur obtient une clé de signature privée. Avec cette clé de signature, il peut dorénavant signer au nom du groupe et n'importe qui peut vérifier la validité de sa signature grâce à la clé publique du groupe. Si jamais la signature est litigieuse, on peut faire appel à l'ouvreur pour étudier la signature : en utilisant la clé d'ouverture et la signature incriminée, l'algorithme d'ouverture retourne l'identité du signataire et une preuve d'ouverture correcte.

Il y a deux pendants à la sécurité. En premier lieu, on désire l'*anonymat*, c'est-à-dire que sans connaissance extérieure il doit être difficile de déterminer qui est à l'origine d'une signature. On s'intéresse également à la *résistance aux forges*, c'est-à-dire d'une part la *traçabilité*, toute signature valide doit être ouvrable en un signataire enregistré et d'autre part la *résistance à la diffamation (non-frameability)* il doit être impossible d'incriminer un utilisateur honnête, que ce soit en signant à sa place ou en biaisant l'ouverture (y compris en cas de coalition des autorités d'enregistrement et d'ouverture).

Pour montrer qu'un tel modèle est réalisable, Bellare *et al.* ont proposé la construction suivante. Étant donné un schéma de signature, un schéma de chiffrement et une preuve de connaissance non-interactive à divulgation nulle de connaissance (toutes ces notions sont précisées par la suite dans la Section 2, page 15), on initialise le système en générant une paire de clés de signature et de vérification et une paire de clés de chiffrement et de déchiffrement. Pour rejoindre le groupe, un utilisateur génère un trousseau de clés de signature, donne la clé de vérification de celui-ci au certificateur, qui va signer sous sa clé à lui la clé de vérification de l'utilisateur, en d'autres termes il va lui donner un certificat d'appartenance au groupe. Un membre peut alors produire une signature du groupe en signant simplement le message avec sa clé personnelle de signature, puis en chiffrant son certificat, sa clé de vérification et sa signature et en envoyant ces données simultanément avec en plus une preuve de connaissance non-interactive à divulgation nulle de connaissance montrant que son certificat et sa signature sont bien valides. L'ouverture se fait en déchiffrant les chiffrés, la clé de vérification donnant alors l'identité du signataire et le certificat de l'autorité servant de preuve.

L'anonymat est assez automatique puisque la signature de groupe est composée uniquement de chiffrés ou de preuve de connaissance non-interactive à divulgation nulle de connaissance, devant tous deux ne pas révéler d'information. La résistance aux forges, vient de la résistance aux forges du schéma de signature sous-jacent, car si ni la signature du certificat ni la signature d'un utilisateur ne peuvent être forgées alors cela garantit que toute signature peut être tracée et qu'il n'est pas possible de piéger quelqu'un impliquant ainsi d'une part la *traçabilité* et d'autre la *résistance à la diffamation*.

Une telle approche est la clé de voûte de nos constructions, où nous allons également combiner des schémas de signature, de chiffrement et des preuves de connaissance non-interactive à divulgation nulle de connaissance.

La méthodologie Groth-Sahai

Pendant très longtemps, le seul moyen de prouver sures de telles réalisations a nécessité de se reposer sur l'heuristique de l'oracle aléatoire [BR93] pour les preuves non-interactives à divulgation nulle de connaissance (en anglais *Non-Interactive Zero-Knowledge Proofs of Knowledge* ou NIZK [FS87]). Dans cette heuristique, on suppose qu'il existe une fonction véritablement aléatoire à laquelle l'attaquant, tout comme les utilisateurs, peut faire appel et qui répond avec une valeur tirée uniformément, par contre elle retourne toujours la même valeur pour une même entrée. Dans l'implémentation concrète d'un tel schéma, l'oracle aléatoire va être substitué par une fonction de hachage.

Les preuves non-interactives à divulgation nulle de connaissance sont des preuves de la véracité d'une affirmation sans donner plus d'informations, très proches des preuves à témoin indistinguable (en anglais *Non-Interactive Witness-Indistinguishable Proofs of Knowledge* ou NIWI [FS90],) qui prouvent la véracité d'une affirmation mais sans révéler le témoin utilisé pour l'affirmer, principalement parce qu'il n'existait pas d'instanciations efficaces dans le modèle standard. Jens Groth et Amit Sahai ont proposé un moyen de construire des preuves NIZK et/ou NIWI pour des énoncés (algébriques) sur des groupes munis d'une application bilinéaire[3]. En particulier, ils proposent des preuves pour vérifier la satisfaisabilité simultanée d'un ensemble d'équations. Ils ont proposé trois instanciations spécifiques basées sur des hypothèses calculatoires différentes : le problème de décision du sous-groupe SD, le problème symétrique Diffie-Hellman SXDH, et le problème décisionnel linéaire DLin.[4] Chacune de ces approches a déjà donné de nombreuses applications dans les dernières années (par exemple des schémas de signatures de groupes [BW06, BW07, Gro07] ou de signatures en blanc [AFG+10, BFPV11]). Leur construction commence par établir des preuves NIWI de satisfaisabilité de certaines équations. Pour cela, ils proposent de s'engager sur un témoin et ensuite de construire une preuve qui dit que ce témoin satisfait une équation. Ils répartissent les équations en trois grandes catégories (les produits de couplages[5] comme $e(A_i, \underline{\mathcal{Y}_j}).e(\underline{\mathcal{X}_i}, \underline{\mathcal{Y}_j}) = g_T$, les multiplications multi-scalaires comme $A_i^{\underline{y_j}}.\underline{\mathcal{X}_i}^{\underline{y_j}} = g$, et les équations quadratiques $a_i.\underline{y_j} + \underline{x_i}.\underline{y_j} = k$), dans le premier type le simulateur peut extraire le témoin de l'engagement et donc la preuve est une preuve de connaissance.

[3] On définit un groupe bilinéaire symétrique par $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ avec un nombre premier p ordre d'un groupe $\mathbb{G}$ généré par g avec e une forme bilinéaire qui va de $\mathbb{G} \times \mathbb{G}$ dans un groupe d'arrivée $\mathbb{G}_T$ d'ordre p généré par $e(g, g)$, et un groupe bilinéaire asymétrique par deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ de même ordre premier p générés par g_1, g_2 et est donc représenté par $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ où $\mathbb{G}_T$ est encore d'ordre p et généré par $e(g_1, g_2)$.

[4] Ils sont définis dans la Section 2.2.2, page 17, sauf SD : Dans un groupe multiplicatif d'ordre composé $n = pq$ où p et q sont premiers, il est dur de déterminer si un élément appartient au sous-groupe d'ordre p

[5] Les couplages sont des formes bilinéaires de $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ où $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ sont des groupes multiplicatifs de même ordre. Dans les applications on leur demande de ne pas être dégénérés, c'est-à-dire que l'image de générateurs de $\mathbb{G}_1$ et $\mathbb{G}_2$ doit être un générateur de $\mathbb{G}_T$

La méthodologie Groth-Sahai et les signatures

Le premier schéma pratique qui a utilisé la méthodologie Groth-Sahai, ou du moins une idée très similaire, était les signatures Boyen-Waters d'Eurocrypt 2006 [BW06] où les preuves sont dérivées des techniques de [GOS06b]. Par la suite Belenkiy *et al.* ont appliqué la transformation Boneh-Boyen, présenté à Eurocrypt 2004 [BB04], à ce schéma dans [BCKL08]. Pour contruire des accréditations anonymes, ils s'engagent sur le message et à la signature, puis grâce aux preuves Groth-Sahai ils prouvent que les contenus sont bien valides. Cependant le message est un scalaire et non un élément de groupe, donc l'extraction ne peut se faire rigoureusement, ce qui introduit une faiblesse dans la notion de sécurité considérée. On parle alors de résistance aux $\mathcal{F}$-forges.

En tenant compte de tout cela, nous pouvons en déduire certains pré-requis sur les briques qui vont nous être nécessaires par la suite, si on désire pouvoir les combiner aisément :

- Le schéma de signature doit avoir une résistance aux forges,
- La signature doit être composée d'éléments d'un groupe bilinéaire,
- Le message doit soit être public, soit un élément de groupe, soit un petit scalaire (pour que le simulateur puisse l'extraire au besoin),
- La vérification de la validité de la signature doit se faire au moyen d'équation de couplages.

Et bien sûr pour respecter nos objectifs, il faut que ces briques soient efficaces.

Signature traçable

En 2004, Kiayias, Tsiounis et Yung présentèrent à Eurocrypt [KTY04] une nouvelle version des signatures de groupes qu'ils nommèrent signatures traçables. Leur but était de revisiter l'ouverture pour qu'elle soit moins pénalisante pour les utilisateurs honnêtes. Il voulait pouvoir autoriser les autorités à déléguer une partie de leur pouvoir de détection à des sous-autorités pour que celles-ci puissent tracer un utilisateur. Au lieu d'ouvrir toutes les signatures, et ainsi mettre grandement en danger les informations privées des signataires honnêtes, ces nouvelles signatures permettent à l'ouvreur de déléguer aux sous-autorités la possibilité de tracer un utilisateur précis sans pour autant pouvoir révoquer l'anonymat des autres. Grâce à cela, les sous-ouvreurs peuvent s'exécuter en parallèle, et les utilisateurs honnêtes n'ont plus à avoir peur pour leur anonymat tant que les autorités ne les tracent pas spécifiquement. Les notions de sécurité usuelles des signatures de groupe s'appliquent encore, mais sont renforcées, ainsi la propriété de correction des signatures implique que l'ouverture d'une signature doit donner un utilisateur qui peut être tracé sur cette signature. L'idée est un peu similaire au chiffrement cherchable de [ABC+05], où une trappe spécifique à un mot-clé, permet de décider si un chiffré contient un mot-clé ou non, sans révéler d'information pour les chiffrés avec d'autres mots clés.

En 2008, Libert et Yung dans [LY09] ont proposé un nouveau schéma de signature traçable prouvée sure dans le modèle standard.

Signature de liste

Les signatures de listes sont une autre variation des signatures de groupe introduites par Canard *et al.* dans [CSST06]. Elles laissent un utilisateur signer anonymement, de manière irrévocable s'il agit honnêtement. Personne ne peut tracer le signataire réel, mais par contre s'il signe deux messages différents au cours du même intervalle de temps, alors les signatures pourront être liées. Bien évidemment on ne doit pas avoir besoin d'ouvrir les signatures pour savoir si un utilisateur a signé plusieurs fois. Les propriétés de sécurités de telles signatures sont proches de celles des signatures de groupe. L'auteur d'une signature doit être anonyme, et il ne doit pas être possible de produire plus d'une signature valide par intervalle de temps sans être détecté.

Intuitivement ces signatures peuvent servir lors d'élections, puisque si l'on défini l'intervalle de temps comme étant le scrutin, un utilisateur va pouvoir voter anonymement sauf s'il essaye de voter plusieurs fois, et n'importe qui peut alors vérifier si quelqu'un a voté plusieurs fois.

Depuis lors, ce fut un problème ouvert de savoir si de telles signatures pouvaient être construites dans le modèle standard.

Signature en Blanc

Les signatures en blanc ont été introduites par Chaum en 1982 dans [Cha83]. Elles autorisent un utilisateur à obtenir une signature sur un message sans que le signataire ne puisse décider de quelle interaction découle une signature. Leur sécurité est définie selon deux axes principaux : d'une part, un utilisateur ne doit pas pouvoir obtenir une signature sur un message non demandé (après q-interactions, un adversaire ne doit pas pouvoir produire de signatures valides pour $q+1$ messages différents), d'autre part comme dit précédemment, le signataire ne doit pas pouvoir découvrir quelle interaction est à l'origine d'une signature.

De telles signatures ont été formalisées par la suite dans [JLO97, PS00], et même instanciées sans oracle aléatoire de nombreuses fois (comme par exemple dans [CKW04, Oka06]). Cependant toutes ces approches n'ont jamais été optimales en terme d'interactions, comme il fallait plus d'un aller/retour pour construire la signature.

Fischlin [Fis06] a donné en 2006 une construction générique pour construire des signatures en blanc qui respectent cette contrainte d'optimalité, et suite à cette article on a vu émerger des constructions [Fuc09, AFG+10]. Pour éviter au signataire de retrouver la session de signature, ils définissent une signature en blanc comme un preuve de connaissance non-interactive d'une signature. Cependant ceci rend les signatures en blanc nettement plus grosses que les signatures du schéma sous-jacent, et ainsi jusqu'à maintenant, cette contrainte d'optimalité du nombre d'interactions a été un problème ouvert, si l'on attend de l'utilisateur qu'il puisse exhiber une signature classique à la fin de l'échange.

Dans le schéma de Fischlin, une signature en blanc est une preuve de connaissance d'une signature d'un chiffré doublée d'une preuve montrant que le chiffré correspond bien à un message donné. Dans le schéma de [Fuc09], l'utilisateur obtient une vraie signature sur un message dont il prouve la connaissance.

On va une étape plus loin : une fois encore l'utilisateur obtient une signature sur un message, mais au lieu d'avoir à faire une preuve de connaissance de cette signature, il lui suffit de changer ses aléas avant de l'exhiber. En procédant ainsi une signature en blanc a exactement le même format qu'une signature du message sous-jacent, et donc en plus de vérifier la contrainte d'optimalité du nombre d'interactions, elle est légère.

Instanciations et Applications

Dans la partie I, page 54, nous donnons des instanciations concrètes basées sur des briques élémentaires, et montrons leur sécurité respective dans le modèle standard.

Autour des Signatures de Groupe

D'abord dans le chapitre 3, page 55, nous combinons les blocs classiques pour résoudre des problèmes ouverts dans le modèle standard. Nous débutons dans la Section 3.2, page 57 par une amélioration du schéma de signatures traçables de Libert et Yung, en ajoutant une fonctionnalité qui permet à l'utilisateur de dire qu'une signature est à lui ou au contraire de le réfuter. Fondamentalement, en plus des propriétés classiques d'une telle signature, on ajoute : étant donné une signature, un utilisateur est en mesure d'affirmer/réfuter, et de justifier cette affirmation, que cette signature est à lui comme illustré sur la figure 1, page xi. Pour ce faire, nous reformalisons les notions de sécurité usuelles, et précisons également ce qu'on attend de notre nouvelle fonctionnalité, c'est-à-dire un utilisateur doit être en mesure de confirmer / infirmer qu'il est l'auteur d'une signature.

Nous construisons ensuite un schéma pour répondre à ces objectifs à partir de blocs usuels, à savoir le certificat de [DP06][6], la fonction de Dodis-Yampolskiy [DY05] et la signature de Waters que nous prouvons dans un groupe bilinéaire asymétrique. Nous avons ensuite combiné ces éléments à l'aide de la méthodologie Groth-Sahai pour pouvoir garantir l'anonymat.

Le certificat Delerablée-Pointcheval [DP06] permettra la délégation des capacités d'ouverture, et ce grâce à une trappe qui ne suffit pas pour signer mais permet de tracer une signature à un utilisateur présumé. Les utilisateurs seront également en mesure de confirmer (*step-in*) ou d'infirmer (*step-out*) qu'ils sont le signataire réel, en utilisant leur clé de signature, de manière convaincante, ce qui est une autre nouvelle propriété attrayante. Pour atteindre cet objectif, nous définissons la notion d'identifiant unique, lié à chaque signature, et spécifique à l'utilisateur.

[6] Un tel certificat est de la forme $(x_i, y_i, A_i = (kg^{y_i})^{1/\gamma+x_i})$ où γ est la clé secrète de l'autorité de certification, $\Omega = g^{\gamma}$ la clé publique du groupe, et y_i la clé secrète de l'utilisateur

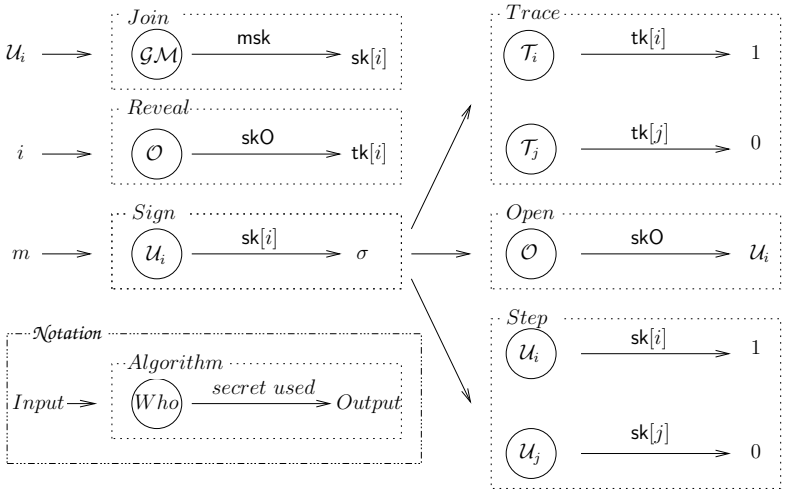


Figure 1: Notre nouveau modèle de Signature Traçable

Nous avons besoin de définir une nouvelle hypothèse, une variante hybride du SDH que nous nommons le $q-$ HHSDH). Bien que cette hypothèse soit nouvelle, nous pouvons montrer qu'elle est encore relativement raisonnable, comme elle est équivalente au $q-$ SDH sous KEA[7]

Pour ce faire, nous introduisons un identifiant unique ID basée sur la fonction de Dodis-Yampolskiy, il est fonction de la clé secrète des utilisateurs. On montre ensuite à l'aide de la méthodologie Groth-Sahai que cette clé secrète a été enregistrée auprès de l'autorité de certification. Et on lie le tout à une signature Waters[8],du message pour résister aux forges. Grâce à l'identifiant, une autorité munie de la correcte trappe est capable de tracer un utilisateur. Par contre seul le possesseur de la clé d'extraction des mises en gage est capable de lever l'anonymat de n'importe quelle signature.

L'instanciation finale est deux fois plus efficace que celle du schéma précédent tout en fournissant une fonctionnalité supplémentaire.

Nous avons ensuite continué à améliorer ce schéma dans la Section 3.3, page 66 pour en faire une signature de liste, pour cela nous clarifions les exigences de sécurité d'origine en définissant des jeux de sécurité appropriés. Munis de cette technique de l'identificateur unique, nous pouvons donner une réponse positive au problème ouvert des signatures de liste dans le modèle standard : si nous construisons l'identifiant unique comme spécifique à l'utilisateur et à l'intervalle de temps, d'une manière déterministe, alors deux signatures du même utilisateur au sein de la même période auront le même identifiant.

Signature sur des Chiffrés Randomisés

Par la suite nous abandonnons l'anonymat du signataire, pour vouloir à la place masquer le message. Pour cela, nous étudions dans le chapitre 4, page 70 une nouvelle primitive que nous appelons Signature sur des chiffrés randomisés : étant donné une signature sur un chiffré, quiconque doit pouvoir randomiser le chiffré et adapter la signature en conséquence sans avoir à connaitre la clé secrète de signature.

Ainsi toute paire chiffré / signature sur ce chiffré peut être randomisée de manière consistante.

Bien entendu cela contredit la notion usuelle de résistance aux forges puisqu'on est en mesure de changer l'objet sur lequel porte une signature, nous devons donc alors définir une nouvelle notion pour ces signatures pour garantir la sécurité de nos applications : la résistance aux forges d'une signature sur un chiffré randomisable implique qu'un adversaire ne doit pouvoir produire une signature sur un chiffré d'un message que s'il en connait déjà une sur le même message. Plus formellement, aucun adversaire ne peut, après maintes requêtes de signatures sur des chiffrés de son choix, produire une signature sur un chiffré dont le clair n'était pas l'un de ceux des chiffrés précédents.

[7]Ces notions sont précisées en Section 3.2.2, page 61. KEA [Dam92] disant qu'étant donné un couple (g,h) pouvoir donner g^a, h^a revient à connaitre a, on en déduit aisément qu'étant donné une liste $(g, g^\gamma, \ldots, g^{\gamma^\ell})$, il est aussi dur sous KEA de donner $(g^x, h^x, g^{1/(\gamma+x)})$ que de donner $(x, g^{1/(\gamma+x)})$.

[8]Étant donné un groupe bilinéaire $(p, \mathbb{G}, e, g)$ et des générateurs indépendants $h, (u_i)$, on définit pour une paire de clés de signatures / de vérifications (h^x, g^x) et un message M (composé de bits m_i) la signature Waters de M $\sigma(M)$ comme la paire $(h^x \mathcal{F}(M)^s, g^s$ où $\mathcal{F}(M) = u_0 \prod u_i^{m_i}$

On étend ensuite notre primitive aux signatures *extractables* sur des chiffrés randomisables : étant donné une clé de déchiffrement dk, une signature $\sigma(\mathcal{C})$ sur un chiffré $\mathcal{C}$ on peut *extraire* une signature $\sigma(M)$ sur un clair M. cela permet à l'utilisateur dans un schéma de signature en blanc de retrouver ($\mathsf{SigExt}_{\mathcal{SC}}$) la signature sur le message une fois que le signataire en a signé le chiffré ($\mathsf{Sign}_{\mathcal{SC}}$).

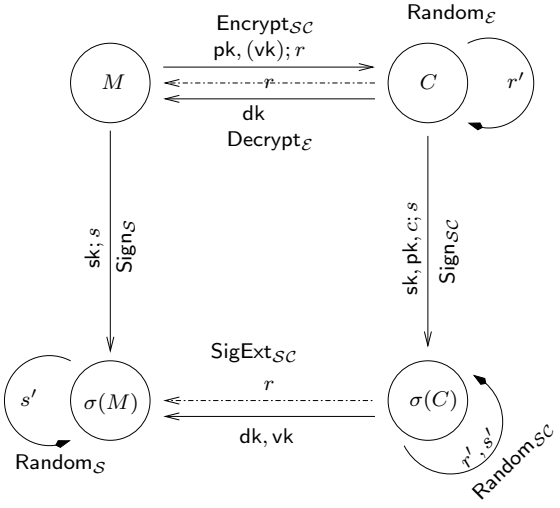


Figure 2: Signatures (fortement) extractables sur des chiffrés randomisables

On dit que la notion d'extraction est forte s'il est possible à l'aide de l'aléa r de l'algorithme de chiffrement $\mathsf{Encrypt}_{\mathcal{SC}}$ d'extraire la signature sur le clair.

Nous donnons plusieurs instanciations de signatures (fortement) extractables sur des chiffrés randomisables, reposant sur des hypothèses classiques. Nos constructions sont toutes basées sur le même genre de briques élémentaires dont elles héritent la sécurité : des preuves NIWI à la Groth-Sahai [GS08] et des signatures Waters dérivées de [Wat05] et utilisées dans [BW06]. Comme la vérification de ces signatures est une équation de couplages, cette brique se combine bien avec la méthodologie Groth-Sahai.

La première instanciation est faite dans un groupe bilinéaire symétrique sur des courbes elliptiques et utilise en plus un chiffrement linéaire [BBS04]. La sécurité sémantique de celle-ci repose sur le problème décisionnel linéaire (DLin), et la résistance aux forges sur le problème calculatoire Diffie-Hellman (CDH).

La méthode naïve qui consiste à s'engager sur le haché Waters du message[9], faire une preuve de connaissance bit par bit du message, puis recevoir une signature à la Waters sur le chiffré [10], et utiliser la clé de déchiffrement pour retrouver une signature en clair est proche de notre méthode. Cependant une étude plus détaillée, montre qu'il est nécessaire d'ajouter une mise en gage sur la clé publique de signature avec les aléas de la mise en gage en exposant pour pouvoir garantir une résistance aux forges.

Pour éviter de se répéter, nous n'allons pas rappeler l'instanciation sur un groupe bilinéaire asymétrique, qui elle repose sur le chiffrement ElGamal et la variante SXDH des preuves Groth-Sahai (mais elle est cependant disponible dans la version complète de [BFPV11]). Même si l'efficacité est meilleure, ce schéma demande de transférer la signature Waters dans un groupe asymétrique, ce qui fait reposer sa sécurité non plus sous CDH mais sous une variante que nous nommons CDH^+ qui requiert de donner des éléments en plus à l'adversaire.

Les tableaux 1, page xiii, donnent la taille d'un couple chiffré-signature selon que l'on soit dans une instanciation symétrique, reposant sur un chiffrement linéaire et une signature Waters classique, ou asymétrique, reposant sur un chiffrement ElGamal et une signature Waters asymétrique, en outre le paramètre k dénote la taille des messages.

Grâce à notre nouvelle primitive, nous obtenons immédiatement une signature en blanc raisonnablement efficace avec un nombre optimal d'interactions sures sous des hypothèses standards. Par ailleurs, en exploitant le fait que notre chiffrement est homomorphe, nous construisons un schéma de vote électronique non-interactif et sans possibilité de vente de vote comme suit : l'utilisateur chiffre son vote, prouve sa validité, et envoie le chiffré, une signature de celui-ci, et la preuve au centre de vote. Ce dernier peut désormais randomiser le texte chiffré, et adapter à la fois la preuve et la signature de l'utilisateur, et

[9] $\mathcal{F}(m) = u_0 \prod u_i^{m_i}$ pour des générateurs u_i indépendants

[10] Le signataire envoie $h^x \mathfrak{C}^s, g^s$, où $\mathfrak{C}$ est un terme du chiffré de $\mathcal{F}(m)$. Dans notre instanciation basée sur DLin, pour un chiffré $u^a, v^b, g^{a+b}\mathcal{F}(m)$ le signataire envoie $h^x(g^{a+b}\mathcal{F}(m))^s, g^s$

Symétrique	$\mathbb{G}$
Waters + Linéaire	$9\,k + 44$

Asymétrique	$\mathbb{G}_1$	$\mathbb{G}_2$
Waters + ElGamal	$6\,k + 9$	$6\,k + 7$

Table 1: Éléments de groupes nécessaires pour un couple chiffré-signature selon l'instanciation choisie

ensuite les publier. Après l'annonce des résultats, l'utilisateur peut vérifier sa signature, qui le convainc que le texte chiffré randomisé contient encore son vote initial en raison de notre notion de résistance aux forges, mais il ne peut pas prouver à qui que ce soit quel était son vote.

Plus tard, nous reprenons le schéma de signature en blanc pour continuer à le modifier. Tout d'abord, dans la Section 4.4.1, page 82 nous considérons une version qui réalise des signatures partiellement en blanc. Ce à quoi nous ajoutons une propriété d'aveuglement parfait, tout en restant sous des hypothèses standards. Pour cela, on utilise l'autre initialisation des mises en gage dans la méthodologie Groth-Sahai. Nous avons également élargi le modèle des signatures partiellement en blanc pour ne plus requérir nécessairement de communication préalable pour se mettre en accord sur la partie publique, pour simplement laisser le signataire la choisir à la volée. Si jamais l'utilisateur ne voulait pas cette modification, il peut simplement rejeter la signature et recommencer à zéro. Nous appelons cette nouvelle primitive des signatures en blanc respectueuses du signataire. Bien sûr, cette notion nouvelle n'interdit pas toute forme d'accord préalable sur la partie publique, elle renforce simplement la notion existante.

Il est maintenant possible de se débarrasser de l'accord préalable sur la partie commune de l'information publique dans le message signé et notre instanciation permet au signataire de le faire d'une manière optimale. Ces deux constructions étant compatibles, nous pouvons présenter une signature partiellement en blanc, optimale qui propose un aveuglement parfait. Notre protocole ne nécessite pas de pré-traitement pour la partie publique du message. L'utilisateur et le signataire peuvent tous deux choisir un morceau de la partie publique, mais au lieu d'avoir un temps de calcul spécifique à l'accord ceci se fait dorénavant lors de l'interaction. Le signataire peut toujours refuser de signer quelque chose où l'information publique de l'utilisateur ne lui convient pas et l'utilisateur peut toujours choisir de ne pas exploiter une signature sans intérêt, c'est pour cela qu'il vaut mieux éviter de gaspiller du temps de communication et rester dans un protocole de deux flux.

En mettant de coté la propriété d'aveuglement parfait, nous profitons de cette propriété asynchrone (l'utilisateur et le signataire choisissent indépendamment leurs parties publiques) dans la Section 4.4.2, page 87 et nous considérons le nouveau contexte où le message devant être signé provient de plusieurs sources indépendantes qui ne peuvent pas communiquer entre elles. Nous présentons d'abord un moyen d'obtenir une signature sur la concaténation des messages d'entrée. Nous présentons également une instanciation plus efficace qui donne une signature sur la somme des messages d'entrée. Une telle somme peut être utile si l'on travaille sur des bulletins de vote, des informations de capteurs, etc. Comme nous utilisons la signature Waters, nous nous sommes intéressés à sa programmabilité sur un alphabet non-binaire d'une manière similaire comme il a été fait dans [HK08] pour l'alphabet binaire. Nous démontrons un résultat négatif sur la $(2, 1)$-programmabilité, mais on a une réponse positive sur la $(1, \mathit{poly})$-programmabilité.[11]

Preuves Implicites de Connaissance et Smooth Projective Hash Functions

Motivation

Nos résultats précédents présentent certes une amélioration de l'efficacité des protocoles actuels mais nous conduisent cependant à nous poser une nouvelle question. Pourquoi utilise-t-on encore des preuves non-interactives de connaissance dans des protocoles interactifs ? N'y aurait-il pas une manière d'utiliser des preuves interactives sans pour autant augmenter le nombre d'échanges dans de tels protocoles ? Dans la plupart des preuves de connaissance, la dernière étape consiste en un envoi d'un message de la part du prouveur au vérifieur pour conclure la preuve. Cependant, même si cette approche semble intuitive cela induit une communication supplémentaire et demande donc au moins 3 échanges (alors que l'on a vu que 2 suffisaient amplement pour des signatures en blanc). Nous allons donc nous intéresser à des preuves implicites de connaissance, où le vérifieur n'apprend pas nécessairement la véracité de la déclaration du prouveur, mais par contre il est assuré que seul un prouveur honnête peut exploiter l'information demandée, ce qui va nous permettre de ne pas augmenter le nombre d'échanges dans nos constructions.

[11] Intuitivement une fonction est (a, b)-programmable si la probabilité que sur $a + b$ variables, a images vérifient une certaine contrainte et b ne la vérifient pas, est non négligeable. Cette notion est précisée en Section 2.6.4, page 42.

Smooth Projective Hash Function

Les *Smooth projective hash functions* (SPHF) ont été présentées par Cramer et Shoup [CS02] pour construire des schémas de chiffrement. Une famille de hachés projectifs est une famille de fonctions de hachage qui peuvent être évaluées de deux manières : soit en utilisant la clé de hachage (secrète) qui permet de calculer la fonction en tout point du domaine, soit en utilisant la clé (publique) *projetée* qui ne permet de calculer la fonction que sur un sous-ensemble du domaine. Une telle famille est dite *lisse (smooth)* si l'image de tout point à l'extérieur du sous-ensemble est indépendante de la clé de projection. De plus s'il est difficile de distinguer si un élément est dans le sous-ensemble, on peut alors voir cette primitive comme un type particulier de preuve de connaissance à divulgation nulle de connaissance d'appartenance au dit sous-ensemble. Cette notion de SPHF a déjà trouvé de nombreuses applications en cryptographie (Par exemple [GL03, Kal05, ACP09]).

Nous préservons aussi d'autres applications basées sur des primitives respectant la vie privée nativement interactives. Et nous étendons l'ensemble des langages compatibles avec ces fonctions, et montrons que par exemple ils peuvent englober des cas non couverts par la méthodologie Groth-Sahai, comme des équations sans couplages, ou encore impliquant des éléments du groupe d'arrivée $\mathbb{G}_T$, et même des conjonctions et disjonctions de langages relativement différents.

Signatures en Blanc

Comme expliqué dans la section précédente, nous pouvons utiliser des Signatures sur des Chiffrés Randomisables pour obtenir des signatures en blanc à interactions minimales. Cependant la construction précédente reposait fortement sur les preuves Groth-Sahai et même si l'amélioration était conséquente par rapport aux solutions antérieures qui ne produisaient pas des signatures classiques, on peut encore vouloir améliorer l'efficacité de notre schéma. Ainsi l'un de nos premiers résultats avec notre nouvelle méthodologie de preuves implicites de connaissance est d'augmenter l'efficacité de la construction précédente sans en diminuer la sécurité.

Il y a de plus en plus de travaux sur les protocoles gravitant autour de la *négociation automatisée de confiance*, ce qui inclut les *Oblivious Signature-Based Envelope* [LDB03] (une famille de protocoles autorisant un utilisateur à envoyer un message sans apprendre l'affiliation de ce dernier, mais en étant sûr qu'il ne pourra le lire que s'il appartient au bon organisme), les *Poignées de main secrètes* [BDS+03] (où deux utilisateurs apprennent s'ils appartiennent à la même organisation, mais sans apprendre l'organisation de l'autre si ce n'est pas le cas), et les schémas de mise en accord de clés gravitant autour des mots de passe [BM93, BPR00], ou d'accréditations secrètes [BHS04]. Tous ces schémas sont étroitement liés (en combinant astucieusement deux d'entre eux, il est possible d'obtenir n'importe quel autre [CJT04]).

Oblivious Signature-Based Envelope

Les *Oblivious Signature-Based Envelopes* (OSBE) ont été introduites dans [LDB03]. Elles peuvent être vues comme une solution élégante pour palier le déséquilibre inhérent de certains protocoles d'identification. Alice est membre d'une organisation et possède un certificat produit par une autorité attestant ce fait. Bob veut envoyer un message privé P aux membres de cette organisation. Cependant à cause de la nature de l'organisation, Alice ne désire donner à Bob (ou à qui que ce soit d'autre) ni son certificat, ni une preuve qu'elle appartient à la dite organisation. Les OSBE permettent à Bob d'envoyer une version obfusquée du message P à Alice, de manière telle qu'Alice puissent y accéder si et seulement si elle est dans la bonne organisation. A l'issue de l'interaction, Bob ne peut pas savoir si Alice appartient réellement à l'organisation.

La sécurité d'un OSBE est usuellement définie en 2 points : l'utilisateur (Bob) ne doit pas pouvoir savoir si Alice utilise une signature valide d'une part, et d'autre part Alice ne doit pouvoir accéder au message de Bob que si elle a utilisé une signature valide. Nous améliorons le premier point, en disant que qui que ce soit (et plus particulièrement l'autorité qui a donné la signature à Alice) ne doit pas pouvoir décider si Alice a utilisé une signature valide, et ajoutons la propriété que l'autorité ne doit pas pouvoir *a posteriori* obtenir le message de Bob même si elle a écouté la communication entre lui et Alice.

Poignées de mains secrètes

Le concept de protocoles de *poignées de mains* secrètes (*Secret Handshakes*) a été présenté en 2003 par Balfanz, Durfee, Shankar, Smetters, Staddon et Wong [BDS+03] (puis développer plus en détails par [JL09, AKB07]). Il permet à deux membres d'un même groupe de s'identifier comme tels mutuellement

en secret, c'est-à-dire qu'un utilisateur n'apprend l'affiliation de l'autre que s'ils appartiennent au même groupe. A la fin du protocole, les participants génèrent une clé de session pour sécuriser de futures communications et un extérieur ne peut savoir si la poignée de main a réussi. Divers raffinements peuvent être joué comme la présence de rôle en plus de l'organisation, ou même encore les organisations peuvent être différentes ...

Échange de Clés Authentifié par mot de passe

Les *Password-Authenticated Key Exchanges* (PAKE) ont été formalisés par Bellovin et Merritt [BM92] et ont donné lieu à de nombreux travaux sous diverses hypothèses (comme [ACP09, CCGS10] et leurs références). Ils permettent à des utilisateurs de générer une clé cryptographiquement forte basée sur un mot de passe simple à mémoriser (avec peu d'entropie) sans avoir besoin d'un système à clé publique. Dans ce contexte, un adversaire contrôlant toutes les communications d'un réseau et apte à corrompre les participants n'importe quand ne doit pas pouvoir faire une attaque par dictionnaire, une fois déconnecté.

Il existe une variante où un utilisateur connaît un mot de passe, et le serveur ne connaît qu'une fonction (à sens unique) de celui-ci. Ainsi si le serveur est compromis, le mot de passe n'est pas révélé immédiatement.

Échange de Clés Authentifié par accréditation

Plus récemment, les *Credential-Authenticated Key Exchanges* (CAKE) ont été introduits par Camenisch, Casati, Groß et Shoup dans [CCGS10]. Dans cette primitive, une clé commune est établie si et seulement une certaine relation est vérifiée par les accréditations que possèdent les deux utilisateurs.

Ils montrent que cette approche doit englober les divers types de protocoles précédents et esquissent des instanciations de divers schémas comme par exemple un protocole de PAKE. Leur méthodologie gravite autour de leur système de preuves à divulgation nulle de connaissance dans le modèle UC sur des problèmes de représentation[12].

Résultats et Instanciations

Notre principale contribution dans la Partie II, page 92 est de définir une méthodologie générale pour faire des preuves implicites de connaissance. Nous nous concentrons sur la présentation de cette approche dans un groupe symétrique donc sous l'hypothèse décisionnelle linéaire DLin. Cependant, on peut montrer que la même approche peut être faite dans un groupe asymétrique sous XDH et non SXDH.

Smooth Projective Hash Functions sur des mises en gage

Pour faire nos preuves implicites de connaissance, nous définissons des Smooth Projective Hash Functions sur des mises en gage. Si l'on désire faire un parallèle avec la méthodologie Groth-Sahai, nous partons du principe que l'on possède un mot M et un témoin m que ce mot appartient à un langage $\mathcal{L}$. Nous nous engageons ensuite sur M et envoyons cette mise en gage. Le *Vérifieur* ensuite calcule une clé de hachage hk, une clé de projection hp sur le langage $\mathcal{L}$ qu'il attend de nous, et sa vue du haché H. Il nous envoie alors hp ainsi que l'information désirée masquée par H. Le prouveur à l'aide de son témoin m et de hp peut alors calculer sa propre vue H' du haché. Si en effet m était bien un témoin que M était dans $\mathcal{L}$ alors $H' = H$ et il peut donc en déduire l'information demandée, sinon il n'apprend rien.

Langages Utilisables

Cette méthodologie nous a poussés à essayer d'accroître le spectre des langages prouvables par des SPHF. Abdalla *et al.* ont montré comment utiliser des conjonctions et disjonctions de langages dans [ACP09], nous allons donc nous concentrer sur des langages de base. Dans le Chapitre 5, page 93, nous procédons par étape. Nous débutons par une mise en gage d'une signature valide et montrons comment procéder, nous montrons également comment gérer une mise en gage d'une mise en gage. Nous voyons alors que ces deux méthodes peuvent se ramener à un langage vérifiant une mise en gage de 1. Nous pouvons déjà itérer à partir de là (et donc gérer des mises en gage de langages), mais surtout cela nous donne une bonne idée sur la manière de développer de nouveaux langages de base. Nous montrons alors que nous pouvons traiter n'importe quel langage composé de mots (découpés en éléments $\mathcal{Y}_i$ à mettre en gage dans

[12] Étant donné g et u_1, u_2, u_3, u_4, il faut donner $\alpha, \beta, \gamma, \delta$ tels que $g = u_1^\alpha u_2^\beta u_3^\gamma u_4^\delta$

Symétrique	$\mathbb{G}$
Groth-Sahai	$9\,k + 44$
avec SPHF	$8\,k + 12$

Asymétrique	$\mathbb{G}_1$	$\mathbb{G}_2$
Groth-Sahai	$6\,k + 9$	$6\,k + 7$
avec SPHF	$5\,k + 6$	1

Table 2: Éléments de groupes nécessaires pour un couple chiffré-signature selon l'instanciation choisie

$\mathbb{G}$, dans $\mathbf{c}_i$, pour $i \in [\![1, m]\!]$ et en éléments $\mathcal{Z}_i$ à mettre en gage dans $\mathbb{G}_T$, dans $\mathbf{C}_i$, pour $i \in [\![m+1, n]\!]$) satisfaisant des équations linéaires étendues de la forme suivantes :

$$\left(\prod_{i\in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i})\right) \cdot \left(\prod_{i\in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}}\right) = \mathcal{B}_k, \text{ pour } k \in [\![1, t]\!].$$

où $\mathcal{A}_{k,i} \in \mathbb{G}$, $\mathcal{B}_k \in \mathbb{G}_T$, et $\mathfrak{z}_{k,i} \in \mathbb{Z}_p$, ainsi que $A_k \subseteq [\![1, m]\!]$ et $B_k \subseteq [\![m+1, n]\!]$ sont publiques.

Intuitivement notre méthode cherche le plus possible à transformer les langages en un langage simple (triplet linéaire). Dans le cas des équations de couplages, le langage est simple, d'une part on demande à ce que chaque chiffré soit valide, donc cela va associer un résidu à chaque chiffré (si on a un triplet (u^r, v^s, h) , il est possible de voir h comme $g^{r+s}\mathcal{Y}_i$), et ensuite un langage supplémentaire qui dit que les résidus sont tels que l'équation est vérifiée.

Nous voyons alors que notre méthodologie peut traiter des mots à la fois dans $\mathbb{G}$ et $\mathbb{G}_T$, ce qui n'était pas possible avec la méthodologie Groth-Sahai. Nous pouvons même montrer que dans certains cas nous arrivons à nous passer de couplages quand tous les mots sont dans $\mathbb{G}$ (comme par exemple pour les quadruplets Diffie-Hellman (g, g^a, g^b, g^{ab})), ce qui est une amélioration significative. Dans le cas de groupe asymétrique, nous pouvons aussi trouver des cas intermédiaires où il faut certes des couplages mais seulement sous XDH et non SXDH (nécessaire pour la méthodologie Groth-Sahai), ce qui permet d'exploiter les courbes elliptiques de type-II et d'être nettement plus efficace.

Dans des instanciations pratiques, nous utilisons souvent des langages simples avec des équations linéaires du type $e(\underline{\mathcal{Y}}, \mathcal{A}) = \mathcal{B}$, notre méthodologie nous demanderait alors 3 éléments de groupe pour la mise en gage et 2 pour la clé de projection (donc la preuve implicite), là où l'approche basée sur Groth-Sahai en requiert 3 pour la mise en gage et également 3 pour la preuve finale, ce qui nous rend sensiblement plus efficace. De plus nous n'avons plus besoin de deux états dans la CRS. De cette façon nous n'avons plus à jongler entre les deux types de clés de mise en gage.

Dans un contexte asymétrique, nous pouvons également faire plus que des équations linéaires (au sens de Groth-Sahai), en gérant simultanément plusieurs équations du type :

$$\left(\prod_{i=1}^{m} e(\mathcal{X}_i, \mathcal{B}_i)\right) \cdot \left(\prod_{j=1}^{n} e(\mathcal{A}_j, \mathcal{Y}_j)\right) \cdot \left(\prod_{k=1}^{o} \mathcal{Z}_k^{\mathfrak{z}_k}\right) = g_T,$$

où $\mathcal{A}_j, \mathcal{B}_i, g_T$ sont des valeurs publiques, dans $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$ respectivement, et où $\mathcal{X}_i, \mathcal{Y}_j, \mathcal{Z}_k$ sont les valeurs privées mises en gage, dans $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$ respectivement.

Avec ce nouvel outil, nous pouvons alors considérer et améliorer plusieurs solutions récentes ce que nous détaillons dans le Chapitre 6, page 102.

Signature en Blanc à interactions minimales

Pour montrer l'efficacité de notre méthode et sa facilité d'utilisation nous révisons notre schéma de signatures en blanc [BFPV11] en supplantant les preuves à la Groth-Sahai par nos nouvelles preuves. Notre approche convient parfaitement et réduit significativement le coût des communications. Nous gagnons même un facteur 3 dans une des instanciations, puisqu'en asymétrique nous pouvons nous passer de toute mise en gage dans $\mathbb{G}_2$, cela nous permet d'ailleurs de reposer sur XDH et non SXDH élargissant ainsi les courbes elliptiques que nous pouvons utiliser. (C'est-à-dire Type-II et Type-III selon la terminologie de [GPS08] au lieu de seulement celles de Type-III comme requis dans [BFPV11].)

Comme précédemment, les tableaux 2, page xvi, donnent la taille d'un couple chiffré-signature selon que l'on soit dans une instanciation symétrique, à base de chiffrement linéaire et d'une signataire Waters classique avec soit des preuves Groth-Sahai soit des SPHF, ou asymétrique, à base de chiffrement ElGamal et d'une signataire Waters asymétrique avec soit des preuves Groth-Sahai soit des SPHF, en outre le paramètre k dénote la taille des messages :

Par rapport au schéma précédent, il y a un gain important dans la partie constante dans l'instanciation sous DLin (dans des groupes symétriques) grâce à nos Smooth Projective Hash Functions, et comme expliqué précédemment le gain est encore plus notable en asymétrique puisque nous retirons quasiment tous les éléments de $\mathbb{G}_2$ (ce qui souligne une forte redondance dans les preuves Groth-Sahai), les éléments de ce groupe étant deux fois plus gros que ceux de $\mathbb{G}_1$ le gain est particulièrement notable.

Oblivious Signature-Based Envelope

Une autre de nos contributions est de clarifier et d'améliorer les attentes de sécurité des schémas d'OSBE dans la Section 6.1, page 102. L'extension de sécurité réside dans une protection face à l'autorité de certification qui n'est pas capable d'apprendre les messages envoyés aux utilisateurs, ni de savoir si ces derniers ont utilisé leur certificat. (Ce qui est schématisé dans la Figure 3, page xvii) La notion d'OSBE est un vrai dual de l'idée de SPHF, il suffit de considérer le langage $\mathcal{L}$ composé des chiffrés d'une signature valide qui sont durs à distinguer dans l'espace des chiffrés sous l'indistingabilité du schéma de chiffrement. Nous montrons comment construire une *Smooth Projective Hash Function* sur ce langage $\mathcal{L}$ et en déduisons un schéma d'OSBE dans le modèle standard. Nous prouvons ensuite la sécurité de notre construction vis à vis de celle du protocole de mise en gage (de chiffrement), de celle de la signature et de celle de la SPHF. On montre ensuite comment construire un schéma simple et efficace d'OSBE reposant sur l'hypothèse classique DLin. En faisant cela, on voit que notre nouvelle primitive est plus efficace que Groth-Sahai, et ne nécessite pas d'interaction supplémentaire dans le protocole.

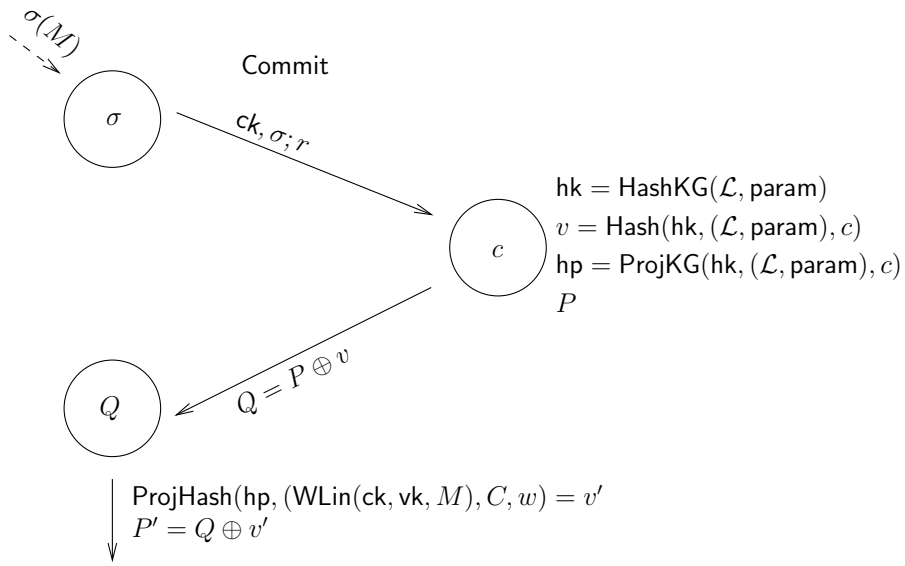


Figure 3: L'émetteur (*sender*) à droite veut envoyer un message P à l'utilisateur U à gauche, et ce si et seulement si U possède une signature $\sigma(M)$ valide sous vk.

Notre méthode ne nécessitant pas d'interaction supplémentaire, elle remplace agréablement Groth-Sahai dans les cas interactifs.

Les OSBE ne sont pas forcément les protocoles les plus utilisés en cryptographie, cependant cette construction est la clé de voûte de nos autres constructions, et les problèmes liés aux OSBE se retrouvent dans tous les protocoles d'échanges de clés. Ils sont donc une étape cruciale dans la mise en place des protocoles d'échange de clé basés sur les langages (LAKE) et la technique de preuves implicites à l'aide de Smooth Projective Hash Functions va prendre une place prépondérante dans nos schémas.

Échange de Clés Authentifié par des Langages

Nous proposons ensuite dans la Section 6.3, page 111 une nouvelle primitive qui englobe les notions précédentes de PAKE et poignées de mains secrètes. Elle est aussi très étroitement liée aux CAKE et nous l'appelons donc LAKE, pour *Language-Authenticated Key-Exchange*, puisque les participants établissent une clé commune si et seulement s'ils possèdent des accréditations qui appartiennent à des langages spécifiques (et possiblement indépendants) sans que les joueurs n'aient forcément à s'accorder

avant le protocole. Cette définition est nettement plus pratique que celle des CAKE de [CCGS10] mais les deux méthodes restent sensiblement similaires[13].

Les PAKE sont un cas particulier de LAKE, où chaque utilisateur possède un mot de passe pw_* et espère que l'autre possède un mot de passe dans le langage $\mathcal{L} = \{pw_*\}$. De même les poignées de mains secrètes disent que chaque utilisateur possède une signature valide selon la clé d'une certaine autorité, et espère que l'autre en possède une valide sous la clé d'une autorité. Nous permettons d'ailleurs une certaine granularité puisque l'autorité peut être la même pour les deux utilisateurs ou non, et où leur identité peut être publique ou non, ...

Cette nouvelle primitive permet donc entre autre des authentifications mutuelles préservant la confidentialité et des protocole d'échange de clé en permettant à deux membres d'un même groupe de s'authentifier mutuellement secrètement sans avoir à révéler leur groupe préalablement.

Pour définir la sécurité de cette primitive, nous utilisons le cadre d'*Universal Composability*, et précisons ce que nous appelons un langage ce qui permet de dissocier la partie publique de la relation (les informations publiques impliquées dans la vérification que l'utilisateur veut faire), de la partie privée que possède chaque utilisateur qui atteste l'appartenance aux langages. Nous fournissons une fonctionnalité idéale de LAKE et donnons des réalisations efficaces de cette primitive (pour notre famille étendue de langages) que nous prouvons sures sous diverses hypothèses dans le modèle standard (avec une CRS) et des corruptions statiques.

Nous produisons également une façon d'évaluer les coûts de tels protocoles en fonction des types de langages et de mises en gage utilisés, que ce soit en nombre d'éléments ou d'exponentiations. Nous utilisons les résultats usuels sur les conjonctions de langage, auxquels nous adjoignons nos nouveaux résultats sur les équations linéaires de couplages, nous traitons également les égalités. Nous expliquons aussi comment gérer les disjonctions avec des précautions supplémentaires.

Avec cette approche nous obtenons les protocoles de PAKE les plus efficaces dans le modèle standard avec CRS (améliorant considérablement les schémas de [ACP09, CCGS10] tant en terme de coût calculatoire qu'en nombre d'échanges) comme par exemple celui de la figure 4, page xix.

Nous améliorons également l'efficacité de plusieurs protocoles de CAKE [CCGS10], et élargissons les langages pour lesquels nous pouvons construire de tels schémas. Ce qui nous permet donc de construire des protocoles de poignées de main secrètes avec des propriétés renforcées (comme la forward-secrecy même avec une autorité corrompue), et nous obtenons aussi un protocole de PAKE résistant aux corruptions du serveur (en adaptant légèrement le protocole précédent, nous arrivons aussi à obtenir un protocole permettant de s'authentifier auprès d'un serveur avec un mot de passe dont il connait uniquement une fonction (*Verifier-based PAKE*)).

Outils supplémentaires

En poursuivant nos objectifs initiaux, nous rencontrons fréquemment des idées annexes qui sans être forcément primordiales pour résoudre le problème peuvent permettre des améliorations importantes de l'efficacité de plusieurs schémas.

Nous présentons ainsi dans la Section 2.6, page 35 certains de ces outils que nous avons dû construire et prouver.

Autour de la fonction de Waters

Avec notre approche modulaire, nous utilisons fréquemment la signature de Waters. Nous avons eu plusieurs résultats annexes autour de celle-ci. Par exemple, nous avons proposé et prouvé une version asymétrique de la signature dans [BFPV11], ce que nous rappelons dans la Section 2.6.3, page 41.

Un autre résultat important autour de la fonction de Waters réside sur sa programmabilité sur un alphabet non-binaire. Alors que dans [HK08], Hofheinz et Kiltz ont montré la $(2,1)$ et $(1,q)$-programmabilité sur un alphabet binaire, nous avons décidé de généraliser leur résultat pour pouvoir autoriser certaines forges homomorphiques comme dans le cas de nos réseaux de senseurs. Nous montrons facilement dans la Section 2.6.4, page 42 que la fonction n'est plus $(2,1)$-programmable sur un alphabet non-binaire. Par contre, en retravaillant leur approche probabiliste et en s'appuyant sur le Théorème Central Limite Local [DM95], nous avons pu montrer qu'elle restait $(1,q)$-programmable, tant que la taille t des blocs ne grandissait pas trop vite (Cela améliore le résultat précédent de [Nac05])[14]. Le théorème central limite local donne en quelque sorte une approximation de $Pr(\sum a_i = k)$ pour des

[13] *A priori*, toutes les primitives d'échange de clés actuelles doivent pouvoir être formalisées par les deux approches.

[14] Nous montrons que tant que $2^t = \mathcal{O}(\log \mathfrak{K})$ la fonction reste $(1,q)$-programmable

CRS: Un groupe $\mathbb{G}$ d'ordre prime p, avec six générateurs indépendants $(g_1, g_2, h, c, d, \zeta) \xleftarrow{\$} \mathbb{G}^6$, une fonction de hachage résistante aux collisions $\mathfrak{H}_K$[a], une application réversible $\mathcal{G}$ de $\{0,1\}^n$ dans $\mathbb{G}$, un schéma de signature $(\mathsf{KeyGen}_{\mathcal{S}}, \mathsf{Sign}, \mathsf{Verif})$. On notera $M = \mathcal{G}(m)$, et $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.

$(\mathsf{SK}_i, \mathsf{VK}_i) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}()$

$\ell_i = (\ell, \mathsf{VK}_i)$

$a_i, r_i \xleftarrow{\$} \mathbb{Z}_p$
$(\mathcal{C}_i, \mathcal{C}'_i) = \mathsf{DCSCom}(\ell_i, M_i, 1_{\mathbb{G}}; r_i, a_i)$,
$t_i \xleftarrow{\$} \mathbb{Z}_p, \chi_i = \mathfrak{H}_K(\mathcal{C}_i, \mathcal{C}'_i), \mathcal{C}''_i = g_1^{t_i}\zeta^{\chi_i}$

$\xrightarrow{\mathsf{VK}_i, \mathcal{C}_i, \mathcal{C}''_i}$

$r_j \xleftarrow{\$} \mathbb{Z}_p$
$\mathcal{C}_j = \mathsf{CSCom}(\ell'_j, M_j; r_j)$,
$\xi_i = \mathfrak{H}_K(\ell_i, \vec{u_i}, e_i)$,
$\mathsf{hk}_i = (\eta_i, \theta_i, \lambda_i, \mu_i) \xleftarrow{\$} \mathbb{Z}_p^4$,
$\mathsf{hp}_i = \mathsf{ProjKG}(\mathsf{hk}_i, (\mathcal{L}_{i,j}, \mathsf{param}), \xi_i)$

$\varepsilon \overset{?}{\neq} 0, z = r_i + \varepsilon a_i$,
$\xi_j = \mathfrak{H}_K(\ell, \vec{u_j}, e_j)$,
$\mathsf{hk}_j = (\eta_j, \theta_j, \lambda_j, \mu_j) \xleftarrow{\$} \mathbb{Z}_p^4$,
$\mathsf{hp}_j = \mathsf{ProjKG}(\mathsf{hk}_j, (\mathcal{L}_{j,i}, \mathsf{param}), \xi_j)$,
$H'_i = \mathsf{ProjHash}(\mathsf{hp}_i, \mathcal{L}_{\mathsf{priv}}, \ell'_i, \mathcal{C}_i\mathcal{C}'^{\varepsilon}_i; z)$,
$H_j = \mathsf{Hash}(\mathsf{hk}_j, \mathcal{L}_{\mathsf{priv}}, \ell'_j, \mathcal{C}_j)$,
$K_i = H'_i \cdot H_j$,
$\sigma_i = \mathsf{Sign}(\mathsf{SK}_i, (\ell_i, \mathcal{C}_i, \mathcal{C}'_i, \mathcal{C}_j, \varepsilon, \mathsf{hp}_i, \mathsf{hp}_j))$

$\xleftarrow{\mathcal{C}_j, \varepsilon, \mathsf{hp}_i}$

$\varepsilon \xleftarrow{\$} \mathbb{Z}_p^*$

$\xrightarrow{\mathsf{hp}_j, \mathcal{C}'_i, t_i, \sigma_i}$

$\chi_i = \mathfrak{H}_K(\mathcal{C}_i, \mathcal{C}'_i)$,
$H_i = \mathsf{Hash}(\mathsf{hk}_i, \mathcal{L}_{\mathsf{priv}'}, \ell'_i, \mathcal{C}_i\mathcal{C}'^{\varepsilon}_i)$,
$H'_j = \mathsf{ProjHash}(\mathsf{hp}_j, \mathcal{L}_{\mathsf{priv}'}, \ell'_j, \mathcal{C}_j; r_j)$,
$K_j = H_i \cdot H'_j$,
Si $\pi_i = g_1^{t_i}\zeta^{\chi_i}$ et si
$\mathsf{Verif}(\mathsf{VK}_i, (\ell_i, \mathcal{C}_i, \mathcal{C}'_i, \mathcal{C}_j, \varepsilon, \mathsf{hp}_i, \mathsf{hp}_j), \sigma_i)$
on définit la session comme *acceptée*.

Figure 4: Le protocole de PAKE découlant de notre méthodologie

[a]Une fonction $\mathfrak{H}_K$ est résistante aux collisions, s'il est dur de trouver 2 entrées distinctes x et y telles que $\mathfrak{H}_K(x) = \mathfrak{H}_K(y)$

variables indépendantes et identiquement distribuées a_i. (C'est une version du Théorème Central Limite dans laquelle la conclusion est renforcée la convergence de la loi devient une convergence ponctuelle localement uniforme des densités.)

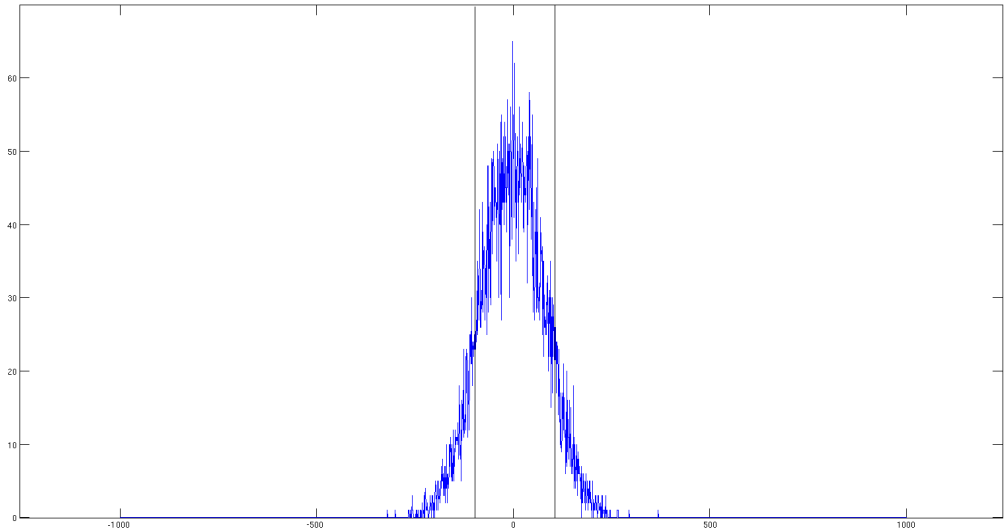


Figure 5: Nombres de sommes de marche aléatoire centrée, dont la somme est la valeur sur l'axe des abscisses, sur une expérience de 10000 lancers

On peut voir sur la figure 5, page xix, les conséquences du Théorème Central Limite Local, la majeure partie des sommes des marches centrées sont en 0, et il y a un nombre conséquents de somme dans un

intervalle pas trop grand autour de ce point, c'est-à-dire dans un intervalle centré en 0 et de taille le double de la racine du nombre de lancers (les bornes de l'intervalle sont ici symbolisées par les barres verticales).

Grâce à cela, on peut éviter d'avoir à découper un message bit par bit pour faire un haché Waters, et ainsi au lieu de calculer $\mathcal{F}(M) = u_0 \prod u_i^{m_i}$ pour chaque bit m_i, il est possible de considérer des blocs de bits M_i et donc de calculer : $u_0 \prod u_i^{M_i}$ ou même $u_0 u_1^M$ si l'espace des messages est de taille raisonnable. Une conséquence immédiate, en plus de la réduction du nombre de générateurs nécessaires, est qu'il est possible de signer plusieurs messages M_i à la Waters avec $\sigma = (\mathsf{sk}(\prod \mathcal{F}(M_i))^s, g^s)$ et de résister aux forges sous CDH tant que le nombre de messages est raisonnable par rapport au paramètre de sécurité, ce que nous exploitons dans une des applications de nos signatures en blanc.

Batch Groth-Sahai

La vérification d'une preuve Groth-Sahai est relativement lente (à cause du nombre de couplages à calculer). Nous étudions dans [BFI+10] une méthode pour *batcher* la vérification, c'est à dire vérifier plusieurs preuves simultanément. Nous accélérons ainsi la vérification sans mettre en défaut son intégrité. Nous partons du travail de Ferrarra *et al.* dans [FGHP09], et parvenons à construire une réponse valide à la question "Est-ce que toutes ces preuves sont valides ?". Dans l'affirmative alors hormis avec une probabilité négligeable toutes les preuves sont en effet valides, sinon une approche de type diviser pour régner permet de retrouver les affirmations fausses.

Nous détaillons dans la Section 2.6.1, page 35 notre approche basée sur la version DLin des preuves Groth-Sahai en détaillant les formules explicites de vérification par lot, puis nous les appliquons dans la Section 2.6.2, page 38 nos résultats à des protocoles existants. (Une fois encore nous omettons de détailler nos résultats en asymétrique)

Dans le tableau 3, page xx, nous présentons un rapide récapitulatif de nos résultats.

	Approche naïve	Calcul en Batch
SXDH		
Produits de Couplages	$5\,m + 3\,n + 16$	$m + 2\,n + 8$
Multiplications multi-scalaires dans $\mathbb{G}_1$	$8\,m + 2\,n + 14$	$\min(2\,n + 9, 2\,m + n + 7)$
Équations quadratiques	$8\,m + 8\,n + 12$	$2\min(m, n) + 8$
DLin		
Produits de Couplages	$12\,n + 27$	$3\,n + 6$
Multiplications multi-scalaires	$9\,n + 12\,m + 27$	$3\,n + 3\,m + 6$
Équations quadratiques	$18\,n + 24$	$3\,n + 6$

Table 3: Nombre de couplages par vérification, où n et m représentent le nombre de variables.

Étonnamment notre approche en plus d'améliorer la vérification d'un groupe de signatures s'avère aussi extrêmement efficace lorsqu'il s'agit de vérifier ne serait-ce qu'une seule signature, dans le cas de la signature de Groth [Gro07] nous passons de 68 à 11 couplages pour en vérifier une seule, le résultat le plus impressionnant provient des P-signatures de [BCKL08], les auteurs évaluaient initialement la vérification d'une preuve dans un contexte symétrique à un calcul de 126 couplages. Avec notre résultat, nous prouvons qu'il en faut simplement 12 pour une unique signature, et à peine $3\,n+9$ pour n signatures.

Multi Cramer-Shoup

Nous nous intéressons également au chiffrement Cramer-Shoup [CS98] dans la Section 2.6.5, page 46. Pour notre approche UC nous avons besoin de plusieurs modifications sur le schéma initial. Nous montrons en premier lieu que si nous utilisons le même haché pour plusieurs chiffrés simultanés alors le chiffrement reste CCA, cela nous permet de chiffrer un vecteur avec un seul calcul de haché ; même si cela peut sembler un peu annexe, ce résultat sera crucial dans notre méthodologie quand on voudra utiliser Cramer-Shoup sans faire exploser le nombre de clés de projection requises.

Ensuite, en s'inspirant de l'approche de Lindell dans [Lin11], nous considérons un chiffré Cramer-Shoup composé en fait de deux chiffrés $\mathcal{C}$ et $\mathcal{C}'$, et nous montrons qu'un tel chiffrement fournit une indistingabilité contre les attaques *à chiffrés choisis avec déchiffrement partiel*, notion que nous précisons

dans la Section 2.6.5, page 48.[15] où étant donné un chiffré $(\mathcal{C}, \mathcal{C}')$ un adversaire est autorisé à demander le déchiffrement de $\mathcal{C}$ et doit fournir le déchiffrement de $(\mathcal{C}, \mathcal{C}')$. $\mathcal{C}$ est un chiffré Cramer-Shoup valide, alors que $\mathcal{C}'$ non puisque le haché considéré est celui de $\mathcal{C}$. Ces résultats sont justifiés et prouvés dans la Section 2.6.5, page 46, tant sur Cramer-Shoup que sur sa version linéaire de [Sha07, CKP07].

De cette construction nous parvenons à élaborer un schéma de mise en gage équivocable dans la Section 2.6.6, page 52. Pour cela, on procède comme au dessus avec $\mathcal{C}$ un chiffré de $\vec{M}$ et $\mathcal{C}'$ un chiffré de $1_{\mathbb{G}}$. Ensuite on utilise une mise en gage de Pedersen sur un haché de $\vec{m}, \mathcal{C}'$ pour faire $\mathcal{C}''$ et on envoie $\mathcal{C}, \mathcal{C}''$. Notre interlocuteur nous envoie alors un challenge ε en réponse, auquel on répond avec $\mathcal{C}'$ et une preuve d'ouverture du Pedersen. Pour pouvoir ouvrir la mise en gage, on combine les aléas de $\mathcal{C}$ et $\mathcal{C}'$ à l'aide d'ε, puis on les envoie pour montrer que $\mathcal{C}\mathcal{C}'^{\varepsilon}$ est bien une mise en gage de M. Cela revient à faire un Sigma-Protocole sur notre mise en gage initiale.

Par exemple si l'on se base sur notre résultat sur le Cramer-Shoup linéaire, on obtient le protocole présent dans la figure 6, page xxi.

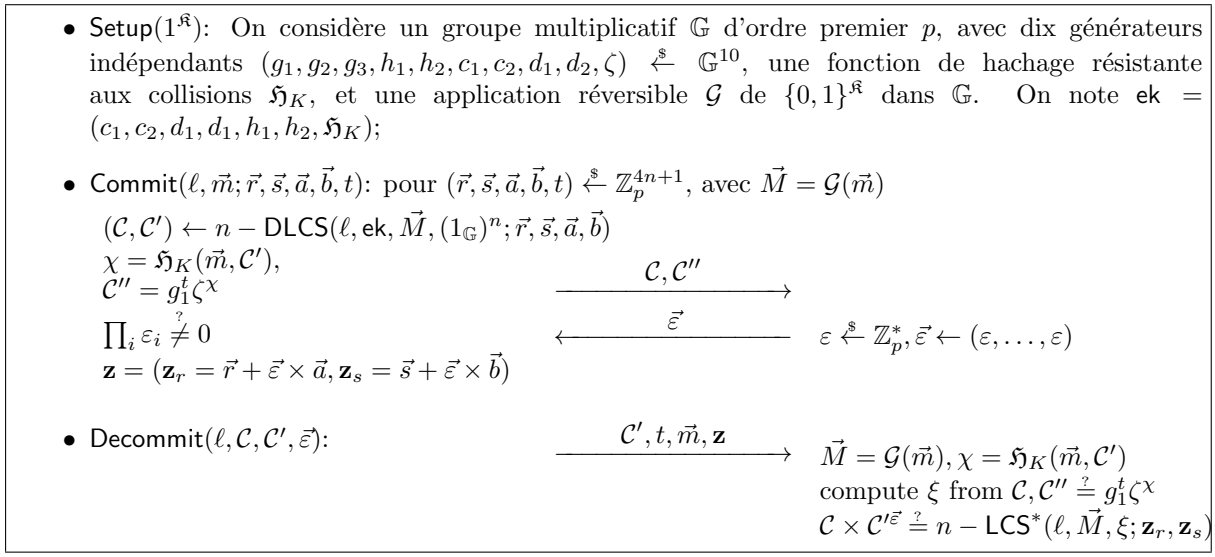


Figure 6: Schéma de mise en gage équivocable $n - \mathsf{DLCS}$

Cette mise en gage n'est pas vraiment extractable, car il est possible pour un $\mathcal{C}'$ habilement construit, c'est à dire chiffrant une valeur autre que $1_{\mathbb{G}}$, d'ouvrir la mise en gage finale sur une valeur autre que sur celle extractable après l'étape de mise en gage (c'est à dire contenue de $\mathcal{C}$). En connaissant le log discret des générateurs du Pedersen, il est possible d'adapter t pour que la vérification de $\mathcal{C}''$ fonctionne.

Bien que ce soit possible de corriger cette propriété en doublant la mise en gage initiale, cette propriété ne nous est pas nécessaire, il nous suffit en effet dans nos instanciations de simplement être sûr que dans le cas d'un tel $\mathcal{C}'$, alors il est impossible pour la personne mettant en gage de contrôler la valeur d'ouverture finale, ce dont nous pouvons être sûrs grâce au ε qui va porter sur un clair différent de $1_{\mathbb{G}}$.

Notre mise en gage demeure équivocable puisque si l'on connait les logarithmes delatifs de g_1 et ped, il est possible de tricher sur la mise en gage de Pedersen et de choisir son $\mathcal{C}'$ après avoir vu le ϵ.

Articles

Les travaux menant à cette thèse ont donné lieu à divers articles publiés ces trois dernières années :

Batch Groth Sahai [BFI+10]

avec Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, et Damien Vergnaud.

Nous revisitons la méthodologie générale Groth-Sahai pour les preuves non-interactives à divulgation nulle de connaissance (et à témoins indistinguables) en leur appliquant des techniques récentes de

[15] Informellement, l'adversaire choisit deux clairs M_0, M_1, voit un chiffré $\mathcal{C}$ de l'un deux, puis après d'éventuelles requêtes de déchiffrement il choisit deux autres clairs N_0, N_1 et reçoit un chiffré $\mathcal{C}'$ de l'un deux (de N_0 si M_0 était chiffré, de N_1 sinon). Après d'autres requêtes de déchiffrement, il doit alors décider quel clair a été choisi. Bien sûr, il échoue s'il pose une requête de déchiffrement impliquant $\mathcal{C}$ ou $\mathcal{C}'$.

vérification par lot. Cette méthode consiste en la vérification simultanée de plusieurs équations en échange d'une erreur négligeable.

Nous présentons d'abord des formules explicites de vérification par lot pour les équations Groth-Sahai génériques : ces méthodes montrent un gain d'un facteur 10. Et ensuite nous les appliquons à deux protocoles spécifiques : les signatures de groupe de Groth [Gro07] et les *P-Signatures* de Belenkiy, Chase, Kholweiss et Lysyanskaya de [BCKL08] où l'on voit des gains encore plus importants et cela dès la vérification de la première signature.

Signatures on Randomizable Ciphertexts [BFPV11]

avec Georg Fuchsbauer, David Pointcheval, Damien Vergnaud.

Les chiffrements randomisables autorisent n'importe quel utilisateur à transformer un chiffré en un chiffré nouveau du même clair. De manière analogue, une signature randomisable peut être transformée en une nouvelle signature sur le même message. Nous combinons ces deux primitives en une nouvelle qui peut être décrite ainsi : étant donné une signature sur un chiffré n'importe qui peut, sans connaissance de la clé de signature ou du message initial, randomiser le chiffré et adapter la signature sur le nouveau chiffré pour qu'elle reste publiquement vérifiable. De plus, quiconque possède la clé de déchiffrement et une signature sur un chiffré peut extraire la signature sur le clair associé. Comme cette notion contredit la résistance aux forges usuelle, nous avons défini une nouvelle notion plus souple qui dit juste qu'un adversaire ne doit pas pouvoir produire une signature sur un chiffré dont le clair associé n'est pas le clair d'un des chiffrés précédemment signés.

Nous instancions cette primitive à l'aide de preuves à la Groth-Sahai et de signatures Waters, et nous les prouvons sous des hypothèses classiques dans le modèle standard avec une CRS. Comme applications, nous montrons comment construire un système de vote efficace, non-interactif, et publiquement vérifiable sans que pour autant l'utilisateur ne puisse vendre son vote (Dans un tel schéma un votant ne peut pas prouver pour qu'il a voté, ce qui empêche donc la vente de votes). Notre primitive permet également un système de signature en blanc à interactions optimales qui en plus produit une signature classique à la fin.

Achieving Optimal Anonymity in Transferable E-Cash with a Judge [BCF$^+$11]

avec Sébastien Canard, Georg Fuchsbauer, Aline Gouget and Hervé Sibert, et Jacques Traoré[16].

Le terme de monnaie électronique (e-cash) désigne la monnaie échangée électroniquement. On désire reproduire dans ce contexte les principales fonctionnalités de la monnaie *traditionnelle*. Une de celles-ci est la *transmission décentralisée*, c'est-à-dire que pour transmettre de l'argent, il n'est pas nécessaire de le redéposer à la banque entre chaque intermédiaire. L'anonymat des utilisateurs dans de telles transactions est une propriété de sécurité qui a été longuement étudiée.

Ce papier propose le premier schéma de monnaie électronique, à la fois sûr et efficace, à parvenir au niveau d'anonymat le plus fort au sens de Canard et Gouget [CG08]. C'est-à-dire qu'il ne doit pas être possible pour des adversaires recevant une pièce de décider s'ils l'ont eu auparavant. Notre proposition est construite à partir des preuves Groth-Sahai et des signatures commutantes de Fuchsbauer [Fuc11].

Traceable Signature with Stepping Capabilities [BP12]

avec David Pointcheval.

Kiayias, Tsiounis and Yung ont introduit de nouvelles signatures pour résoudre des problèmes intervenant lors de l'ouverture dans les schémas de signatures de groupe. Ils voulaient permettre aux autorités de déléguer une partie de leurs capacités de détections à des sous-autorités. Mais au lieu de les autoriser à ouvrir n'importe quelle signature et ainsi à mettre en danger la vie privée des utilisateurs honnêtes, les sous-autorités ne peuvent savoir que si une signature appartient ou non à un utilisateur spécifique.

En 2008, Libert and Yung ont proposé une nouvelle instanciation de ces signatures qui fut la première à la fois efficace et prouvée sure dans le modèle standard. Nous présentons une nouvelle instanciation plus efficace (en nombre d'éléments de groupe envoyés). Nous ajoutons aussi une fonctionnalité supplémentaire permettant aux utilisateurs d'affirmer ou d'infirmer la parenté d'une signature.

Comme les signatures de liste sont relativement proche de cette primitive, nous en profitons pour adapter notre instanciation et ainsi présenter les premières signatures de liste sans oracle aléatoire.

[16] Ce papier n'est pas repris dans cette thèse, cependant les techniques utilisées sont inspirées de nos autres résultats gravitant autour de la méthodologie Groth-Sahai

Round-Optimal Privacy-Preserving Protocols with Smooth Projective Hash Functions [BPV12b]

avec David Pointcheval, et Damien Vergnaud.

Dans ce papier, nous démontrons que la notion de *smooth projective hash functions* peut s'avérer efficace pour créer des protocoles interactifs à la fois optimaux en nombre d'interactions et protégeant les données personnelles des utilisateurs. Nous montrons que cette approche est parfaitement applicable pour des schémas reposant sur des hypothèses usuelles dans le modèle standard avec une CRS, et s'avère plus efficace que la méthodologie Groth-Sahai.

Pour illustrer cela, nous construisons des schémas d'OSBE en considérant, le langage constitué des chiffrés de signatures valides sous une certaine clé, et des schémas signature en blanc optimaux en nombre d'interactions, en considérant cette fois le langage d'une conjonction de chiffrés d'un bit.

Nous voyons ainsi qu'à l'aide des *Smooth Projective Hash Functions*, il n'est plus nécessaire de reposer sur SXDH en asymétrique.

Compact Round-Optimal Partially-Blind Signature [BPV12a]

avec David Pointcheval, et Damien Vergnaud.

Les signatures partiellement en blanc ont de nombreuses applications autour de l'anonymat, comme dans les schémas de monnaie et de vote électroniques. Elles étendent la notion de signature en blanc classique en considérant des messages composés de deux parties l'une publique (commune à l'utilisateur et au signataire) et l'autre privée (choisie par l'utilisateur et signée à l'aveugle). Le signataire ne peut ensuite lier la paire message-signature à l'interaction initiale parmi d'autres paires avec la même partie publique.

Ce papier présente un tel schéma avec un nombre optimal d'interactions, qui en plus peut être complètement *masquant* (il n'existe aucun moyen calculatoire de connaître le message signé) dans le modèle standard muni d'une CRS et ce sous des hypothèses classiques (DLin et CDH ou leurs pendants en asymétrique). Ce schéma est plus efficace que les précédents tant en nombre d'interactions qu'en poids final et repose sur des hypothèses plus faibles, et comme précédemment il produit à la fin une signature Waters donc très efficace.

En plus de cela, nous montrons comment se passer de la mise en accord préalable requise par les schémas précédents pour définir la partie publique des messages. Dans notre protocole cette étape peut être mise de côté puisque techniquement les deux utilisateurs peuvent choisir la partie publique à la volée. Intuitivement, cette notion n'est pas plus faible, puisqu'un signataire pouvait refuser de signer tant que la partie publique ne lui convenait pas et inversement un utilisateur pouvait décider de ne pas utiliser la signature finale.

Avec cette propriété asynchrone, nous montrons aussi comment signer en blanc plusieurs messages venant de sources différentes, d'abord en les concaténant sans rien apprendre sur les messages initiaux avant d'obtenir la signature finale, puis en les additionnant, pour cela nous montrons un résultat sur la programmabilité de la fonction de Waters sur des alphabets non-binaires.

Efficient UC-Secure Authenticated Key-Exchange for Algebraic Languages [BCPV12]

avec Céline Chevalier, et David Pointcheval, et Damien Vergnaud.[17]

Les protocoles d'*échanges authentifiés de clés* (AKE) permettent à deux participants d'établir une clé partagée et cryptographiquement sure via un réseau non sécurisé en utilisant divers moyens d'authentifications, comme des clés cryptographiques, des petites (à faible entropie) clés secrètes ou des *accréditations*. Dans ce papier, nous donnons une méthodologie générale qui englobe toutes les primitives d'AKE précédentes comme les échanges de clé authentifiés par mots de passe (PAKE), ou des protocoles de poignées de main secrètes. Nous l'appelons LAKE, le L étant pour désigner que nous reposons sur des Langages (*Language Authenticated Key-Exchange*).

Nous présentons en premier la primitive générale dans le modèle UC, pour ensuite montrer que l'approche de Gennaro-Lindell peut permettre d'y répondre facilement. Mais pour cela, nous avons besoin de *smooth projective hash functions* sur de nouveaux langages, dont l'implémentation efficace est d'un intérêt indépendant. Nous présentons de telles SPHF pour les langages définis comme étant une combinaison d'équations linéaires de produits de couplages.

Combiné avec un système de mise en gage efficace dérivé de schéma de Lindell nous obtenons ainsi une réalisation pratique d'un protocole de poignées de main secrètes, mais aussi des protocoles de *Credential-Authenticated Key Exchange*. Nous obtenons aussi des protocoles de PAKE extrêmement efficaces et

[17] En soumission

présentons même une version de ceux-ci où le serveur ne stocke pas le mot de passe directement mais seulement une fonction à sens unique de celui-ci pour protéger l'utilisateur en cas de corruptions.

Nos protocoles sont tous prouvés sûrs dans le modèle UC, sous l'hypothèse décisionnelle linéaire (DLin). ⋆

Chapter 1

Introduction

Contents

Until recent years, cryptology and more precisely cryptography was used to refer to encryption, i.e. to allow the transmission of information between parties over an insecure channel. The first methods were simple and revolved around *substitution* and *transposition* ciphers, Spartan were claimed to use scytale around 900 BC, or Caesar's cipher where one simply shifts each letter with the third one afterwards in the alphabet.

Throughout the history there have been a fight between cryptographers and cryptanalysts. This fight transformed the goals of cryptography. While the primary goal was to keep unintended parties from learning the contents of the message being exchanged, *data authenticity* became more and more a concern as it aims at guaranteeing that the content of the message has not be tampered with, or that the sender is indeed the correct sender. One of the challenge of modern cryptography is to design new protocols which fulfil those two goals simultaneously, any user wants to be able to access to his information when needed in a secure way, without telling to mush to the server with whom he is communicating.

These new goals have widen the expectation relying on cryptography. With these new considerations the cryptography has transitioned from an *ad hoc* approach where designers tried to solve problems as a whole, to a well established science with a more *modular* approach where small building blocks are considered and proven independently and then combined into a more complex scheme.

There is a duality between those two approaches, the first one often yields efficient results bound to more complex security proofs, while the latter may be less efficient but with a more intuitive construction which leads to an easier security analysis.

Throughout this thesis we are going to try to bridge the gap between those approaches, by proposing new modular constructions which are still efficient, and respect the privacy of the users.

Our work can be divided into two main parts, the first one shows that the classical blocks that are encryption, signature, and non-interactive zero-knowledge (NIZK) proof systems can be combined efficiently, and even commuted to implement various classical schemes (like Group signatures, or Blind signatures), while the second one aims to emphasize the possibilities of implicit proofs of knowledge through many interactive protocols (like Oblivious Signature-Based Envelope, PAKE, CAKE or Secret Handshakes).

Thanks to digital signatures, encryptions and NIZK, one can achieve many cryptographic primitives providing means of identification, authentication while simultaneously preserving anonymity. We first

focus on group signature [Cv91]: they allow member of a group, managed by a group manager, to sign anonymously on behalf of the group. To prevent abuse, this anonymity can be revoked by an additional authority. We then consider blind signatures [Cha83] which are quite the opposite, the user's identity is known while the message signed is hidden. They were introduced for electronic cash, and used in electronic voting where the value of a ballot should be hidden while his validity remains guaranteed.

A huge category of interactive protocols between two participants can easily be summed up by saying that either a user A wants to give a message to B, possibly at his request, only if B already possesses some information, either two users A and B wants to established a shared secret if each possesses some different information. Those ideas includes many common protocols, some blind signatures requires that B possesses the message signed, (anonymous) credentials allow B to access information only if he is allowed to without revealing is true identity to A, Password Based Key Exchange protocols allow two user to established a shared secret key if they both posses the same password, while Secret Handshakes requires them to belong to the same organisation.

1.1 A Brief History of Cryptography

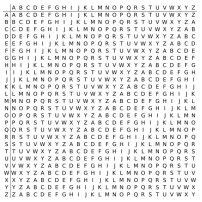
1.1.1 Ancient Time

The oldest known text to contain some cryptographic aspect seems to be dated from the ancient Egypt around 4000 years ago. The tomb of Khnumothep in Menet Khufu is covered with unusual hieroglyphic instructions.

Other known cryptographic uses at this time were either *transposition* ciphers like the Scytale used by the Spartans in 900 BC, where you had to roll a piece of paper around some cylinder, like a staff, write your message on it, unroll the paper, and only a person with a same-sized staff can easily read the message.

There were also lots of *substitution* ciphers, the more famous being Caesar's cipher (cf [Sue21][1]) where each letters is shifted by three (caesar becomes fdhvdu), there were also three kind present in the Bible (Atbash, Albam and Albah) who were mirrors in the alphabet (a shifts with z, b with y, ...), half mirror (a with m, b with l, ..., n with z, o with y...), or some kind of Ceasar's Cipher, but all of them were convolutions; and there was the *mlecchita-vikalpa* in India, who was a Caesar's cipher but with a shift of 1 (love becomes mpwf).

1.1.2 Medieval Era



There were quite few improvements. Al-Khindi proposed the first known cryptanalysis of ciphers, using a technique based on the frequency analysis around 800 AD. All known ciphers remained vulnerable until the first polyalphabetic ciphers. Leon Battista Alberti explained them clearly around 1467; Johannes Trithemius proposed the tabula recta in *Poligraphia*, thanks to which Blaise de Vigenère devised a practical polyalphabetic system (the Vigenère cipher) which was to be broken by Charles Babbage only around 1854.

1.1.3 Modern Era

In 1883, Auguste Kerckhoffs presented the premise of the modern cryptography approach in [Ker83] by stating the 6 rules reminded in figure 1.1, page 3:

While the last rules are really intuitive and call for simplicity and low communication cost, the second rule remains the most important one. It was the end of *Security through Obscurity*, cryptographic cryptosystem should remain secure even if the enemy knows them. Or as later said Shannon: "The enemy knows the system". While in practical applications this is not always the case yet (some encryption on DVD for example), modern research starts from this premise. This coincides with the transition of cryptography from pure military / diplomatic use, to a more everyday use.

Two major milestones allow to go a little further: the introduction of public-key cryptography by Diffie and Hellman [DH76] where no initial secret has to be shared to start a conversation over an insecure channel, and zero-knowledge proofs [GMR89] which allowed user to prove the knowledge of a secret without revealing it.

[1] "Exstant et ad Ciceronem, item ad familiares domesticis de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset; quae si qui investigare et persequi velit, quartam elementorum litteram, id est D pro A et perinde reliquas commutet."

1. The system must be practically, if not mathematically, indecipherable;
2. It must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience;
3. It must be easy to communicate and remember the keys without requiring written notes, it must also be easy to change or modify the keys with different participants;
4. The system ought to be compatible with telegraph communication;
5. It must be portable, and its usage and function must not require the concourse of several people;
6. Finally, regarding the circumstances in which such system is applied, it must be easy to use and must neither require stress of mind nor the knowledge of a long series of rules.

Figure 1.1: Translation of the 6 rules stated by Kerckhoffs about modern cryptography

In chapter 2, page 15 we are going to precise those notions, and then build protocols based on them.

1.2 Digital Signatures Enhanced with NIZK

1.2.1 Motivation

Group Signatures

A good example of modular design comes from the BSZ model for dynamic group signatures from [BSZ05]. In this model three kinds of participants, the issuer, the opener and group members play. To become a member of the group, a user has to *join* the group by interacting with the issuer (or at least someone who possesses the *issuer key*), at the end of the interaction the user obtains a private signing key. With this key, he can now sign in name of the group and anyone can verify the validity of his signature thanks to the *group public key*. If something is litigious, the *opener* can open a group signature: on input of the opening key, and the incriminated signature, an algorithm outputs the identity of the signer and a proof of correct opening.

There are two main security requirements, first *anonymity* which says that without extra-knowledge it should be hard to know who produced a signature, and then *unforgeability* which simultaneously states that any valid signature should be opened to a registered user, and that no coalition can produce a valid signature and a proof of correct opening that incriminates an honest user.

To show that such model can be achieved, Bellare *et al.* give the following definition. Given a signature scheme, an encryption scheme, and non-interactive zero-knowledge proofs (those notions are properly defined later on throughout Section 2, page 15), the setup produces a pair of verification and signing keys, a pair of encryption and decryption keys. To join a group, a user produced a personal signature key pair, and gets from the issuer a certificate, in other word a signature under the issuer key, on the verification key. A member can then produce a group signature simply by signing the message with her personal signing key, encrypting her certificate, her verification key, and this signature, and then producing those ciphertexts together with a non-interactive zero-knowledge proof that the certificate and signature in the plaintext are indeed valid. The opening is done by decrypting the ciphertexts, where the verification key gives the user's identity and the signature corresponds to the unforgeable proof.

The anonymity is quite intuitive given that the group signature is only composed of a NIZK [FS87] and ciphertexts, both leaking no information. The unforgeability of the group signatures comes from the unforgeability of the underlying signature scheme, the signature from the issuer can not be forged so the traceability is guaranteed, and the signature under a user signing key can not be forged which induces the non-frameability.

Such modular approach is also the corner stone of our constructions, where we also combine together compatible signature scheme, encryption scheme, and non-interactive zero-knowledge proofs.

The Groth-Sahai Methodology

Until recently, the random oracle model was commonly used in practical instantiations who needed either non-interactive zero-knowledge proofs or non-interactive witness-indistinguishable proofs, mostly because the generic instantiations in the standard model were way too inefficient. Groth and Sahai proposed a way to produce *efficient* and practical NIZK and non-interactive witness-indistinguishable (NIWI [FS90],) proofs for (algebraic) statements related to groups equipped with a bilinear map.[2] In particular, they give proofs for the simultaneous satisfiability of a set of equations. They proposed three instantiations of their system based on different (mild) computational assumptions: the subgroup decision problem (SD), the symmetric external Diffie-Hellman problem (SXDH) and the decision linear problem (DLin). Each one of these has already given rise to several applications in recent years (*e.g.* group signature schemes [BW06, BW07, Gro07] or blind signatures [AFG+10, BFPV11]). Their construction starts by building a witness-indistinguishable proof of satisfiability of certain equation. To do so, one commit to a witness and than constructs proofs which claim that the witness satisfy the equation. They divide the equations into three main kinds (pairing product, multi-scalar multiplication, and quadratic equation), in the first one, where equations are composed of product of pairings applied to the variables and constants for the group, a simulator can extract the witness from the commitment and so the proofs are in fact proofs of knowledge.

Groth-Sahai Methodology and Signatures

The first practical schemes to use Groth-Sahai methodology, or an idea close-enough, were the Boyen and Waters group signatures [BW06] where the proofs were in fact derived from the original techniques from [GOS06b]. Belenkiy *et al.* then apply the Boneh-Boyen [BB04] transformation to the Boyen-Waters scheme in [BCKL08] to obtain fully secure signatures. To construct anonymous credentials, they commit to the message and a signature and then thanks to Groth-Sahai proofs they prove that their content is valid. However as the message is a scalar and not a group element, the extraction can not be done properly, which induce either a weakening in the security notion, where the consider *F-unforgeability*, or a bit per bit commitment of the message.

With that in mind, this gives us some requirements on the blocks we are going to use if we want to be able to combine them properly.

- The signature scheme has to be existentially unforgeable against chosen-message attacks,
- The signature should be composed of elements in a bilinear group,
- The message should either be public, or a group element, or a short scalar, (for the extractability)
- The signature verification should be done with pairing product equations.

Of course, as this is one of our primary goal, we want all the building blocks to be efficient.

Traceable Signature

Traceable signatures schemes were introduced by Kiayias, Tsiounis and Yung in [KTY04] in order to solve traceability issues in group signature schemes. They wanted to enable authorities to delegate some of their detection capabilities to tracing sub-authorities. Instead of opening every single signatures and then threatening privacy, traceable signatures allow the opener to delegate the tracing decision for a specific user without revoking the anonymity of the other users: the opener can delegate its tracing capability to sub-openers, but against specific signers without letting them trace other users. This gives two crucial advantages: on the one hand tracing agents (sub-openers) can run in parallel; on the other hand, honest users do not have to fear for their anonymity if authorities are looking for signatures produced by misbehaving users only. The security properties are the same as those for the group signatures, excepts that the correctness is superseded by if a signature opens to i then it also leads to a positive answer for the trace procedure under user i's related keys. This is in the same vein as searchable encryption [ABC+05], where a trapdoor, specific to a keyword, allows to decide whether a ciphertext contains this keyword or not, and provides no information about ciphertexts related to other keywords.

In 2008, Libert and Yung in [LY09] proposed the first traceable signature schemes proven secure in the standard model with limited possibilities for the signer.

[2] A bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ is a composed of multiplicative groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p generated respectively by $g_1, g_2, e(g_1, gb)$ where e is a pairing, i.e. a non-degenerated bilinear form from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. If $\mathbb{G}_1 = \mathbb{G}_2$ such group is said to be symmetric.

List Signature

List signatures were introduced by Canard et al. in [CSST06]. They let users sign anonymously, in an irrevocable way, but grant linkability in a specific time-frame: no one can trace back the actual signer, but if a user signs two messages within a specific time-frame, the signatures will be linkable.

Security properties are similar to group signatures: Anonymity, given two honest users i_0 and i_1, the adversary should not have any significant advantage in guessing which one of them has issued a valid signature, and Soundness, an adversary can produce at most one valid signature per time-frame per corrupted player, Since then, it has been an open problem to know if there was any way to construct such a list signature scheme in the standard model.

Such signature may find application in voting protocols where one want to detect double voting without opening each ballot.

Round-Optimal Blind Signature

Blind Signatures were introduced by Chaum in [Cha83]. They allow a user to obtain a signature on a message so that the signer can not decide which interaction has lead to a specific signature. This leads to two main security requirements, the Blindness where given two interactions producing two valid signatures on two different messages an adversary should not be able to efficiently guess which interaction produced which signature, and the unforgeability where after q interactions an adversary should not be able to produce $q+1$ valid signatures (on different messages). They were further formalized in [JLO97, PS00], and even instantiated without random oracles many times (like in [CKW04, Oka06]). However all those approaches were not round optimal, as they had more than one round of communication. Fischlin [Fis06] gives a generic construction of *round-optimal* blind signatures which has been efficiently instantiated recently [Fuc09, AFG+10]. To prevent the signer from linking a blind signature to the signing session, they define a blind signature as a (non-interactive) *proof of knowledge* of a signature. This makes blind signatures significantly longer than signatures of the underlying scheme, and so up to now instantiation of round-optimal blind signatures in the standard model has been an open problem, if we expect to obtain the user to exhibit a regular signature at the end.

In Fischlin's scheme a blind signature is a proof of knowledge of a signature on a ciphertext together with a proof that the ciphertext decrypts to the message. In the scheme in [Fuc09], the user obtains an actual signature on the message, of which he proves knowledge.

We go one step further: again, the user can extract a signature on the message; but instead of making a proof of knowledge, it suffices to simply randomize it to make it unlinkable. A blind signature has therefore the same format as the underlying signatures and, in addition to being round-optimal, is thus short.

1.2.2 Instantiations and Applications

In Part I, page 54, we give concrete instantiations build on simple blocks, and show their respective security in the standard model.

Around Group Signatures

First in Chapter 3, page 55, we combine classical blocks to solve open problems in the standard model. We start in Section 3.2, page 57 by building a new efficient traceable signature which allows user both to step in and out of a signature. Basically in addition to the classical properties of a traceable signature, we provide the following: given a signature, a user is able to claim, and justify this claim, that this signature is her, or that he did not do it. To do that we start by recalling the various security notions, and precise what we expect from *stepping capacities*, i.e. a user should be able to prove/disprove he is the author of a signature. We then build a scheme answering those goals from standard building blocks, namely the certificate from [DP06][3], the Dodis-Yampolskiy Verifiable Random Function [DY05] and Waters Signature that we prove in an asymmetric bilinear group. We then combine these elements with the Groth-Sahai methodology to achieve anonymity.

The Delerablée-Pointcheval [DP06]-like certificate will allow delegation of tracing, since a trapdoor, not enough for signing, enables tracing decision between a signature and an alleged user. Users will also be able to confirm (step-in) or deny (step-out) being the actual signer, using their signing key only, in

[3]Such certificate is of the form $(x_i, y_i, A_i = (kg^{y_i})^{1/\gamma+x_i})$ where γ is the secret key of the certification authority, $\Omega = g^\gamma$ the group public key, and y_i the user secret key

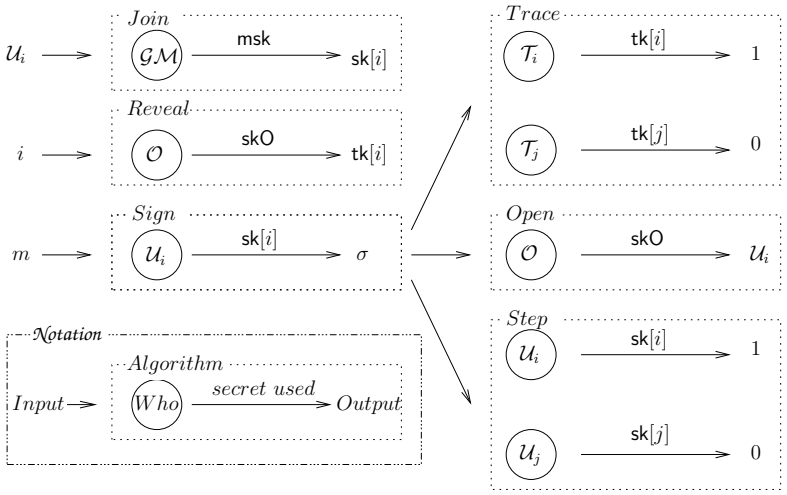


Figure 1.2: An Extended Traceable Signature Scheme

a convincing way, which is a new attractive property. To achieve this, we define the notion of unique identifier, related to each signature, and specific to the user and an additional input.

We need to define a new assumption, the q-Hybrid Hidden Strong Diffie-Hellman ($q-\mathsf{HHSDH}$). While this assumption is new, we can show it is still quite reasonable, as under the Knowledge-of-Exponent assumption we show that it is equivalent to the q-Strong Diffie-Hellman.[4]

The final instantiation more than halves the communication cost of traceable signature. The key to this constructions is the creation of an ephemeral identifier ID based on the user secret key thanks to the Dodis-Yampolskiy Verifiable Random Function. Given the correct trapdoor a sub-authority is able to trace a specific signer thanks to this ID. The Groth-Sahai proofs are used to prove that the secret key used in the identifier is registered by the authority, and the Waters signature is simply used to sign the message and provides unforgeability.

We then further improve this scheme in Section 3.3, page 66. For that we have to clarify the original security requirements by defining proper security games. Granted the previous technique of unique identifier, we can give a positive answer to the open problem of list signatures in the standard model: if we make the unique identifier specific to the user and the time-frame, in a deterministic way, then two signatures by the same user within the same time-frame will have the same identifier, which provides linkability.

Signature on Randomizable Ciphertexts

Eventually dropping the anonymity of the signer, we then consider a totally different approach where we want to hide the message. For that purpose, we propose in Chapter 4, page 70 a new primitive called Signature on Randomizable Ciphertexts: Given a signature on a ciphertext, anyone, knowing neither the signing key nor the encrypted message, can randomize the ciphertext and *adapt* the signature to the fresh encryption.

A pair of a ciphertext and a signature on it can thus be randomized simultaneously and consistently.

Since adapting a signature on one ciphertext to a signature on another ciphertext contradicts the standard notion of unforgeability for signatures, we define a weaker notion, which still implies the security of our applications: unforgeability of signatures on randomizable ciphertexts means that the only thing an adversary can do is produce signatures on encryptions of messages of which he already knows a signature on an encryption; but he cannot make a signature on an encryption of a *new* message. Formally, no adversary can, after querying signatures on ciphertexts of its choice, output a signature on a ciphertext whose decryption is different from the decryption of all queried ciphertexts.

We then extend our primitive to *extractable* signatures on randomizable ciphertexts: given the decryption key dk, from a signature $\sigma(C)$ on a ciphertext C one can *extract* a signature $\sigma(M)$ on the encrypted plaintext M. This enables the user in a blind-signature scheme to recover ($\mathsf{SigExt}_{\mathcal{SC}}$) a signature on the message after the signer has signed ($\mathsf{Sign}_{\mathcal{SC}}$) an encryption of it.

[4] Those notions are described in Section 3.2.2, page 61

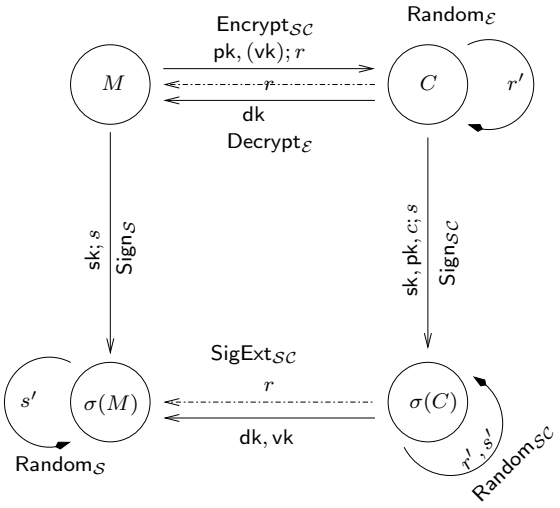


Figure 1.3: (Strong) extractable signatures on randomizable ciphertexts

Symmetric Pairing	$\mathbb{G}$
Waters + Linear	$9\,k + 44$

Asymmetric Pairing	$\mathbb{G}_1$	$\mathbb{G}_2$
Waters + ElGamal	$6\,k + 9$	$6\,k + 7$

Table 1.1: Number of group elements required for a ciphertext-signature pair depending on the chosen instantiation

This notion is called strong, when using the randomness r from the encryption algorithm $\mathsf{Encrypt}_{\mathcal{SC}}$, one can also extract the signature on the plaintext.

We give several instantiations of extractable signatures on randomizable ciphertexts, all of which are based on weak assumptions. Our constructions use the following building blocks, from which they inherit their security: Witness-indistinguishable Groth-Sahai proofs for languages over pairing-friendly groups [GS08] and Waters signatures derived from the scheme in [Wat05] and used in [BW06]. Since verification of Waters signatures is a statement of the language for Groth-Sahai proofs, these two building blocks combine smoothly.

The first instantiation of our new primitive is in symmetric pairing-friendly elliptic curves and additionally uses linear encryption [BBS04]. Both unforgeability and semantic security of this construction rely solely on the decision linear assumption (DLin). The construction is nearly straightforward. If one would commit to a Water hash of message together with a bit-per-bit proof of knowledge of this message, then receives a Waters-like signature of this commitment (i.e. the signer takes $\mathcal{C}_3$ which carries the message, and sends $h^x\mathcal{C}_3^s, g^s$, and then use the decyrption key to recover the signature using the decryption key, one would obtain a scheme very close to ours. The main difference is that we show for unforgeability reasons that the first user should also commit to a value based on the verification key raised to the scalars used in the message commitment.

To avoid redundancy, we omit the instantiation in *asymmetric* bilinear groups, using ElGamal encryption and the SXDH variant of Groth-Sahai proofs (available in the full version [BFPV11]). While is has improved efficiency, this setting requires to transfer Waters' signature scheme to asymmetric groups. Whereas standard Waters signatures are secure under the computational Diffie-Hellman assumption (CDH), we prove our variant secure under a slightly stronger assumption, we term CDH^+, where some additional elements in the second group are given to the adversary.

The following table details the size of a ciphertext-signature pair, where the parameter k denotes the bit length of a message, first in case of an instantiation on a symmetric elliptic curve, and then on an asymmetric one:

Using our new primitive, we immediately obtain a reasonably efficient round-optimal blind-signature scheme based on standard assumptions. Moreover, exploiting the fact that our encryption is homomorphic, we construct a non-interactive receipt-free universally verifiable e-voting scheme as follows: the user encrypts his vote, proves its validity, and sends the encryption, a signature on it, and the proof to the voting centre. The latter can now randomize the ciphertext, *adapt* both the proof and the user's signature,

and publish them. After the results are announced, the user can verify his signature, which convinces him that the randomized ciphertext still contains his original vote due to our notion of unforgeability; however he cannot prove to anyone what his vote was.

Later on, we reconsider the Blind-Signature scheme, to further alter it. First, in Section 4.4.1, page 82 we consider a version which achieves partially-blind signatures with perfect blindness while remaining under standard assumption, using the perfectly hiding instantiation of Groth-Sahai commitments [GS08]. We also widen the model of partially-blind signatures to supplement the predetermined communication with an on-the-fly public information generated by the signer: the signer can simply include it during the signing process, even if the user does not want this extra information. In the latter case, the user can simply discard the signature and start anew. We call this new primitive *signer-friendly partially-blind signatures*. This new notion allows to skip the prior agreement and allow the public information to be set on-the fly. Of course this new notion doesn't forbid any kind of prior agreement on the public part, it just strengthens the existing notion.

It is now possible to get rid of the prior agreement on the common piece of information in the signed message and our instantiation allows the signer to do so in a round-optimal way. These two constructions being compatible, we can present a round-optimal partially-blind signature with perfect blindness. Our protocol does not need any pre-processing for the public part of the message. Basically both the user and the signer can choose a piece of the public part, but instead of having a computational overhead for the agreement both can simply choose during the 2 flows what they want. The signer can always refuse to sign something where the user's public information doesn't suit him and the user can always choose not to exploit an uninteresting signature, so a protocol should avoid to waste communication costs when one can manage without any security loss to stay in a two-flows protocol.

And then, discarding the perfect blindness, we take advantage of this asynchronous property (the user and the signer can independently choose their inputs) in Section 4.4.2, page 87 and we consider the new context where the message to be signed comes from several independent sources that cannot communicate together. We first present a way to obtain a signature on the concatenation of the input messages. We also present a shorter instantiation which gives a signature on the sum of the input messages. Such a sum can be useful when working on ballots, sensor informations, ... Since we still apply the Waters signature, this led us to consider the Waters function programmability over a non-binary alphabet, in a similar way as it was done in [HK08] for the binary alphabet. We prove a negative result on the $(2, 1)$-programmability, but a nice positive one on the $(1, poly)$-programmability, which is of independent interest.[5]

1.3 Smooth Projective Hash Function, and Implicit Proof of Knowledge

1.3.1 Motivation

Our previous result while improving the efficiency of previous schemes lead us to a new question. Why are we using non-interactive proof of knowledge, in an interactive protocol? Is there a way to preserve round optimality while using a possibly interactive proof. In standard interactive proof of knowledge the last interaction is a message sent by the prover to the verifier, while this may seem intuitive this induces an extra round, as in our schemes most of the time, the prover initiates the procedure and so we would need at least 3 flows to plug such proof. We instead focus on implicit proof of knowledge, where the verifier may not necessarily learn the veracity of the prover statement, but should be convinced that only an honest prover can exploit the information requested, to try to build round-optimal privacy-preserving interactive protocols.

Smooth Projective Hash Function

Smooth projective hash functions (SPHF) were introduced by Cramer and Shoup [CS02] for constructing encryption schemes. A projective hashing family is a family of hash functions that can be evaluated in two ways: using the (secret) hashing key, one can compute the function on every point in its domain, whereas using the (public) *projected* key one can only compute the function on a special subset of its domain. Such a family is deemed *smooth* if the value of the hash function on any point outside the special subset is independent of the projected key. If it is hard to distinguish elements of the special subset from non-elements, then this primitive can be seen as special type of zero-knowledge proof system for membership in the special subset. The notion of SPHF has found applications in various contexts in cryptography (*e.g.* [GL03, Kal05, ACP09]).

[5] cf Section 2.6.4, page 42.

We present some other applications with privacy-preserving primitives that were already inherently interactive. And we reconsider the manageable set of languages with those functions, and show that it can work in some cases where the Groth-Sahai methodology can not be used.

Blind signatures

As explained in the previous section, we are going to present Signatures on Randomizable Ciphertexts which allow Round-Optimal Blind Signature. However this construction is heavily reliant on the Groth-Sahai methodology, so while it was a neat improvement when compared to existing solution who were not producing classical signatures, it does not really fit our efficiency requirements. And as one of our first results we are going to apply our new methodology for implicit proofs of knowledge to show the impact on efficiency without weakening the security.

There is a growing field of protocols,around *automated trust negotiation*, which includes Oblivious Signature-Based Envelope [LDB03], Secret Handshakes [BDS+03], Password-based Authenticated Key-Exchange [BM93, BPR00], and Hidden Credentials [BHS04]. Those schemes are all closely related (as if you tweak two of them, you can produce any of the other protocols [CJT04]).

Oblivious Signature-Based Envelope

Oblivious Signature-Based Envelope (OSBE) were introduced in [LDB03]. It can be viewed as a nice way to ease the asymmetrical aspect of several authentication protocols. Alice is a member of an organization and possesses a certificate produced by an authority attesting she is in this organization. Bob wants to send a private message P to members of this organization. However due to the sensitive nature of the organization, Alice does not want to give Bob neither her certificate nor a proof she belongs to the organization. OSBE lets Bob sends an obfuscated version of this message P to Alice, in such a way that Alice will be able to find P if and only if Alice is in the required organization. In the process, Bob cannot decide whether Alice does really belong to the organization. The security is usually defined through 2 mains points: Oblivious, if $\mathcal{R}_0$ knows and uses a valid signature σ and $\mathcal{R}_1$ does not use such a valid signature, the sender cannot distinguish an interaction with $\mathcal{R}_0$ from an interaction with $\mathcal{R}_1$,and (Weakly) semantically secure, if $\mathcal{S}_0$ owns P_0 and $\mathcal{S}_1$ owns P_1, the recipient that does not use a valid signature cannot distinguish an interaction with $\mathcal{S}_0$ from an interaction with $\mathcal{S}_1$. We strengthen those notions by superseding the oblivious notion by allowing the authority to be the adversary.

Secret Handshakes

The concept of *Secret Handshakes* has been introduced in 2003 by Balfanz, Durfee, Shankar, Smetters, Staddon and Wong [BDS+03] (see also [JL09, AKB07]). It allows two members of the same group to identify each other secretly, in the sense that each party reveals his affiliation to the other only if they are members of the same group. At the end of the protocol, the parties can set up an ephemeral session key for securing further communication between them and an outsider is unable to determine if the handshake succeeded.

Password-Authenticated Key Exchange

Password-Authenticated Key Exchange (PAKE) was formalized by Bellovin and Merritt [BM92] and followed by many proposals based on different cryptographic assumptions (see [ACP09, CCGS10] and references therein). It allows users to generate a strong cryptographic key based on a shared "human-memorable" (*i.e.* low-entropy) password without requiring a public-key infrastructure. In this setting, an adversary controlling all communication in the network and able to corrupt participants at any time should not be able to mount an off-line dictionary attack.

A variation was introduced recently, where one user knows a password, and the other a one way function of it. This way if a server is compromise the password itself is not revealed.

Credential-Authenticated Key Exchange

More recently, *Credential-Authenticated Key Exchange* (CAKE) were presented by Camenisch, Casati, Groß and Shoup [CCGS10]. In this primitive, a common key is established if and only if a specific relation is satisfied between credentials hold by the two players.

1.3.2 Results and Instantiations

Our main contribution in Part II, page 92 is to define a methodology to do round-optimal implicit proof of knowledge. We present all our instantiations in a symmetric group, under the DLin assumption and possibly CDH when we require a signature. However the same can be done in an asymmetric setting using XDH and CDH^+, and no longer SXDH.

Smooth Projective Hash Functions on Commitment

To do our implicit proofs of knowledge we define Smooth Projective Hash Functions on Commitment. If one would want to draw a parallel with existing Groth-Sahai methodology. We assume we possess some word W and a witness w that it belongs to a language $\mathcal{L}$. We then commit to W and then send this commitment. The *Verifier* will then compute a projection key hp, a hash key hk, his view H of the hash value. He then sends the prover hp together with the requested information masked by his H. The prover using his witness w and hp can now compute his own view of the hash value H'. If indeed w is a witness that the plaintext W belongs to $\mathcal{L}$ then $H' = H$ and he can recover the requested information, otherwise we show he does not learn anything.

Manageable Languages

A direct consequence of those new protocols is to try to extend the set of languages handleable by our methodology. Abdalla *et al.* have shown how to manage conjunction and disjunction of languages in [ACP09], we now have to further consider basic language. In Chapter 5, page 93, we proceed slowly, we first precise what we call language, and then show how to handle commitment of a valid signature, and a commitment of a commitment. We show that those two methods can in fact be reduced to checking if we have a commitment of 1 in a specific group. A direct result is that we can now iterate (and handle commitment of commitment of commitment), but more importantly this also gives a good intuition on how to proceed to further expand the basic languages. And we show that we can handle any language composed of word (composed of elements $\mathcal{Y}_i$ committed in $\mathbb{G}$, in $\mathbf{c}_i$, for $i \in [\![1, m]\!]$ and $\mathcal{Z}_i$ committed in $\mathbb{G}_T$, in $\mathbf{C}_i$, for $i \in [\![m+1, n]\!]$) satisfying extended linear pairing equations:

$$\left(\prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i})\right) \cdot \left(\prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}}\right) = \mathcal{B}_k, \text{ for } k \in [\![1, t]\!].$$

where $\mathcal{A}_{k,i} \in \mathbb{G}$, $\mathcal{B}_k \in \mathbb{G}_T$, and $\mathfrak{z}_{k,i} \in \mathbb{Z}_p$, as well as $A_k \subseteq [\![1, m]\!]$ and $B_k \subseteq [\![m+1, n]\!]$ are public. Intuitively our method tries to transform any language into the verification of a basic one (most of the time a linear tuple).

We can see that our methodology can manage languages where words are simultaneously in $\mathbb{G}$ and $\mathbb{G}_T$, which was not possible with Groth-Sahai. In fact, we can even show, that in some cases if we only have word in $\mathbb{G}$ (for example a Diffie-Hellman tuple (g, g^a, g^b, g^{ab})), then we can prove the word belongs to the language without using any pairing which is a huge improvement, or in case of asymmetric instantiation we may rely only on XDH and not SXDH (needed by Groth-Sahai), which allows type-II curves.

In practical instantiations we often have languages with linear pairing equations like $e(\underline{\mathcal{Y}}, \mathcal{A}) = \mathcal{B}$, our methodology would require 3 elements for the commitment, and 2 for the projection key (and so the implicit proof), while Groth-Sahai methodology would still require 3 elements for the commitment but also 3 for the proofs, therefore we are more efficient. In addition, we need only one state in the CRS (no need to juggle between the perfectly binding, hiding instantiation of the commitment key).

In the asymmetric setting, we can manage a little more than linear pairing product equations, in the sense of Groth-Sahai, as we can handle simultaneous equations of the form:

$$\left(\prod_{i=1}^{m} e(\mathcal{X}_i, \mathcal{B}_i)\right) \cdot \left(\prod_{j=1}^{n} e(\mathcal{A}_j, \mathcal{Y}_j)\right) \cdot \left(\prod_{k=1}^{o} \mathcal{Z}_k^{\mathfrak{z}_k}\right) = g_T,$$

where $\mathcal{A}_j, \mathcal{B}_i, g_T$ are public values, in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively, and $\mathcal{X}_i, \mathcal{Y}_j, \mathcal{Z}_k$ are the committed private values, in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively.

With this new powerful tool, we can now apprehend several existing problems, and improves current solutions in Chapter 6, page 102:

Symmetric Pairing	$\mathbb{G}$
Groth-Sahai based	$9\,k + 24$
with SPHF	$8\,k + 12$

Asymmetric Pairing	$\mathbb{G}_1$	$\mathbb{G}_2$
Groth-Sahai based	$6\,k + 9$	$6\,k + 7$
with SPHF	$5\,k + 6$	1

Table 1.2: Number of group elements required for a ciphertext-signature pair depending on the chosen instantiation

Round-Optimal Blind Signature

To show the efficiency of the method, and the ease of application, we adapt the two Blind Signature schemes proposed in [BFPV11]. Our approach fits perfectly and decreases significantly the communicational complexity of the schemes (it is divided by more than three in one construction). Moreover the asymmetric version of the scheme only relies on a weakened security assumptions: the XDH assumption instead of the SXDH assumption and permits to use more bilinear group settings (namely, Type-II and Type-III bilinear groups [GPS08] instead of only Type-III bilinear groups for the construction presented in [BFPV11]).

One can see a nice improvement in the constant part in the DLin instantiation when we use our Smooth Projective Hash Protocols, one can also see a huge improvement in the asymmetric setting where we remove nearly all elements in $\mathbb{G}_2$ underlying a huge redundancy with the Groth-Sahai approach.

Oblivious Signature-Based Envelope

Another contribution is to clarify and increase the security requirements of an OSBE scheme in Section 6.1, page 102. The main improvement residing in some protection for both the sender and the receiver against the Certification Authority. The OSBE notion echoes directly to the idea of SPHF if we consider the language $\mathcal{L}$ defined by encryption of valid signatures, which is hard to distinguish under the security of the encryption schemes. We show how to build, from a SPHF on this language, an OSBE scheme in the standard model with a CRS, this is described quickly in the Figure 1.4, page 11. And we prove the security of our construction in regards of the security of the commitment (the ciphertext), the signature and the SPHF scheme. We then show how to build a simple and efficient OSBE scheme relying on a classical assumption, DLin. To build this scheme, we use the SPHF in our new way, avoiding the need of costly Groth-Sahai proofs when an interaction is inherently needed in the primitive.

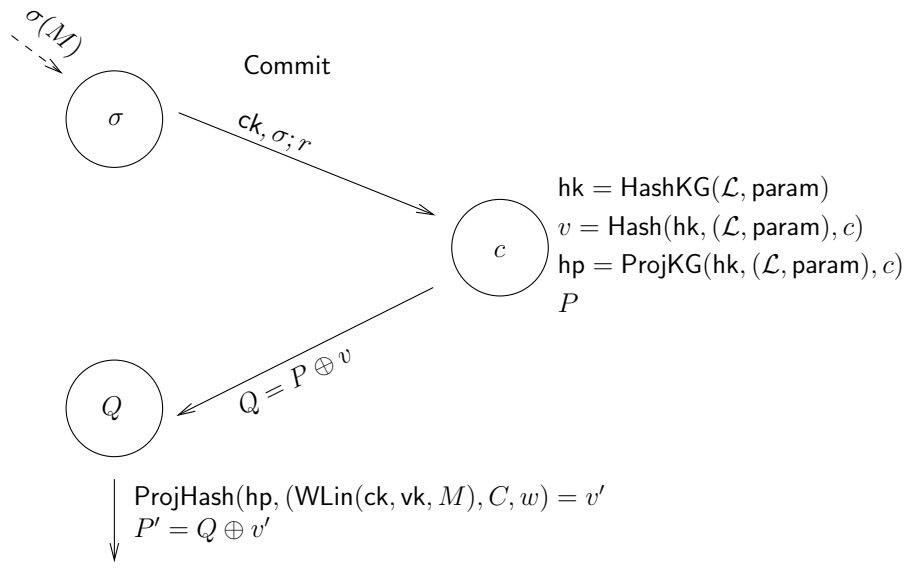


Figure 1.4: The sender on the right wants to send a message P to U on the left, if and only if U owns a signature $\sigma(M)$ valid under vk.

We can see that our method does not add any other interaction, and so supplement smoothly Groth-Sahai proofs.

While OSBE may not be the most used protocols in modern cryptography, there are in fact the cornerstone of several constructions, and so this results leads us to the following LAKE.

Language Authenticated Key Exchange

We then propose in Section 6.3, page 111 a new primitive that encompasses the previous notion of PAKE and Secret Handshakes. It is closely related to CAKE and we call it LAKE, for *Language-Authenticated Key-Exchange*, since parties establish a common key if and only if they hold credentials that belong to specific (and possibly independent) languages that the players do not need to agree on prior to the protocol execution. The definition of the primitive is more practice-oriented than the definition of CAKE from [CCGS10] but the two notions are very similar[6].

PAKE are a special case of LAKE, where each user possess a password pw_* and expects the other participant to possess a word in the language $\mathcal{L} = \{pw_i\}$. And Secret Handshakes says that each user possesses a signature on their identity by a specific authority and expect the user to possess a valid signature by another authority (possibly the same authority). We add some granularity on the requirements as the authority can be the same or not, the user identities can be public or not, ...

In particular, the new primitive enables privacy-preserving authentication and key exchange protocols by allowing two members of the same group to secretly and privately authenticate to each other without revealing this group beforehand.

In order to define the security of this primitive, we use the UC framework and an appropriate definition for languages that permits to dissociate the public part of the policy, the private common information the users want to check and the (possibly independent) secret values each user owns that assess the membership to the languages. We provide an ideal functionality for LAKE and give efficient realizations of the new primitive (for a large family of languages) secure under classical mild assumptions, in the standard model (with a common reference string - CRS), with static corruptions.

With this approach, we obtain the most efficient PAKE scheme secure in the standard model with CRS (improving the schemes from [ACP09,CCGS10] in terms of computational workload, communication complexity and round complexity).

We also significantly improve the efficiency of several CAKE protocols [CCGS10] and we enlarge the set of languages for which we can construct practical schemes. Notably, we obtain a very practical realization of Secret Handshakes that provides very strong security properties (such as forward-secrecy even in the case of corruption of the group authority), and also a very efficient instantiation of PAKE protocol resistant to server corruption.

1.4 Extra-Tools

While following our original motivations, we often encounter side ideas which are not necessarily primordial to solve the problem but which may lead to neat improvement in the efficiency of several scheme.

We present in Section 2.6, page 35 some of the tools, we had to build and prove.

Waters Function / Signature

With our modular approach, we have a constant use of Waters signature and its function. We have many side results around this theme. For example, we proposed and prove the security of an asymmetric version of the Waters Signature in [BFPV11], and recall it in Section 2.6.3, page 41. Waters is the main signature in the standard model, and as many protocols try to be efficient and for that are instantiated over asymmetric bilinear groups, we thought presenting an asymmetric version of this signature might allow new results.

Another result related to Waters function is about its programmability over a non-binary alphabet. While in [HK08], Hofheinz and Kiltz have shown the $(2,1)$ and $(1,q)$-programmability over a binary alphabet, we decided to further develop their result when some kind of homomorphic forgeability may be needed like in wireless (ad hoc) sensor network. We easily show in Section 2.6.4, page 42 that over a non-binary alphabet Waters is no longer $(2,1)$-programmable. We decided to further develop their probabilistic approach, and for that we rely on the Local Central Limit Theorem [DM95], and we show it remains $(1,q)$-programmable, as long as the message bloc size t does not grow to fast [7](This improves the previous result from [Nac05]). The Local Central Limit Theorem basically provides an approximation of $Pr(\sum a_i = k)$ for independent and identically distributed random variables a_i (It is a version of the

[6] Actually we believe that any interesting AKE primitive can be formalized in either way.

[7] i.e. $2^t = \mathcal{O}(\log \mathfrak{K})$.

Central Limit Theorem in which the conclusion is strengthened from convergence of the law to locally uniform pointwise convergence of the densities).

With that, we can avoid sequencing a message bit-per-bit on the Waters hash, and so instead of computing $\mathcal{F}(M) = u_0 \prod u_i^{m_i}$, one can now consider blocks of bits M_i and compute a Waters like hash: $u_0 \prod u_i^{M_i}$ or even $u_0 u_1^M$ if the message space is moderately small. Another thing, from the same result is the application to wireless sensor network, where we prove that signing several messages M_i in a Waters fashion so with $\sigma = (\mathsf{sk}(\prod \mathcal{F}(M_i))^s, g^s)$ is still unforgeable under CDH is the number of messages is reasonable with respect to the security parameter.

Batch Groth-Sahai

The verification of a Groth-Sahai NIZK is relatively slow (lots of pairing computations). We consider in [BFI+10] a method to batch the verification of several proofs simultaneously, to achieve the verification faster while not undermining the soundness. We consider the work of Ferrarra *et al.* in [FGHP09], and manage to find a way to answer correctly to the question "Are all those proofs of knowledge valid?". If the answer is true then except with a negligible probability all the statements are valid, else with a divide and conquer approach we can deduce which statements are false.

We detail in Section 2.6.1, page 35 our approach around the DLin version of Groth-Sahai proofs, and then apply in Section 2.6.2, page 38 our results to existing protocols.

	Naive computation	Batch computation
SXDH		
Pairing-product equation	$5\,m + 3\,n + 16$	$m + 2\,n + 8$
Multi-scalar multiplication equation in $\mathbb{G}_1$	$8\,m + 2\,n + 14$	$\min(2\,n + 9, 2\,m + n + 7)$
Quadratic equation	$8\,m + 8\,n + 12$	$2\min(m, n) + 8$
DLin		
Pairing-product equation	$12\,n + 27$	$3\,n + 6$
Multi-scalar multiplication equation	$9\,n + 12\,m + 27$	$3\,n + 3\,m + 6$
Quadratic equation	$18\,n + 24$	$3\,n + 6$

Table 1.3: Number of pairings per verification, where n and m stand for the number of variables.

A surprising result in that our approach leads to significant improvement even on the verification of a single signature, for example on Groth Signature [Gro07] we go from 68 to 11 pairings involved in the verification of one single signature, the more impressive result is on the P-signatures where in [BCKL08], the authors evaluated that the verification of the proof in the DLin instantiation requires the computation of 126 pairings. With our result, we prove it can be reduced to only 12 for a single signature, and even $3\,n + 9$ for n signatures.

Multi Cramer-Shoup

We also consider the Cramer-Shoup encryption [CS98] in Section 2.6.5, page 46. For our UC approach, we need several modification to the original scheme. First we show that if we use a global hash value for several ciphertexts at once, then the encryption remains CCA, this allows to encrypt vectors at once, with only one hash computation; while it may seem quite annex as a result, this will reveal to be crucial when one wants to use Cramer-Shoup for our methodology on languages without a massive growth in the number of projection keys.

And then following the approach of Lindell in [Lin11], we consider Cramer-Shoup ciphertext composed in fact of two ciphertexts $\mathcal{C}$,and $\mathcal{C}'$, we show that the encryption provides indistinguishability against partial-decryption chosen-ciphertext attacks when considering such ciphertext $\mathcal{C}, \mathcal{C}'$ the adversary is allowed to query the decryption of $\mathcal{C}$ and has to provide the decryption of $\mathcal{C}, \mathcal{C}'$. $\mathcal{C}$ is a valid Cramer-Shoup ciphertext, while $\mathcal{C}'$ is not as the hash value used is the one corresponding to $\mathcal{C}$. Those results are further develop in Section 2.6.5, page 46, over both the regular Cramer-Shoup and the linear one from [Sha07, CKP07]

From this last construction, we manage to construct an interactive commitment which remains efficient while being both extractable and equivocable in Section 2.6.6, page 52. We commit into 3 steps, first we compute $\mathcal{C}, \mathcal{C}'$ as before, where $\mathcal{C}$ encrypts M and $\mathcal{C}'$ encrypts $1_{\mathbb{G}}$, and $\mathcal{C}''$ as a Pedersen commitment to $\mathcal{C}'$, and send $\mathcal{C}, \mathcal{C}''$. Then the receiver sends a challenge ϵ. And we answer with $\mathcal{C}'$ and a proof

of correct opening of the Pedersen. To decommit, one simply combines the random used in both $\mathcal{C}$ and $\mathcal{C}'$ with ϵ to show that $\mathcal{C}\mathcal{C}'^{\epsilon}$ is indeed a commitment to M.

$\star$

Chapter 2

Technical Introduction

Contents

In this chapter we will give an overview of the different notions we are going to use throughout this thesis. We first start by giving a basic guideline for our notations, we follow when possible the standard conventions, however in some cases they were not fully compatible with each other. We then remind some classical cryptographic notions, and give some instantiations.

We end this chapter by proposing improvements to those classical instantiations, which are going to be needed afterwards.

2.1 Notations

Throughout the thesis, we try to have some uniformity in our notations, here are the main conventions we are going to follow.

- Groups will be denoted by bold letters according to the standard convention. When we don't consider any pairing, or we are in the symmetric case, we will use the group $\mathbb{G}$, in case of asymmetric pairings the groups will be $\mathbb{G}_1$ and $\mathbb{G}_2$. The target group will always be $\mathbb{G}_T$.

- Group generators will be noted by lowercase letters ($g \in \mathbb{G}$), possibly with indices denoting the groups ($g_1 \in \mathbb{G}_1$) or if they are part of sequences $(u_i)_{i\in[\![0,n]\!]}$.
- Generic public group elements will be noted in uppercase letters $\mathcal{A}, \mathcal{B} \in \mathbb{G}$.
- Columns vectors will be noted by $\vec{u}$, line vectors by $\mathbf{v}$, and matrices by $\vec{\mathbf{w}}$. This last one is composed of group elements w_{ij}.
- a, b will be used for generic scalars. r, s, t will often be used as the generic randomness used in encryption and signature schemes. In the other cases, a scalar will be in lowercase greek letters. A random draw will be noted $\rho \xleftarrow{\$} \mathbb{Z}_p$.
- In algorithms the random coins used will be postponed after the inputs and a semicolon ;. For example, $\mathsf{Encrypt}(\mathsf{ek}, M; r)$, will be an encryption algorithm on a message M, under an encryption key ek, with randomness r.
- There will possibly be some minor exceptions:
  - e will be the non-degenerated bilinear form of pairing.
  - $\mathfrak{K}$ will be the security parameter, $1^{\mathfrak{K}}$ will be a string composed of $\mathfrak{K}$ concatenated 1.
  - m or M will often design a message. The message is viewed as its scalar representation so in $\mathbb{Z}_p$.
  - p will be a prime number, generally the order of the groups. (In that case p has often $\mathfrak{K}$ bits.)
  - A generic proof of knowledge will be noted π.
  - A signature will be noted $\sigma()$, a ciphertext or a commitment by $\mathcal{C}()$, we will represent the Waters function by $\mathcal{F}()$.
- If required the group law will be noted " $\cdot$" but may be forgotten, in some cases, to ease the reading; " $\odot$ " will be for the entry-wise product in matrices.

$$(a_{ij})_{\substack{i\in[\![1,k]\!]\\ j\in[\![1,n]\!]}} \odot (b_{ij})_{\substack{i\in[\![1,k]\!]\\ j\in[\![1,n]\!]}} = (a_{ij}b_{ij})_{\substack{i\in[\![1,k]\!]\\ j\in[\![1,n]\!]}}$$

- We will use $\langle\cdot,\cdot\rangle$ for bilinear products between vectors of either scalars or group elements. For $\vec{a}, \vec{b} \in \mathbb{Z}_p^n$ and $\vec{\mathcal{A}}, \vec{\mathcal{B}} \in \mathbb{G}^n$, we define:

$$\langle \vec{a}, \vec{b} \rangle := \textstyle\sum_{i=1}^n a_i \cdot b_i \qquad \langle \vec{a}, \vec{\mathcal{B}} \rangle := \prod_{i=1}^n \mathcal{B}_i^{a_i} \qquad \langle \vec{\mathcal{A}}, \vec{\mathcal{B}} \rangle := \prod_{i=1}^n e(\mathcal{A}_i, \mathcal{B}_i).$$

- We will note " $\bullet$ " the distributed pairing: $\mathbb{G}^{n\times 2} \times \mathbb{G}^{n\times 2} \to \mathbb{G}_T^{2\times 2}$:

$$\vec{\mathbf{c}} \bullet \vec{\mathbf{d}} := \begin{pmatrix} \prod_{i=1}^n e(c_{i,1}, d_{i,1}) & \prod_{i=1}^n e(c_{i,1}, d_{i,2}) \\ \prod_{i=1}^n e(c_{i,2}, d_{i,1}) & \prod_{i=1}^n e(c_{i,2}, d_{i,2}) \end{pmatrix}.$$

- And " $\overset{s}{\bullet}$ " its symmetric variant: $\mathbb{G}^{n\times 3} \times \mathbb{G}^{n\times 3} \to \mathbb{G}_T^{3\times 3}$:

$$\begin{aligned} \vec{\mathbf{c}} \overset{s}{\bullet} \vec{\mathbf{d}} &:= \begin{pmatrix} \prod_{i=1}^n e(c_{i,1}, d_{i,1}) & \prod_{i=1}^n e(c_{i,1}, d_{i,2})^{\frac{1}{2}} e(c_{i,2}, d_{i,1})^{\frac{1}{2}} & \prod_{i=1}^n e(c_{i,1}, d_{i,3})^{\frac{1}{2}} e(c_{i,3}, d_{i,1})^{\frac{1}{2}} \\ \prod_{i=1}^n e(c_{i,2}, d_{i,1})^{\frac{1}{2}} e(c_{i,1}, d_{i,2})^{\frac{1}{2}} & \prod_{i=1}^n e(c_{i,2}, d_{i,2}) & \prod_{i=1}^n e(c_{i,2}, d_{i,3})^{\frac{1}{2}} e(c_{i,3}, d_{i,2})^{\frac{1}{2}} \\ \prod_{i=1}^n e(c_{i,3}, d_{i,1})^{\frac{1}{2}} e(c_{i,1}, d_{i,3})^{\frac{1}{2}} & \prod_{i=1}^n e(c_{i,3}, d_{i,2})^{\frac{1}{2}} e(c_{i,2}, d_{i,3})^{\frac{1}{2}} & \prod_{i=1}^n e(c_{i,3}, d_{i,3}) \end{pmatrix} \\ &:= \Big(\prod_{i=1}^n e(c_{i,j}, d_{i,k})^{\frac{1}{2}} e(c_{i,k}, d_{i,j})^{\frac{1}{2}}\Big)_{\substack{j\in[\![1,3]\!]\\ k\in[\![1,3]\!]}}. \end{aligned}$$

- Concatenation will be noted "||".

2.2 Definitions

2.2.1 Generalities

Hard problem
⌜ A function $f : \mathbb{N} \to \mathcal{R}$ is said to be negligible, if $\forall c \in \mathbb{N}, \exists k_0 \in \mathbb{N}, \forall k \geq k_0 : |f(k)| < k^{-c}$. A problem is said to be hard, if there exists no polynomial time algorithm solving it with non-negligible probability. ⌟

Cyclic Group
⌜ A cyclic group is a tuple $(p, \mathbb{G}, g)$ where $\mathbb{G}$ is a group entirely generated by g where $g^p = 1_{\mathbb{G}}$. (The neutral element of $\mathbb{G}$) ⌟

Bilinear Groups
⌜ A bilinear group is a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ where $\mathbb{G}_1, \mathbb{G}_2$ et $\mathbb{G}_T$ are cyclic groups of prime order p, generated respectively by g_1,g_2 and $e(g_1, g_2)$, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerated bilinear form, i.e. :

$$\forall X \in \mathbb{G}_1, \forall Y \in \mathbb{G}_2, \forall \lambda, \mu \in \mathbb{Z}_p : e(X^\lambda, Y^\mu) = e(X, Y)^{\lambda\mu}$$

and $e(g_1, g_2)$ does indeed generate the prime order group $\mathbb{G}_T$. In the following we will suppose there exists a polynomial time algorithm GrpGen which takes $1^{\mathfrak{K}}$ as input, and which outputs such bilinear groups. In this case p is a prime order of $\mathfrak{K}$ bits.

Such groups are commonly instantiated on elliptic curves on which such pairings can be defined as bilinear forms. Galbraith *et al.* [GPS08] have split such instantiations in three main types:

- Type I, where $\mathbb{G}_1 = \mathbb{G}_2$, and $g_1 = g_2$, those groups are said to be symmetric and can be simplified in $(p, \mathbb{G}, \mathbb{G}_T, e, g)$. This first case often leads to problems based on the DLin hypothesis,
- Type II, if there exists a computationally efficient homomorphism from $\mathbb{G}_2$ in $\mathbb{G}_1$, but none from $\mathbb{G}_1$ to $\mathbb{G}_2$. This case often leads to problems based on the XDH hypothesis,
- Type III, if such efficient homomorphism does not exist in either way. This last case often leads to problems based on the SXDH hypothesis.

⌟

2.2.2 Security Hypotheses

We are now going to describe common security hypotheses used to prove the security of our protocols, all except the traceable signatures (section 3, page 55) only rely on those standard hypotheses. All the first except the discrete logarithm are decisional hypotheses, some computational versions exist also. We will remind the non-standard hypotheses only when needed in Section 3, page 55.

Discrete Logarithm (DL)
⌜ The Discrete Logarithm hypothesis says that given $(p, \mathbb{G}, g)$, and an extra element $h \in \mathbb{G}$ it is hard to find $\mu \in \mathbb{Z}_p$ such that $h = g^\mu$. ⌟

Decisional Linear (DLin [**BBS04**])
⌜ The Decisional Linear hypothesis says that in a multiplicative group $(p, \mathbb{G}, g)$ when we are given $(g^\lambda, g^\mu, g^{\alpha\lambda}, g^{\beta\mu}, g^\psi)$ for unknown random $\alpha, \beta, \lambda, \mu \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $\psi = \alpha + \beta$. ⌟

Decisional Diffie Hellman (DDH [**Bon98**])
⌜ The Decisional Diffie-Hellman hypothesis states that in a multiplicative group $(p, \mathbb{G}, g)$, given (g^μ, g^ν, g^ψ) for unknown $\mu, \nu \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $\psi = \mu\nu$. ⌟

External Diffie Hellman in $\mathbb{G}_1$ (XDH [**BBS04**])
⌜ This variant of the previous hypothesis states that in a type II bilinear group, given $(g_1^\mu, g_1^\nu, g_1^\psi)$ for unknown $\mu, \nu \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $\psi = \mu\nu$. (In other words DDH is hard in $\mathbb{G}_1$.) A variant can say that DDH is hard in $\mathbb{G}_2$. ⌟

Symmetric External Diffie Hellman (SXDH [**ACHdM05**])
⌜ This last variant, used mostly in type III bilinear groups, states that DDH is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. ⌟

We also describe two computational hypotheses related to the DDH:

Computational Diffie Hellman (CDH [DH76])
⌜ The Computational Diffie-Hellman hypothesis states that in a multiplicative group $(p, \mathbb{G}, g)$, given (g^μ, g^ν) for unknown $\mu, \nu \xleftarrow{\$} \mathbb{Z}_p$, it is hard compute $g^{\mu\nu}$. ⌟

Advanced Computational Diffie-Hellman problem (CDH^+ [BFPV11]):
⌜ Let us be given two (multiplicative) groups $(\mathbb{G}_1, \mathbb{G}_2)$ of prime order p with (g_1, g_2) as respective generators. The CDH^+ assumption states that given $(g_1, g_2, g_1^\mu, g_2^\mu, g_1^\nu)$, for random $\mu, \nu \in \mathbb{Z}_p$, it is hard to compute $g_1^{\mu\nu}$. ⌟

2.2.3 Universal Composability

In chapter 6, page 102, our main goal will be to provide protocols security in the universal composability framework. This framework was introduced in [Can01].

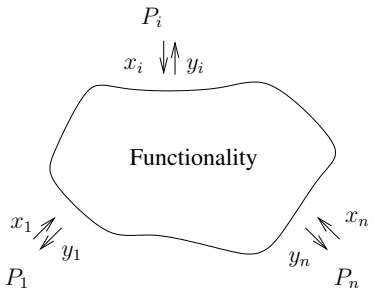
This framework can be quite overwhelming. The aim of the following is just to give a brief overview to have some common conventions.

In the context of multi-party computation, one wants several users P_i with inputs x_i to be able to compute a specific function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ without learning anything except y_i. This approach was seen for example in Yao's Millionaires' problem [Yao82], where two millionaires want to know who is richer without revealing their respective wealth. So here, x_i is the wealth of the millionaire i, and f simply returns which one is richer (in this specific case $y_1 = y_i = y_n$).

Instead of following the classical approach which aims to list exhaustively all the expected properties, Canetti did something else and tried to define how a protocol should ideally work.

For that, he divided the world into two spaces, the real world, where the protocol is run with some possible attack, and the ideal world where everything would go smoothly. For a good protocol, it should be impossible to distinguish the real world from the ideal one.

In the ideal world there is an incorruptible entity named the ideal functionality, to which players can send their inputs privately, and then receive the corresponding output without any kind of communication between the players. This way the functionality can be set to be correct, without revealing anything except what is expected.



A protocol, in the real world with an adversary, should create an execution similar to the one obtained by the ideal functionality. This means that the communication between the players should not give more information than the functionality description, and its output. In this case the protocol runs not really against the adversary but against the environment who picks the inputs given to the players, and obtains the outputs. After the interaction the environment should output a bit saying whether he is in the real world.

The main constraint is that the adversary is now free to interact with the environment whenever he wants which prevents the simulator from rewinding when needed. The adversary has access to the communication between the players but not their inputs/outputs, while the environment has only access to the inputs/outputs.

To prove that a protocol realizes the ideal functionality, we consider an environment $\mathcal{Z}$ which can choose inputs given to all the users and whose goal is to distinguish in which case he receives outputs from the real world execution with an adversary $\mathcal{A}$, and in which case they come from an ideal execution with an ideal adversary $\mathcal{S}$ who interacts solely with the functionality. Such protocol realizes the functionality if for all polynomial adversary $\mathcal{A}$, there exists a polynomial simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish the real world from the ideal one with a non-negligible probability.

Since players are not individually authenticated, but just afterward if the credentials are mutually consistent with the two players' languages, the adversary will be allowed to interact on behalf of any player from the beginning of the protocol, either with the credentials provided by the environment (static

corruption) or without (impersonation attempt). As with the Split Functionality [BCL+05], according to whom sends the first flow for a player, either the player itself or the adversary, we know whether this is an honest player or a dishonest player (corrupted or impersonation attempt, but anyway controlled by the adversary). In the adaptive corruption setting, the adversary could get complete access to the private credentials and the internal memory of an honest player, and then get control of it, at any time. But we will restrict to the static corruption setting in this thesis. It is enough to deal with most of the concrete requirements: related credentials, arbitrary compositions, and forward-secrecy.

More precisely, we will work in the UC framework with joint state proposed by Canetti and Rabin [CR03] (for the CRS). Informally, this allows different protocols to have some common states while preserving their security. Basically for a given session identifier sid we also define sub-session identifier ssid, and so we have a functionality, possibly generated on the fly, for each ssid.

2.2.4 Standard Cryptographic Primitives

Encryption

An encryption scheme $\mathcal{E}$ is described through four algorithms $(\mathsf{Setup}_{\mathcal{E}}, \mathsf{KeyGen}_{\mathcal{E}}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}_{\mathcal{E}}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$ outputs a pair of keys, a (public) encryption key pk and a (private) decryption key dk;
- $\mathsf{Encrypt}(\mathsf{pk}, M; \rho)$ outputs a ciphertext $\mathbf{c} = \mathcal{C}(M)$, on the message M, under the encryption key pk, with the randomness ρ;
- $\mathsf{Decrypt}(\mathsf{dk}, \mathbf{c})$ outputs the plaintext M, encrypted in the ciphertext $\mathbf{c}$ or $\perp$.

Such encryption scheme is required to have the following security properties:

- *Correctness*: For every pair of keys $(\mathsf{ek}, \mathsf{dk})$ generated by $\mathsf{KeyGen}_{\mathcal{E}}$, every messages M, and every random ρ, we should have $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{Encrypt}(\mathsf{ek}, M; \rho)) = M$.
- *Indistinguishability under Chosen Plaintext Attack [GM84]*: This notion ($\mathsf{IND}-\mathsf{CPA}$), formalized by the adjacent game, states that an adversary shouldn't be able to efficiently guess which message has been encrypted even if he chooses the two original plaintexts.
  The advantages are:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = |\Pr[\mathsf{Exp}^{\mathsf{ind}-1}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}) = 1]|$$
$$\mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathfrak{K}, t) = \max_{\mathcal{A} \le t} \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}).$$

$\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{Setup}_{\mathcal{E}}(1^{\mathfrak{K}})$
2. $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$
3. $(M_0, M_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk})$
4. $c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, M_b)$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : c^*)$
6. `RETURN` b'

One might want to increase the requirements on the security of an encryption, in this case the $\mathsf{IND}-\mathsf{CPA}$ notion can be strengthened into Indistinguishability under Adaptive Chosen Ciphertext Attack $\mathsf{IND}-\mathsf{CCA2}$ (The non-adaptive notion was introduced in [NY90], while the adaptive one was introduced a year later in [RS92]):

- $\mathsf{IND}-\mathsf{CCA2}$: This notion states that an adversary should not be able to efficiently guess which message has been encrypted even if he chooses the two original plaintexts, and can ask several decryption of ciphertexts as long as they are not the challenge one.

$\mathsf{Exp}^{\mathsf{ind}-\mathsf{cca}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{ESetup}(1^{\mathfrak{K}})$
2. $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$
3. $(M_0, M_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{pk}, \mathsf{ODecrypt}(\cdot))$
4. $c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, M_b)$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : c^*, \mathsf{ODecrypt}(\cdot))$
6. `IF` $(c^*) \in \mathcal{CT}$ `RETURN` 0
7. `ELSE RETURN` b'

  – Where the ODecrypt oracle outputs the decryption of c under the challenge decryption key dk. The input queries (c) are added to the list $\mathcal{CT}$ of decrypted ciphertexts.

One may want to extend the notion of encryption to a labelled encryption, where the message M is encrypted but with some extra public information ℓ. This label can be useful to include session information for example.

Labelled Encryption Scheme

A labelled public-key encryption scheme is defined by four algorithms:

- $\mathsf{Setup}_{\mathcal{E}}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$ generates a pair of keys, the encryption key pk and the decryption key dk;
- $\mathsf{Encrypt}(\ell, \mathsf{pk}, M; \rho)$ produces a ciphertext $\mathbf{c}$ on the input message $M \in \mathcal{M}$ under the label ℓ and encryption key pk, using the random coins ρ;
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, \mathbf{c})$ outputs the plaintext M encrypted in $\mathbf{c}$ under the label ℓ, or $\perp$.

Commitment

Commitments allow a user to commit to a value without revealing it, but without the possibility to later change his mind. It is composed of three algorithms:

- $\mathsf{Setup}(1^{\mathfrak{K}})$ generates the system parameters, according to the security parameter $\mathfrak{K}$;
- $\mathsf{Commit}(m; r)$ produces a commitment $\mathbf{c}$ on the input message $m \in \mathcal{M}$ using the random coins $r \xleftarrow{\$} \mathcal{R}$;
- $\mathsf{Decommit}(\mathbf{c}, m; w)$ opens the commitment $\mathbf{c}$ and reveals the message m, together with a witness w that proves the correct opening.

Such a commitment scheme should be both *hiding*, which says that the commit phase does not leak any information about m, and *binding*, which says that the decommit phase should not be able to open to two different messages. Additional features are also sometimes required, such as non-malleability, extractability, and/or equivocability. We may also include a label ℓ, which is an additional public information that has to be the same in both the commit and the decommit phases.

All these properties are satisfied by a scheme that realizes the ideal functionality, in the universal composability framework, for multiple commitments presented on Figure 2.1, page 21: no information is leaked from m to any player during the commit phase, so the concrete flows should not leak any information about the actual committed message m, hence the *hiding property*, and even the *non-malleability*; the $\mathsf{Decommit}$ message automatically reveals the initially committed value, so it is not possible to change the output, hence the *binding property*. Now, in the proof that a concrete scheme emulates the ideal functionality, in order to be able to send the appropriate commit query to $\mathcal{F}_{\text{LMCOM}}$ from the concrete flows, sent by the adversary on behalf of a corrupted user P_i, the simulator must be able to extract the actual committed message m, hence the *extractability property*. Furthermore, from a commit-query sent by an honest user, the simulator should be able to generate appropriate concrete flows, without any information about the actual message the environment sent, on behalf of P_i, to the functionality $\mathcal{F}_{\text{LMCOM}}$. But when the adversary $\mathcal{A}$ asks for the $\mathsf{Decommit}$, the simulator learns the message m the commitment should be open to, hence the *equivocability property*.

One can also define a commitment scheme, with a slightly different approach like in the case of Mercurial Commitments [CHL+05b]. Basically we have some commitment key ck generated by one of the setup algorithm (and none except the simulator should know which algorithm was used), in one case we have a equivocable scheme, while in the other we have an extractable scheme. If a user decommit to a plaintext, any observer can be assured that the commitment can not be extracted to any other value, but still the commitment may not be extractable:

Mercurial Commitment Scheme

$\ulcorner \mathcal{C} = (\mathsf{Setup}_{\mathcal{C}}, \mathsf{WISetup}, \mathsf{ExSetup}, \mathsf{Commit}, \mathsf{Extract})$:

- $\mathsf{Setup}_{\mathcal{C}}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and more specifically the commitment key ck by running one of the following;
  - $\mathsf{WISetup}(1^{\mathfrak{K}})$, outputs a perfectly hiding commitment key ck;

The functionality $\mathcal{F}_{\text{LMCOM}}$ is parametrized by a security parameter $\mathfrak{K}$. It interacts with an adversary $\mathcal{S}$ and a set of parties $P_1,\ldots,P_n$ via the following queries:

- Commit phase: Upon receiving a message $(\mathsf{Commit} : \mathsf{sid}, \mathsf{ssid}, P_i, P_j, \ell, m)$ from P_i where $m \in \{0,1\}^{\mathsf{polylog}(\mathfrak{K})}$, record the tuple $(\mathsf{ssid}, P_i, P_j; \ell, m)$ and send the message $(\mathsf{receipt}; \mathsf{sid}, \mathsf{ssid}, P_i, P_j; \ell)$ to P_j and $\mathcal{S}$. Ignore any future commit query with the same ssid from P_i to P_j;
- Decommit phase: Upon receiving a message $(\mathsf{Decommit} : \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ from P_i: If a tuple $(\mathsf{ssid}, P_i, P_j; \ell, m)$ was previously recorded, then send the message $(\mathsf{reveal}; \mathsf{sid}, \mathsf{ssid}, P_i, P_j; m)$ to P_j and $\mathcal{S}$. Otherwise, ignore.

Figure 2.1: Ideal Functionality $\mathcal{F}_{\text{LMCOM}}$

  - $\mathsf{ExSetup}(1^{\mathfrak{K}})$, outputs an extractable commitment pair of keys $(\mathsf{ek}, \mathsf{ck})$, where the extraction key ek will be kept secret;

- $\mathsf{Commit}(\mathsf{ck}, M; \rho)$, outputs a commitment $\mathbf{c} = \mathcal{C}(M)$ of a message M under a commitment key ck and a random $\rho \in \mathcal{R}$;
- $\mathsf{Extract}(\mathsf{ek}, \mathbf{c})$, if $(\mathsf{ek}, \mathsf{ck})$ were created through ExSetup then it outputs M from $\mathbf{c}$ using the extraction key ek.
- $\mathsf{Decommit}(\mathbf{c}, M; \rho)$ decommits $\mathbf{c}$ under the randomness ρ; it outputs a plaintext M such that $c = \mathsf{Commit}(\mathsf{ck}, M; \rho)$.

⌟

In this case we expect the following properties:

- If ExSetup was used to build $(\mathsf{ek}, \mathsf{ck})$, for every $(\mathsf{ck}, \mathbf{c})$, there should exist a unique M, and a unique ρ such that $\mathbf{c} = \mathsf{Commit}(\mathsf{ck}, M; \rho)$ (or $\mathsf{Decommit}(\mathbf{c}, M; \rho) = M$), moreover $\mathsf{Extract}(\mathsf{ek}, \mathbf{c})$ should output M, in this case the scheme is perfectly binding.
- The keys ck^* output by WISetup are indistinguishable from those output by ExSetup and leads to perfectly hiding commitments $\forall \mathbf{c}, M, \exists \rho, \mathbf{c} = \mathsf{Commit}(\mathsf{ck}^*, M; \rho)$. So, after a generation with ExSetup, the scheme is computationally hiding.

It should be noted that in the original definition of mercurial commitments, the Decommit algorithm is called *Tease*, because it shows the possible opening value without telling if the open algorithm exists. The main difference with the previous UC commitment, is that the mercurial commitment is not simultaneously equivocable and extractable.

One may also want to add a randomization property on the commitment:

- The commitment is randomizable, if for every commitment $\mathbf{c}$, randomness ρ', we are able to compute $\mathsf{Random}_{\mathcal{C}}(\mathsf{ck}, \mathbf{c}; \rho')$. If ρ' is uniformly chosen then $\mathbf{c}'$ is distributed like a fresh commitment $\mathsf{Commit}(\mathsf{ck}, M; \rho)$ when ρ is uniformly chosen.

This last commitment scheme, can be viewed as a lossy encryption [BHY09]. An encryption scheme, with two possible key generation algorithms, one leading to a regular encryption, the other leading to a ciphertext independent from the original message. Those keys are assumed to be indistinguishable. Such scheme is $\mathsf{IND}-\mathsf{CPA}$. It can easily be seen that under the lossy key, the adversary can't have an efficient strategy to guess which message has been chosen as the ciphertext is independent of the original message hence the hiding property, while in the regular setting, the decryption key of the encryption leads to the uniqueness of the plaintext hence the binding property.

Digital Signature

A digital signature scheme $\mathcal{S}$ [DH76, GMR88] allows a signer to produce a verifiable proof that he indeed produced a message. It is described through four algorithms $(\mathsf{Setup}_{\mathcal{S}}, \mathsf{KeyGen}_{\mathcal{S}}, \mathsf{Sign}, \mathsf{Verif})$:

Digital Signature Scheme
⌜ $\sigma = (\mathsf{Setup}_{\mathcal{S}}, \mathsf{KeyGen}_{\mathcal{S}}, \mathsf{Sign}, \mathsf{Verif})$:

- $\mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$ where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, for example the message space;
- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$, outputs a pair of $(\mathsf{sk}, \mathsf{vk})$, where sk is the (secret) signing key, and vk is the (public) verification key;
- $\mathsf{Sign}(\mathsf{sk}, M; \mu)$, outputs a signature $\sigma(M)$, on a message M, under the signing key sk, and some randomness μ;
- $\mathsf{Verif}(\mathsf{vk}, M, \sigma)$ checks the validity of the signature σ with respect to the message M and the verification key vk. And so outputs a bit.

⌟

In the following we will expect at least two properties for signatures:

- *Correctness*: For every pair $(\mathsf{vk}, \mathsf{sk})$ generated by $\mathsf{KeyGen}_{\mathcal{S}}$, for every message M, and for all randomness μ, we have $\mathsf{Verif}(\mathsf{vk}, M, \mathsf{Sign}(\mathsf{sk}, M; \mu)) = 1$.
- *Strong Existential Unforgeability under Chosen Message Attacks [SPMLS02].* Even after querying n valid signatures σ_i on chosen messages M_i, an adversary should not be able to output a fresh valid signature. To formalize this notion, we define a signing oracle `Sign`:
  - $\mathtt{Sign}(\mathsf{vk}, m)$: This oracle outputs a signature on m valid under the verification key vk. The resulting pair (m, σ) is added to the signed pair set $\mathcal{S}'$.

$\mathsf{Exp}^{\mathsf{st-uf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$
3. $(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk}, \mathtt{Sign}(\mathsf{vk}, \cdot))$
4. $b \leftarrow \mathsf{Verif}(\mathsf{vk}, m^*, \sigma^*)$
5. `IF` $(m^*, \sigma^*) \in \mathcal{S}'$ `RETURN` 0
6. `ELSE RETURN` b

The probability of success against this game is denoted by

$$\mathsf{Succ}^{\mathsf{st-uf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{st-uf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}) = 1], \quad \mathsf{Succ}^{\mathsf{st-uf}}_{\mathcal{S}}(\mathfrak{K}, t) = \max_{\mathcal{A} \leq t} \mathsf{Succ}^{\mathsf{st-uf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}).$$

Or *Existential Unforgeability under Chosen Message Attacks [GMR88]* ($\mathsf{EUF-CMA}$). Even after querying n valid signatures on chosen messages (M_i), an adversary should not be able to output a valid signature on a fresh message M. To formalize this notion, we define a signing oracle `Sign`:

- $\mathtt{Sign}(\mathsf{vk}, m)$: This oracle outputs a signature on m valid under the verification key vk. The requested message is added to the signed messages set $\mathcal{SM}$.

$\mathsf{Exp}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$
3. $(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk}, \mathtt{Sign}(\mathsf{vk}, \cdot))$
4. $b \leftarrow \mathsf{Verif}(\mathsf{vk}, m^*, \sigma^*)$
5. `IF` $m^* \in \mathcal{SM}$ `RETURN` 0
6. `ELSE RETURN` b

The probability of success against this game is denoted by

$$\mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}) = 1], \quad \mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S}}(\mathfrak{K}, t) = \max_{\mathcal{A} \leq t} \mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(\mathfrak{K}).$$

Blind Signature

The previous scheme can be extended in several ways. One of them is quite useful in electronic votes, and e-cash protocols, when someone might want an authority (Bank, Poll centre, ...) to sign a specific message he wants to keep secret. For that, we use the notion of *Blind Signature* $\mathcal{BS}$ introduced by Chaum [Cha83] for electronic cash in order to prevent the bank from linking a coin to its spender.

Such protocol can easily be derived from digital signatures. Instead of having a signing phase $\mathsf{Sign}(\mathsf{sk}, M; \mu)$ we have an interactive phase $\mathsf{BSProtocol}\langle \mathcal{S}, \mathcal{U} \rangle$ between the user $\mathcal{U}(\mathsf{vk}, M; \rho)$ who will (probably) transmit a masked information on M under some randomness ρ in order to obtain a signature valid under the verification key vk, and the signer $\mathcal{S}(\mathsf{sk}; \mu)$, who will generate something based on this value, and his secret key which should lead the user to a valid signature.

Such signatures are correct if when both the user and signer are honest then $\mathsf{BSProtocol}\langle \mathcal{S}, \mathcal{U} \rangle$ does indeed lead to valid signature on M under vk.

There are two additional security property, one protecting the signer, the other the user (cf Figure 2.2, page 23).

- On one hand, there is an *Unforgeability* property, where a malicious user shouldn't be able to compute $n+1$ valid signatures on different messages after at most n interactions with the signer.
- On the other hand, the *Blindness* property says that a malicious signer who signed two messages M_0 and M_1 shouldn't be able to decide which one was signed first.

$\mathsf{Exp}^{\mathsf{bl}-b}_{\mathcal{BS},\mathcal{S}^*}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{BSSetup}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, M_0, M_1) \leftarrow \mathcal{A}(\mathtt{FIND} : \mathsf{param})$
3. $\sigma_b \leftarrow \mathsf{BSProtocol}\langle \mathcal{A}, \mathcal{U}(\mathsf{vk}, M_b)\rangle$
4. $\sigma_{1-b} \leftarrow \mathsf{BSProtocol}\langle \mathcal{A}, \mathcal{U}(\mathsf{vk}, M_{1-b})\rangle$
5. $b^* \leftarrow \mathcal{S}^*(\mathtt{GUESS} : M_0, M_1)$;
6. $\mathtt{RETURN}\ b^* = b$.

$\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{BS},\mathcal{U}^*}(\mathfrak{K})$
1. $(\mathsf{param}) \leftarrow \mathsf{BSSetup}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{BSKeyGen}(\mathsf{param})$
3. For $i = 1, \ldots, q_s$, $\mathsf{BSProtocol}\langle \mathcal{S}(\mathsf{sk}), \mathcal{A}(\mathtt{INIT} : \mathsf{vk})\rangle$
4. $\big((m_1, \sigma_1), \ldots, (m_{q_s+1}, \sigma_{q_s+1})\big) \leftarrow \mathcal{A}(\mathtt{GUESS} : \mathsf{vk})$;
5. $\mathtt{IF}\ \exists i \neq j, m_i = m_j\ \mathtt{OR}\ \exists i, \mathsf{Verif}(\mathsf{pk}, m_i, \sigma_i) = 0\ \mathtt{RETURN}\ 0$
6. $\mathtt{ELSE\ RETURN}\ 1$

Figure 2.2: Security Games for the *Blind Signatures*

In [Fis06], Fischlin gave a generic construction of such signatures in a round-optimal way in the common-reference string (CRS) model. The first practical instantiation of round-optimal blind signatures in the standard model was proposed in [AFG+10] but it relies on non-standard computational assumptions. In section 4, page 70, we develop the result we presented in [BFPV11], where we achieved such round-optimality under classical hypothesis and reasonable communication costs. A round-optimal blind signature protocol is simply composed of two communications, one where the user sends some value so the signer, and then the signer answers.

Hash Function Family

A hash function family $\mathcal{H}$ is a family of functions $\mathfrak{H}_K$ from $\{0,1\}^*$ onto a fix-length output, either $\{0,1\}^k$ or $\mathbb{Z}_p$.

Universal One-Way [NY89]
⌜ A family is said to be *universal one-way* if for any adversary $\mathcal{A}$ on a family $\mathcal{H}$, it should be hard for him to pick conveniently a scalar x, such that he is able to find a second-preimage for a random function $\mathfrak{H}_K \in \mathcal{H}$. More precisely, we denote

$$\mathsf{Succ}^{\mathsf{uow}}_{\mathcal{H}}(\mathcal{A}) = \Pr[x \leftarrow \mathcal{A}(\mathcal{H}), \mathfrak{H}_K \xleftarrow{\$} \mathcal{H}, y \leftarrow \mathcal{A}(\mathfrak{H}_K) : \mathfrak{H}_K(x) = \mathfrak{H}_K(y)], \mathsf{Succ}^{\mathsf{uow}}_{\mathcal{H}}(t) = \max_{\mathcal{A} \leq t}\{\mathsf{Succ}^{\mathsf{uow}}_{\mathcal{H}}(\mathcal{A})\}.$$

⌟

In some cases, we may rely on a stronger property when we can't postpone the choice of the hash function in our simulations:

Collision-Resistant
⌜ A family is said to be *collision-resistant* if for any adversary $\mathcal{A}$ on a random function $\mathfrak{H}_K \xleftarrow{\$} \mathcal{H}$, it is hard to find a collision. More precisely, we denote

$$\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(\mathcal{A}) = \Pr[\mathfrak{H}_K \xleftarrow{\$} \mathcal{H}, (m_0, m_1) \leftarrow \mathcal{A}(\mathfrak{H}_K) : \mathfrak{H}_K(m_0) = \mathfrak{H}_K(m_1)], \qquad \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) = \max_{\mathcal{A} \leq t}\{\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(\mathcal{A})\}.$$

⌟

Pseudo-Random Function Family [GGM84]

A pseudo-random function family $\mathcal{F}$ is a family of functions $\mathfrak{F}_k : X \to Y$, such that a random function in $\mathcal{F}$ is indistinguishable from a random function in the set Y^X of functions from X into Y. More precisely, we consider an adversary $\mathcal{A}$ able to make q queries to the function f within time t, and we denote

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{A}) = |\Pr[\mathfrak{F}_k \xleftarrow{\$} \mathcal{F} : 1 \leftarrow \mathcal{A}^{\mathfrak{F}_k}()] - \Pr[f \xleftarrow{\$} Y^X : 1 \leftarrow \mathcal{A}^f()]|, \qquad \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t) = \max_{\mathcal{A} \leq t}\{\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{A})\}.$$

If one is able to add a proof π showing that the output of the function was generated honestly, this defines a Verifiable Random Function [MRV99].

Smooth Projective Hash Functions [CS02]

Smooth projective hash functions (SPHF) were introduced by Cramer and Shoup [CS02] for constructing encryption schemes. A projective hashing family is a family of hash functions that can be evaluated in two ways: using the (secret) hashing key, one can compute the function on every point in its domain, whereas using the (public) *projected* key one can only compute the function on a special subset of its domain. Such a family is deemed *smooth* if the value of the hash function on any point outside the special subset is independent of the projected key. The notion of SPHF has found applications in various contexts in cryptography (*e.g.* [GL03, Kal05, ACP09]), and we will rely on it, for our constructions in part II, page 92

Smooth Projective Hashing System

⌜ A Smooth Projective Hash Function over a language $\mathcal{L} \subset X$, onto a set $\mathbb{G}$, is defined by five algorithms $(\mathsf{SPHFSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$:

- $\mathsf{SPHFSetup}(1^{\mathfrak{K}})$ where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and the description of an $\mathcal{NP}$ language $\mathcal{L}$;
- $\mathsf{HashKG}(\mathcal{L}, \mathsf{param})$, outputs a hashing key hk for the language $\mathcal{L}$;
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$, derives the projection key hp, possibly depending on the word W [GL03, ACP09] thanks to the hashing key hk.
- $\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$, outputs a hash value $v \in \mathbb{G}$, thanks to the hashing key hk, and W
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w)$, outputs the hash value $v' \in \mathbb{G}$, thanks to the projection key hp and the witness w that $W \in \mathcal{L}$.

⌟

In the following, we consider $\mathcal{L}$ as a hard-partitioned subset of X, i.e. it is computationally hard to distinguish a random element in $\mathcal{L}$ from a random element in $X \setminus \mathcal{L}$.

A Smooth Projective Hash Function SPHF should satisfy the following properties:

- *Correctness*: Let $W \in \mathcal{L}$ and w a witness of this membership. Then, for all hashing keys hk and associated projection keys hp we have $\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W) = \mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w)$.
- *Smoothness*: For all $W \in X \setminus \mathcal{L}$ the following distributions are statistically indistinguishable:

$$\Delta_0 = \left\{ (\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{SPHFSetup}(1^{\mathfrak{K}}), \mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W), v = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W) \end{array} \right\}$$

$$\Delta_1 = \left\{ (\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{SPHFSetup}(1^{\mathfrak{K}}), \mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W), v \xleftarrow{\$} \mathbb{G} \end{array} \right\}.$$

  This is formalized by

$$\mathsf{Adv}^{\mathsf{smooth}}_{\mathrm{SPHF}}(\mathfrak{K}) = \sum_{V \in \mathbb{G}} \left| \Pr_{\Delta_1}[v = V] - \Pr_{\Delta_0}[v = V] \right| \text{ is negligible.}$$

- *Pseudo-Randomness*: If $W \in \mathcal{L}$, then without a witness of membership the two previous distributions should remain computationally indistinguishable: for any adversary $\mathcal{A}$ within reasonable time

$$\mathsf{Adv}^{\mathsf{pr}}_{\mathrm{SPHF},\mathcal{A}}(\mathfrak{K}) = |\Pr_{\Delta_1}[\mathcal{A}(\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, v) = 1] - \Pr_{\Delta_0}[\mathcal{A}(\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, v) = 1]| \text{ is negligible.}$$

Abdalla *et al.* [ACP09] explained how to combine SPHF to deal with conjunctions and disjunctions of the languages. In the following we simply recall those results:

Let us assume we have two Smooth Projective Hash Functions, defined by SPHF_1 and SPHF_2, on two languages, $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, both subsets of X, with hash values in the same group $(\mathbb{G}, \oplus)$. We note W an element of X, w_i a witness that $W \in \mathcal{L}_i$, $\mathsf{hk}_i = \mathsf{HashKG}_i(\mathcal{L}_i, \mathsf{param})$ and $\mathsf{hp}_i = \mathsf{ProjKG}_i(\mathsf{hk}_i, (\mathcal{L}_i, \mathsf{param}_i), W)$.

We can then define the SPHF on $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$, where $w = (w_1, w_2)$ as:

- $\mathsf{SPHFSetup}(1^{\mathfrak{K}})$, $\mathsf{param} = (\mathsf{param}_1, \mathsf{param}_2)$, and $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$;
- $\mathsf{HashKG}(\mathcal{L}, \mathsf{param})$: $\mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2)$
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$: $\mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2)$
- $\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$: $\mathsf{Hash}_1(\mathsf{hk}_1, (\mathcal{L}_1, \mathsf{param}_1), W) \oplus \mathsf{Hash}_2(\mathsf{hk}_2, (\mathcal{L}_2, \mathsf{param}_2), W)$
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w = (w_1, w_2))$:

$$\mathsf{ProjHash}_1(\mathsf{hp}_1, (\mathcal{L}_1, \mathsf{param}_1), W, w_1) \oplus \mathsf{ProjHash}_2(\mathsf{hp}_2, (\mathcal{L}_2, \mathsf{param}_2), W, w_2)$$

We can also define the SPHF on $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$, where $w = w_1$ or $w = w_2$ as:

- $\mathsf{SPHFSetup}(1^{\mathfrak{K}})$, $\mathsf{param} = (\mathsf{param}_1, \mathsf{param}_2)$, and $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$;
- $\mathsf{HashKG}(\mathcal{L}, \mathsf{param})$: $\mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2)$
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$: $\mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2, \mathsf{hp}_\Delta)$ where

$$\mathsf{hp}_\Delta = \mathsf{Hash}_1(\mathsf{hk}_1, (\mathcal{L}_1, \mathsf{param}_1), W) \oplus \mathsf{Hash}_2(\mathsf{hk}_2, (\mathcal{L}_2, \mathsf{param}_2), W)$$

- $\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$: $\mathsf{Hash}_1(\mathsf{hk}_1, (\mathcal{L}_1, \mathsf{param}_1), W)$
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w)$: If $W \in \mathcal{L}_1$, $\mathsf{ProjHash}_1(\mathsf{hp}_1, (\mathcal{L}_1, \mathsf{param}_1), W, w_1)$, else (if $W \in \mathcal{L}_2$), $\mathsf{hp}_\Delta \ominus \mathsf{ProjHash}_2(\mathsf{hp}_2, (\mathcal{L}_2, \mathsf{param}_2), W, w_2)$

2.3 Classical Instantiations

2.3.1 Waters Signature

To sign scalar message in the standard model, we often use Waters Signatures [Wat05]. This signature scheme is defined by four algorithms:

Waters Signature Scheme
$\ulcorner \mathcal{S} = (\mathsf{Setup}_{\mathcal{S}}, \mathsf{KeyGen}_{\mathcal{S}}, \mathsf{Sign}, \mathsf{Verif})$:

- $\mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and more specifically the bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, an extra generator h, and generators $(u_i)_{[\![0,k]\!]}$ for the Waters function, where k is a polynomial in $\mathfrak{K}$, $\mathcal{F}(m) = u_0 \prod_{i \in [\![1,k]\!]} u_i^{m_i}$, where $m = (m_1, \ldots, m_k) \in \{0,1\}^k$.
- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$ picks a random $x \xleftarrow{\$} \mathbb{Z}_p$ and outputs the secret key $\mathsf{sk} = Y = h^x$, and the verification key $\mathsf{vk} = X = g^x$;
- $\mathsf{Sign}(\mathsf{sk}, m; \mu)$ outputs a signature $\sigma(m) = (Y\mathcal{F}(m)^\mu, g^{-\mu})$;
- $\mathsf{Verif}(\mathsf{vk}, m, \sigma)$ checks the validity of σ, by checking if the following pairing equation holds: $e(g, \sigma_1) \cdot e(\mathcal{F}(m), \sigma_2) \stackrel{?}{=} e(X, h)$

$\lrcorner$

Theorem 2.3.1 *This scheme is* $\mathsf{EUF} - \mathsf{CMA}$ *under the* CDH *assumption.*

Knowledge of m is not mandatory to sign and verify, only $\mathcal{F}(m)$ is required. However as m is required in the security proof, if only $\mathcal{F}(m)$ is used an additional proof of knowledge of m, Π_m should be added, and so we indeed sign $(\mathcal{F}(m), \Pi_m)$, inviting to some transformations leading to blind signatures as we will see in section 4, page 70.

We will show in section 2.6.4, page 42, how to extend the security proof to a non-binary alphabet with a little alteration to the [HK08] approach where we will base our demonstration on the local central limit theorem. This will drastically shorten the size of the public key, by decreasing the number of u_i.

Theorem 2.3.2 *Waters Signature is randomizable if we define:*

- $\mathsf{Random}(\mathsf{vk}, \mathcal{F}(m), \sigma = (\sigma_1, \sigma_2); \mu')$ *outputs* $\sigma' = (\sigma_1 \cdot \mathcal{F}(m)^{\mu'}, \sigma_2 \cdot g^{-\mu'})$.

Proof: We simply have:

$$\sigma = \mathsf{Sign}(\mathsf{sk}, \mathcal{F}(m); \mu) \quad \Rightarrow \quad \mathsf{Random}(\mathsf{vk}, \mathcal{F}(m), \sigma; \mu') = \mathsf{Sign}(\mathsf{sk}, \mathcal{F}(m); \mu + \mu' \mod p).$$

Due to the additive law in $\mathbb{Z}_p$, fresh signature distribution is indistinguishable from randomized one. □

We have introduced in [BFPV11], some variants of Waters signature, particularly one in an asymmetrical setting (Type II, III). It is described in section 2.6.3, page 41

2.3.2 Pedersen Commitment

The Pedersen commitment [Ped92] is an equivocable commitment:

- $\mathsf{Setup}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with two independent generators g and ζ;
- $\mathsf{Commit}(m; r)$, for a message $m \xleftarrow{\$} \mathbb{Z}_p$ and random coins $r \xleftarrow{\$} \mathbb{Z}_p$, produces a commitment $c = g^m \zeta^r$;
- $\mathsf{Decommit}(c, m; r)$ outputs m and r, which opens c into m, with checking ability: $c \stackrel{?}{=} g^m \zeta^r$.

This commitment is computationally binding under the discrete logarithm assumption: two different openings (m, r) and (m', r') for a commitment c, leads to the discrete logarithm of ζ in basis g, that is equal to $(m' - m) \cdot (r - r')^{-1} \bmod p$. Granted this logarithm as additional information from the setup, one can equivocate any dummy commitment.

2.3.3 ElGamal Encryption / Commitment

Encryption

ElGamal encryption [ElG85] is defined by the following four algorithms:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: The scheme needs a multiplicative group $(p, \mathbb{G}, g)$,. The global parameters param consist of these elements $(p, \mathbb{G}, g)$.
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$: Chooses one random scalar $\mu \xleftarrow{\$} \mathbb{Z}_p$, which define the secret key $\mathsf{dk} = \mu$, and the public key $\mathsf{pk} = X = g^\mu$.
- $\mathsf{Encrypt}(\mathsf{pk} = X, M; \alpha)$: For a message $M \in \mathbb{G}$ and a random scalar $\alpha \xleftarrow{\$} \mathbb{Z}_p$, computes the ciphertext as $\mathbf{c} = \big(c_1 = X^\alpha M, c_2 = g^\alpha\big)$.
- $\mathsf{Decrypt}(\mathsf{dk} = \mu, \mathbf{c} = (c_1, c_2))$: One computes $M = c_1/(c_2^\mu)$.

As shown by Boneh [Bon98], this scheme is $\mathsf{IND} - \mathsf{CPA}$ under the hardness of DDH.

Theorem 2.3.3 *This encryption is randomizable:*

- $\mathsf{Random}(\mathsf{ek}, M, \mathbf{c}; \alpha')$ *outputs* $\mathbf{c}' = (c_1 \cdot X^{\alpha'}, c_2 \cdot g^{\alpha'})$*, for random* $\alpha' \xleftarrow{\$} \mathbb{Z}_p$.

One can easily see that this is equivalent to a fresh encryption with random $(\alpha + \alpha')$.

Cramer-Shoup Encryption

The Cramer-Shoup encryption scheme [CS98] is an $\mathsf{IND} - \mathsf{CCA2}$ version of the ElGamal Encryption. We present it here as a labeled public-key encryption scheme, the classical version is done with $\ell = \emptyset$.

- $\mathsf{Setup}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with a generator g
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$ generates $(g_1, g_2) \xleftarrow{\$} \mathbb{G}^2$, $\mathsf{dk} = (x_1, x_2, y_1, y_2, z) \xleftarrow{\$} \mathbb{Z}_p^5$, and sets, $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. It also chooses a Collision-Resistant hash function $\mathfrak{H}_K$ in a hash family $\mathcal{H}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (g_1, g_2, c, d, h, \mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = (\ell, \mathbf{u} = (g_1^r, g_2^r), e = M \cdot h^r, v = (cd^\xi)^r)$, where v is computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$: one first computes $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$ and checks whether $u_1^{x_1 + \xi y_1} \cdot u_2^{x_2 + \xi y_2} \stackrel{?}{=} v$. If the equality holds, one computes $M = e/(u_1^z)$ and outputs M. Otherwise, one outputs $\perp$.

The security of the scheme is proven under the DDH assumption and the fact the hash function used is a Universal One-Way Hash Function.

Commitment

A doubled-version of this encryption may be used to allow mercurial commitments.

Commitment Scheme
$\ulcorner \mathcal{C} = (\mathsf{Setup}_{\mathcal{C}}, \mathsf{WISetup}, \mathsf{ExSetup}, \mathsf{Commit}, \mathsf{Extract})$:

- $\mathsf{Setup}_{\mathcal{C}}(1^{\mathfrak{K}})$, the global parameters param consist of these elements $(p, \mathbb{G}, g)$, the commitment key ck is defined by running one of the following;
  - $\mathsf{WISetup}(1^{\mathfrak{K}})$ picks $(\mu, \nu) \xleftarrow{\$} \mathbb{Z}_p^2$ outputs a perfectly hiding commitment key $\mathsf{ck} = (u_{1,1} = g, u_{1,2} = g^{\mu}, u_{2,1} = g^{\nu}, u_{2,2} = g^{\mu\nu+1})$;
  - $\mathsf{ExSetup}(1^{\mathfrak{K}})$ picks $(\mu, \nu) \xleftarrow{\$} \mathbb{Z}_p^2$ outputs a perfectly binding commitment key $\mathsf{ck} = (u_{1,1} = g, u_{1,2} = g^{\mu}, u_{2,1} = g^{\nu}, u_{2,2} = g^{\mu\nu})$, and the extraction key is $\mathsf{ek} = \mu$;
- $\mathsf{Commit}(\mathsf{ck}, M; \alpha, \beta)$, outputs a commitment $\mathbf{c} = (u_{1,1}^{\alpha} u_{2,1}^{\beta}, M u_{1,2}^{\alpha} u_{2,2}^{\beta})$;
- $\mathsf{Extract}(\mathsf{ek}, \mathbf{c})$, if $(\mathsf{ek}, \mathsf{ck})$ were created through $\mathsf{ExSetup}$ then it computes $M = c_2 c_1^{-\mu}$.

$\lrcorner$

Such instantiation perfectly fits the previous commitment definition:

- In the case of an extractable setup, we simply have a standard ElGamal encryption where the encryption scalar is $\alpha + \nu\beta$, so it is indeed perfectly binding.
- In the case of a hiding setup, the extra 1 in $u_{2,2}$ leads to a mask of M by g^{β}, which leads to an extra degree of freedom in the system, and so with this key the commitments are perfectly hiding.
- The two cases are indistinguishable under the DDH assumption.

Like in the case of the ElGamal encryption, this scheme is randomizable.
In the following to ease the notation, the commitment key will be noted $(\mathbf{u}_1 = (u_{1,1}, u_{1,2}), \mathbf{u}_2 = (u_{2,1}, u_{2,2})) \in \mathbb{G}^{2\times 2}$, and a commitment $\mathcal{C}(M) := (1, M) \odot \mathbf{u}_1^{\alpha} \odot \mathbf{u}_2^{\beta} = (u_{1,1}^{\alpha} \cdot u_{2,1}^{\beta}, M u_{1,2}^{\alpha} u_{2,2}^{\beta})$.

2.3.4 Linear Encryption / Commitment

Encryption

Linear encryption is defined by the four algorithms:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: The scheme needs a multiplicative group $(p, \mathbb{G}, g)$. The global parameters param consist of these elements $(p, \mathbb{G}, g)$.
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$: Choose two random scalars $\mu, \nu \xleftarrow{\$} \mathbb{Z}_p$, which define the secret key $\mathsf{dk} = (\mu, \nu)$, and the public key $\mathsf{pk} = (X_1 = g^{\mu}, X_2 = g^{\nu})$.
- $\mathsf{Encrypt}(\mathsf{pk} = (X_1, X_2), M; \alpha, \beta)$: For a message $M \in \mathbb{G}$ and random scalars $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_p^2$, defines the ciphertext as $\mathbf{c} = \big(c_1 = X_1^{\alpha}, c_2 = X_2^{\beta}, c_3 = g^{\alpha+\beta} \cdot M\big)$.
- $\mathsf{Decrypt}(\mathsf{dk} = (\mu, \nu), \mathbf{c} = (c_1, c_2, c_3))$: One computes $M = c_3/(c_1^{1/\mu} c_2^{1/\nu})$.

As shown by Boneh, Boyen and Shacham [BBS04], this scheme is semantically secure against chosen-plaintext attacks ($\mathsf{IND} - \mathsf{CPA}$) under the hardness of DLin.

Theorem 2.3.4 *This encryption is randomizable:*

- $\mathsf{Random}(\mathsf{ek}, M, \mathbf{c}; \alpha', \beta')$ *outputs* $\mathbf{c}' = (c_1 \cdot X_1^{\alpha'}, c_2 \cdot X_2^{\beta'}, c_3 \cdot g^{\alpha'+\beta'})$, *for random* $\alpha', \beta' \xleftarrow{\$} \mathbb{Z}_p^2$.

One can easily see that this is equivalent to a fresh encryption with random $(\alpha + \alpha', \beta + \beta')$.

Linear Cramer-Shoup Encryption (LCS).

The Linear Cramer-Shoup encryption scheme [CKP07, Sha07] is the equivalent of Cramer-Shoup relying on the linear encryption. Once again we directly present the labeled version

- $\mathsf{Setup}(1^{\mathfrak{K}})$ generates a group $\mathbb{G}$ of order p, with three independent generators $(g_1, g_2, g_3) \xleftarrow{\$} \mathbb{G}^3$;
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$ generates $\mathsf{dk} = (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) \xleftarrow{\$} \mathbb{Z}_p^9$, and sets, for $i = 1, 2$, $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. It also chooses a hash function $\mathfrak{H}_K$ in a collision-resistant hash family $\mathcal{H}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (c_1, c_2, d_1, d_2, h_1, h_2, \mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r, s)$, for a message $M \in \mathbb{G}$ and two random scalars $r, s \xleftarrow{\$} \mathbb{Z}_p$, the ciphertext is $C = (\mathbf{u} = (g_1^r, g_2^s, g_3^{r+s}), e = M \cdot h_1^r h_2^s, v = (c_1 d_1^{\xi})^r (c_2 d_2^{\xi})^s)$, where v is computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$: one first computes $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$ and checks whether $u_1^{x_1+\xi y_1} \cdot u_2^{x_2+\xi y_2} \cdot u_3^{x_3+\xi y_3} \stackrel{?}{=} v$. If the equality holds, one computes $M = e/(u_1^{z_1} u_2^{z_2} u_3^{z_3})$ and outputs M. Otherwise, one outputs $\perp$.

This scheme is indistinguishable under chosen-ciphertext attacks, under the DLin assumption and if one uses a collision-resistant hash function (or simply a Universal One-Way Hash Function).

We will present in section 2.6.5, page 46, a *revised*-version of this scheme where we use a global hash to encrypt several messages.

Commitment

A doubled-version of this encryption may be used to allow mercurial commitments.

Commitment Scheme

$\ulcorner \mathcal{C} = (\mathsf{Setup}_{\mathcal{C}}, \mathsf{WISetup}, \mathsf{ExSetup}, \mathsf{Commit}, \mathsf{Extract})$:

- $\mathsf{Setup}_{\mathcal{C}}(1^{\mathfrak{K}})$, the global parameters param consist of $(p, \mathbb{G}, g)$, the commitment key ck is defined by running one of the following;
  - $\mathsf{WISetup}(1^{\mathfrak{K}})$ picks $(\mu, \nu, \rho, \tau) \xleftarrow{\$} \mathbb{Z}_p^4$ outputs a perfectly hiding commitment key $\mathsf{ck} = (u_{1,1} = g^{\mu}, u_{2,2} = g^{\nu}, u_{3,1} = g^{\mu\rho}, u_{3,2} = g^{\nu\tau}, u_{3,3} = g^{1+\rho+\tau})$;
  - $\mathsf{ExSetup}(1^{\mathfrak{K}})$ picks $(\mu, \nu, \rho, \tau) \xleftarrow{\$} \mathbb{Z}_p^4$ outputs a perfectly binding commitment key $\mathsf{ck} = (u_{1,1} = g^{\mu}, u_{2,2} = g^{\nu}, u_{3,1} = g^{\mu\rho}, u_{3,2} = g^{\nu\tau}, u_{3,3} = g^{\rho+\tau})$, and the extraction key is $\mathsf{ek} = (\mu, \nu)$;
- $\mathsf{Commit}(\mathsf{ck}, M; \alpha, \beta, \gamma)$, outputs a commitment $\mathbf{c} = (u_{1,1}^{\alpha} u_{3,1}^{\gamma}, u_{2,2}^{\beta} u_{3,2}^{\gamma}, g^{\alpha+\beta} u_{3,3}^{\gamma} M)$;
- $\mathsf{Extract}(\mathsf{ek}, \mathbf{c})$, if $(\mathsf{ek}, \mathsf{ck})$ were created through $\mathsf{ExSetup}$ then it computes $c_3/(c_1^{1/\mu} c_2^{1/\nu}) = M$.

$\lrcorner$

Such instantiation perfectly fits the previous commitment definition:

- In the case of an extractable setup, we simply have a standard linear encryption where the encryption scalars are respectively $\alpha + \rho\gamma, \beta + \tau\gamma$, so it is indeed perfectly binding.
- In the case of a hiding setup, the extra 1 in $u_{3,3}$ leads to a mask of M by g^{γ}, which leads to an extra degree of freedom in the system, and so with this key the commitments are perfectly hiding.
- The two cases are indistinguishable under the DLin assumption.

Like in the case of the linear encryption, this scheme is randomizable.

In the following to ease the notation, the commitment key ck will be noted $(\mathbf{u}_1 = (u_{1,1}, 1, g), \mathbf{u}_2 = (1, u_{2,2}, g), \mathbf{u}_3 = (u_{3,1}, u_{3,2}, u_{3,3})) \in \mathbb{G}^{3\times 3}$, and a commitment $\mathcal{C}(M) := (1, 1, M) \odot \mathbf{u}_1^{\alpha} \odot \mathbf{u}_2^{\beta} \odot \mathbf{u}_3^{\gamma} = (u_{1,1}^{\alpha} \cdot u_{3,1}^{\gamma}, u_{2,2}^{\beta} \cdot u_{3,2}^{\gamma}, M \cdot g^{\alpha+\beta} \cdot u_{3,3}^{\gamma})$.

2.4 Zero-Knowledge and Witness Indistinguishable Proofs

Zero-Knowledge Proofs are a powerful tool introduced by Goldwasser, Micali and Rackoff in [GMR89]. They let a user prove the veracity of a statement $\mathcal{S}$ without leaking any additional information. They have found many applications in various protocols since (Anonymous Credentials, Anonymous Signatures (Group Signatures, Ring Signatures, Blind Signatures,...), Online Voting, PAKE, Proof of a shuffle, ...).

Such proofs are expected to have three properties:

- **Completeness:** If $\mathcal{S}$ is true, the honest verifier will be convinced of this fact.
- **Soundness:** If $\mathcal{S}$ is false, no cheating prover can convince the honest verifier that it is true except with negligible probability.
- **Zero-knowledge:** Anything that is feasibly computable from the proof is also feasibly computable from the assertion itself .

An upgrade to this kind of proofs can be obtained by removing the interaction between the prover and the verifier. In this case we speak about NIZK, Non-interactive Zero-Knowledge Proof. Fiat Shamir [FS87] presented an heuristic showing how to transform Interactive proofs into Non-Interactive ones. Several approaches have been proposed since ([BFM90], [DDO+01],...), but they are all rather inefficient, until Groth-Sahai methodology in [GS08] to prove pairing equations.

Zero-Knowledge Proofs can also be relaxed into Witness-Indistinguishable proofs [FS90], where the proof hides the witness used to prove a statement. Efficient Groth-Sahai proofs are only NIWI and not NIZK.

To formally define a randomizable WI/ZK proof system Proof for some commitment scheme Com on equations, we need additional algorithms $\mathsf{Prove}, \mathsf{Verify}$. Prove takes as input several values M_i satisfying some equations E, a commitment key ck, a valid description $\mathcal{E}$ of the equation E (for clarity in the following we will identify the description to the equation and so write E as the input used in algorithms), some randomness μ_i, and outputs a proof π, together with $\vec{\mathbf{c}}$ a matrix of commitments to the witnesses. Verify then takes as inputs this matrix $\vec{\mathbf{c}}$, the commitment key ck together with the proof π, the description E and outputs either 0 or 1 depending on the validity of the proof. We may define as before a randomization algorithm, $\mathsf{RandProof}$ which outputs randomized values of $\vec{\mathbf{c}}, \pi$ for some new randomness.

We require four security properties:

- *Completeness:* For every honest scenario, i.e. values M_i satisfying E, and every randomness μ_i, we should have $\mathsf{Verify}(\mathsf{Prove}((M_i), \mathsf{ck}, E; \mu_i), \mathsf{ck}, E) = 1$.
- *Soundness:* If the commitment scheme is initialized in the perfectly binding setup. Then for all valid proofs π, we have $(M_i) = \mathsf{Extract}(\mathsf{ek}, \vec{\mathbf{c}})$, such that (M_i) does indeed verify E. (In other words, if the initial values M_i does not verify E, it should be impossible to generate a valid proof π.)
- *Randomizability:* As usual a randomized proof should have the same distribution as a fresh one.

WI *Witness-Indistinguishability:* If ck is generated by the perfectly hiding setup, and there exists both $(M_i), (M'_i)$ verifying an equation E, and $(\mu_i), (\nu_i)$ such that $\forall i, \mathsf{Com}(\mathsf{ck}, M_i, \mu_i) = \mathsf{Com}(\mathsf{ck}, M'_i, \nu_i)$, then π and π' should have the same distribution.

ZK *Zero-Knowledge:* If ck is generated by the perfectly hiding setup, then for all $E, (M_i) \in \mathcal{L}_E, (M'_i) \notin \mathcal{L}_E$ and $(\mu_i), (\nu_i)$ such that $\forall i, \mathsf{Com}(\mathsf{ck}, M_i, \mu_i) = \mathsf{Com}(\mathsf{ck}, M'_i, \nu_i)$, π and π' should have the same distribution.

2.4.1 Groth-Sahai Methodology

Groth and Sahai have introduced a methodology to build Non-Interactive Zero-Knowledge / Witness Indistinguishable proofs of *satisfiability of pairing-product like equations*. The three types of equations handled by such proofs are the following:

A *pairing-product equation* over variables $\vec{\mathcal{X}} = (\mathcal{X}_1, \ldots, \mathcal{X}_m) \in \mathbb{G}_1^m$ and $\vec{\mathcal{Y}} = (\mathcal{Y}_1, \ldots, \mathcal{Y}_n) \in \mathbb{G}_2^n$ is of the form

$$\langle \vec{\mathcal{A}}, \underline{\vec{\mathcal{Y}}} \rangle \cdot \langle \underline{\vec{\mathcal{X}}}, \vec{\mathcal{B}} \rangle \cdot \langle \underline{\vec{\mathcal{X}}}, \Gamma \underline{\vec{\mathcal{Y}}} \rangle = t_T \ , \tag{2.1}$$

defined by constants $\vec{\mathcal{A}} \in \mathbb{G}_1^n$, $\vec{\mathcal{B}} \in \mathbb{G}_2^m$, $\Gamma = (\gamma_{i,j})_{\substack{1 \le i \le m \\ 1 \le j \le n}} \in \mathbb{Z}_p^{m \times n}$ and $t_T \in \mathbb{G}_T$.

A *multi-scalar multiplication equation* over variables $\vec{y} \in \mathbb{Z}_p^n$ and $\vec{\mathcal{X}} \in \mathbb{G}_1^m$ is of the form

$$\langle \underline{\vec{y}}, \vec{\mathcal{A}} \rangle \cdot \langle \vec{b}, \underline{\vec{\mathcal{X}}} \rangle \cdot \langle \underline{\vec{y}}, \Gamma \underline{\vec{\mathcal{X}}} \rangle = T, \tag{2.2}$$

defined by the constants $\vec{\mathcal{A}} \in \mathbb{G}_1^n$, $\vec{b} \in \mathbb{Z}_p^m$, $\Gamma \in \mathbb{Z}_p^{m \times n}$ and $T \in \mathbb{G}_1$.

A multi-scalar multiplication equation in group $\mathbb{G}_2$ is defined analogously.

A *quadratic equation in* $\mathbb{Z}_p$ over variables $\vec{x} \in \mathbb{Z}_p^m$ and $\vec{y} \in \mathbb{Z}_p^n$ is of the form

$$\langle \vec{a}, \underline{\vec{y}} \rangle + \langle \underline{\vec{x}}, \vec{b} \rangle + \langle \underline{\vec{x}}, \Gamma \underline{\vec{y}} \rangle = t, \tag{2.3}$$

defined by the constants $\vec{a} \in \mathbb{Z}_p^n$, $\vec{b} \in \mathbb{Z}_p^m$, $\Gamma \in \mathbb{Z}_p^{m \times n}$ and $t \in \mathbb{Z}_p$.

Groth and Sahai have detailed generic construction of the proofs π and specific instantiations under different security assumptions. We will focus on those based on linear commitments and ElGamal commitments as our first primitives are based on such proofs, and so totally skip those presented in composite order groups.

ElGamal Instantiation

In order to generate a proof of such relations, the methodology invites us to commit to the witness vectors $\vec{\mathcal{X}}$ with randomness $\vec{R}$, and to $\vec{\mathcal{Y}}$ with $\vec{S}$ with two double ElGamal commitments scheme (recalled in 2.3.3, page 27), one in $\mathbb{G}_1$ and one in $\mathbb{G}_2$ with respective commitment keys $\mathbf{u} \in \mathbb{G}_1^{2\times 2}$ and $\mathbf{v} \in \mathbb{G}_2^{2\times 2}$. As both need to be semantically secure, we will work under the SXDH assumption, so on type III curves.

We will note $\iota_1(g_1)=(1_1, g_1), \iota_2(g_2)=(1_2, g_2), \iota_T(t_T) := \begin{pmatrix} 1_T & 1_T \\ 1_T & t_T \end{pmatrix}$ and focus on product pairing equations.

For that we have:

- $\mathsf{Prove}((\mathcal{X}_i), (\mathcal{Y}_j), (\mathbf{u}, \mathbf{v}), E; (R_i), (S_j), T \in \mathbb{Z}_p^{2\times 2})$ outputs a proof $\pi = (\phi, \theta)$, together with $\vec{\mathbf{c}}, \vec{\mathbf{d}} \in \mathbb{G}_1^{2\times m} \times \mathbb{G}_2^{2\times n}$ commitments to the witnesses with respective randomness $\vec{R} \in \mathbb{Z}^{2\times m}, \vec{S} \in \mathbb{Z}^{2\times n}$. The proof is at most composed of four elements in $\mathbb{G}_1$ and four in $\mathbb{G}_2$.
  $\phi = \vec{R}^\top \iota_2(\vec{\mathcal{B}}) + \vec{R}^\top \Gamma \iota_2(\vec{\mathcal{Y}}) + (\vec{R}^\top \Gamma \vec{S} - T^\top)\mathbf{v}$
  $\theta = \vec{S}^\top \iota_1(\vec{\mathcal{A}}) + \vec{S}^\top \Gamma^\top \iota_1(\vec{\mathcal{X}}) + T\mathbf{u}$
- $\mathsf{Verify}((\vec{\mathbf{c}}, \vec{\mathbf{d}}), \mathsf{ck}, E, (\phi, \theta))$ checks if:
$$\left(\iota_1(\vec{\mathcal{A}}) \bullet \vec{\mathbf{d}}\right) \odot \left(\vec{\mathbf{c}} \bullet \iota_2(\vec{\mathcal{B}})\right) \odot \left(\vec{\mathbf{c}} \bullet \Gamma \vec{\mathbf{d}}\right) = \iota_T(t_T) \odot \left(\vec{\mathbf{u}} \bullet \phi\right) \odot \left(\theta \bullet \vec{\mathbf{v}}\right)$$
- $\mathsf{RandProof}$ outputs randomized values of $\vec{\mathbf{c}}, \vec{\mathbf{d}}, \pi$ for some new randomness $(\vec{R}', \vec{S}', T')$. Given the definition of ϕ, θ this randomization is quite straightforward

The *Soundness* and the *Witness Indistinguishability* of such a proof directly come from the security of the commitment, and the extra randomness T.

Intuitively each term in the proof is here to compensate some part introduced by the randoms in the verification equation: $\vec{R}^\top \iota_2(\vec{\mathcal{B}})$ will be matched with the random part in $\left(\vec{\mathbf{c}} \bullet \iota_2(\vec{\mathcal{B}})\right)$, $\vec{S}^\top \iota_1(\vec{\mathcal{A}})$ with $\left(\iota_1(\vec{\mathcal{A}}) \bullet \vec{\mathbf{d}}\right)$, $\vec{R}^\top \Gamma \iota_2(\vec{\mathcal{Y}}), \vec{S}^\top \Gamma^\top \iota_1(\vec{\mathcal{X}})$ will each annihilate the extra terms in the pairing between one of the plaintext with the commitment of the other, while $\vec{R}^\top \Gamma \vec{S} \mathbf{v}$ will remove the pairing between the two randoms. ($\left(\vec{\mathbf{c}} \bullet \Gamma \vec{\mathbf{d}}\right)$ can be viewed as $\vec{\mathcal{X}}_i \bullet \Gamma \vec{\mathbf{d}} \odot \vec{\mathbf{c}} \bullet \Gamma \vec{\mathcal{Y}}_i \odot \vec{R} \odot \Gamma \vec{S}$), the extra terms in T are here to randomize the proof.

Examples

1. *Proof of equality:* Let's consider an equation like $e(\underline{\mathcal{X}_1}, g_2)/e(\underline{\mathcal{X}_2}, g_2) = 1_T$. We commit to $\mathcal{X}_i$ in $\mathbb{G}_1$ by computing $\mathbf{c}_i = \iota_1(\mathcal{X}_i) + R_i \mathbf{u} = (u_{1,1}^{R_{1,i}} u_{2,1}^{R_{2,i}}, \mathcal{X}_i u_{1,2}^{R_{1,i}} u_{2,2}^{R_{2,i}})$. The proof is then: $\phi = R^\top \iota_2(\mathcal{B}) - T^\top \mathbf{v}, \theta = T\mathbf{u}$. In this case θ does not hide the value T, therefore we can only use $\pi = \phi = R^\top \iota_2(\vec{\mathcal{B}})$. Furthermore due to the nature of ι_2, $\pi = \begin{pmatrix} 1_2 & g_2^{R_{1,1}+R_{2,1}} \\ 1_2 & g_2^{R_{1,2}+R_{2,2}} \end{pmatrix}$, and so we only need 2 group elements in $\mathbb{G}_2$ for the proof. (The initial equation without any variable $\mathcal{Y} \in \mathbb{G}_2$ is called a linear pairing product equation.)

2. *Proof of a Diffie Hellman tuple:* Let's consider an equation like $e(\underline{\mathcal{X}}, g_2)/e(g_1, \underline{\mathcal{Y}}) = 1_T$.

   (a) Picks random $(r_1, r_2, s_1, s_2) \xleftarrow{\$} \mathbb{Z}_p^4$ and computes the commitments to the variables:

   $$\mathbf{c} = (u_{1,1}^{r_1} u_{2,1}^{r_2}, \mathcal{X} u_{1,2}^{r_1} u_{2,2}^{r_2}), \mathbf{d} = (v_{1,1}^{s_1} v_{2,1}^{s_2}, \mathcal{Y} v_{1,2}^{s_1} v_{2,2}^{s_2}),$$

   (b) The equation is a pairing product equation, where $\mathcal{A} = g_1^{-1}, \mathcal{B} = g_2$ and Γ is null, so he now picks $T \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ and computes:

   $$\begin{aligned} \phi &= R^\top \mathcal{B} - T^\top \mathbf{v} = \begin{pmatrix} v_{1,1}^{-t_{1,1}} v_{2,1}^{-t_{2,1}} & g_2^{r_1} v_{1,2}^{-t_{1,1}} v_{2,2}^{-t_{2,1}} \\ v_{1,1}^{-t_{1,2}} v_{2,1}^{-t_{2,2}} & g_2^{r_2} v_{1,2}^{-t_{1,2}} v_{2,2}^{-t_{2,2}} \end{pmatrix}, \\ \theta &= S^\top \mathcal{A} + T\mathbf{u} = \begin{pmatrix} u_{1,1}^{t_{1,1}} u_{2,1}^{t_{1,2}} & g_1^{-s_1} u_{1,2}^{t_{1,1}} u_{2,2}^{t_{1,2}} \\ u_{1,1}^{t_{2,1}} u_{2,1}^{t_{2,2}} & g_1^{-s_2} u_{1,2}^{t_{2,1}} u_{2,2}^{t_{2,2}} \end{pmatrix} \end{aligned}$$

   (c) To check the proof, one needs to check if:

   $$\left[\iota_1(\mathcal{A}) \bullet \vec{\mathbf{d}}\right] + \left[\vec{\mathbf{c}} \bullet \iota_2(\mathcal{B})\right] \stackrel{?}{=} \left[\mathbf{u} \bullet \pi\right] + \left[\theta \bullet \mathbf{v}\right].$$

   In other words, does $L = \left[(1_1, g_1^{-1}) \bullet (v_{1,1}^{s_1} v_{2,1}^{s_2}, Y v_{1,2}^{s_1} v_{2,2}^{s_2})\right] \odot \left[(u_{1,1}^{r_1} u_{2,1}^{r_2}, X u_{1,2}^{r_1} u_{2,2}^{r_2}) \bullet (1_2, g_2)\right]$ equal $R = \left[\mathbf{u} \bullet \pi\right] \odot \left[\theta \bullet \mathbf{v}\right]$?

$$\begin{aligned} L &= \begin{pmatrix} 1_T & 1_T \\ e(g_1^{-1}, v_{1,1}^{s_1} v_{2,1}^{s_2}) & e(g_1^{-1}, Y v_{1,2}^{s_1} v_{2,2}^{s_2}) \end{pmatrix} \odot \begin{pmatrix} 1_T & e(u_{1,1}^{r_1} u_{2,1}^{r_2}, g_2) \\ 1_T & e(X u_{1,2}^{r_1} u_{2,2}^{r_2}, g_2) \end{pmatrix} \\ &= \begin{pmatrix} 1_T & e(u_{1,1}^{r_1} u_{2,1}^{r_2}, g_2) \\ e(g_1^{-1}, v_{1,1}^{s_1} v_{2,1}^{s_2}) & e(g_1^{-1}, v_{1,2}^{s_1} v_{2,2}^{s_2}) \cdot e(u_{1,2}^{r_1} u_{2,2}^{r_2}, g_2) \end{pmatrix} \odot \begin{pmatrix} 1_T & 1_T \\ 1_T & e(g_1^{-1}, Y).e(X, g_2) \end{pmatrix} \\ R &= \begin{pmatrix} e(u_{1,1}, v_{1,1}^{-t_{1,1}} v_{2,1}^{-t_{2,1}}) \cdot e(u_{2,1}, v_{1,1}^{-t_{1,2}} v_{2,1}^{-t_{2,2}}) & e(u_{1,1}, g_2^{r_1} v_{1,2}^{-t_{1,1}} v_{2,2}^{-t_{2,1}}) \cdot e(u_{2,1}, g_2^{r_2} v_{1,2}^{-t_{1,2}} v_{2,2}^{-t_{2,2}}) \\ e(u_{1,2}, v_{1,1}^{-t_{1,1}} v_{2,1}^{-t_{2,1}}) \cdot e(u_{2,2}, v_{1,1}^{-t_{1,2}} v_{2,1}^{-t_{2,2}}) & e(u_{1,2}, g_2^{r_1} v_{1,2}^{-t_{1,1}} v_{2,2}^{-t_{2,1}}) \cdot e(u_{2,2}, g_2^{r_2} v_{1,2}^{-t_{1,2}} v_{2,2}^{-t_{2,2}}) \end{pmatrix} \\ &\odot \begin{pmatrix} e(u_{1,1}^{t_{1,1}} u_{2,1}^{t_{1,2}}, v_{1,1}) \cdot e(u_{1,1}^{t_{2,1}} u_{2,1}^{t_{2,2}}, v_{2,1}) & e(u_{1,1}^{t_{1,1}} u_{2,1}^{t_{1,2}}, v_{1,2}) \cdot e(u_{1,1}^{t_{2,1}} u_{2,1}^{t_{2,2}}, v_{2,2}) \\ e(g_1^{-s_1} u_{1,2}^{t_{1,1}} u_{2,2}^{t_{1,2}}, v_{1,1}).e(g_1^{-s_2} u_{1,2}^{t_{2,1}} u_{2,2}^{t_{2,2}}, v_{2,1}) & e(g_1^{-s_1} u_{1,2}^{t_{1,1}} u_{2,2}^{t_{1,2}}, v_{1,2}).e(g_1^{-s_2} u_{1,2}^{t_{2,1}} u_{2,2}^{t_{2,2}}, v_{2,2}) \end{pmatrix} \\ &= \begin{pmatrix} 1_T & e(u_{1,1}^{r_1} u_{2,1}^{r_2}, g_2) \\ e(g_1^{-1}, v_{1,1}^{s_1} v_{2,1}^{s_2}) & e(g_1^{-1}, v_{1,2}^{s_1} v_{2,2}^{s_2}).e(u_{1,2}^{r_1} u_{2,2}^{r_2}, g_2) \end{pmatrix} \end{aligned}$$

   And so $L = R$ if and only if $\begin{pmatrix} 1_T & 1_T \\ 1_T & e(g_1^{-1}, Y).e(X, g_2) \end{pmatrix} = \begin{pmatrix} 1_T & 1_T \\ 1_T & 1_T \end{pmatrix}$.
   So if indeed $e(g_1^{-1}, Y).e(X, g_2) = 1_T$.

Linear instantiation

Due to the fact that $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$ in this setting, the equations (2.1), (2.2) and (2.3) simplify to the following equations respectively:

$$\begin{aligned} \langle \vec{\mathcal{A}}, \underline{\vec{\mathcal{Y}}} \rangle \cdot \langle \underline{\vec{\mathcal{Y}}}, \Gamma \underline{\vec{\mathcal{Y}}} \rangle &= t_T \\ \langle \vec{a}, \underline{\vec{\mathcal{Y}}} \rangle \cdot \langle \underline{\vec{x}}, \vec{\mathcal{B}} \rangle \cdot \langle \underline{\vec{x}}, \Gamma \underline{\vec{\mathcal{Y}}} \rangle &= T \\ \langle \underline{\vec{x}}, \vec{b} \rangle + \langle \underline{\vec{x}}, \Gamma \underline{\vec{x}} \rangle &= t \end{aligned}$$

In order to generate a proof of such relation, the methodology invites us to commit to the witness vectors $\vec{\mathcal{X}}_i$ with a linear commitment scheme (recalled in section 2.3.4, page 28) in $\mathbb{G}$ with a commitment key $\mathbf{u} \in \mathbb{G}^{3\times 3}$. We will work under the DLin assumption, so we can use type I curves.

We define $\iota(\mathcal{X}) := (1, 1, \mathcal{X}), \iota_T(t_T) := \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & t_T \end{pmatrix}$.

We also define $H_1 := \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, H_2 := \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, H_3 := \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$, it should be noted that $\mathbf{u} \overset{s}{\bullet} H_i \mathbf{u} = 0$, and the H_i are antisymmetric.

To commit to $\mathcal{X} \in \mathbb{G}$, one chooses randomness $R = (\alpha, \beta, \gamma) \xleftarrow{\$} \mathbb{Z}_p^3$ and sets $\mathbf{c}_\mathcal{X} := \iota(\mathcal{X}) \odot \mathbf{u}_1^\alpha \odot \mathbf{u}_2^\beta \odot \mathbf{u}_3^\gamma$, to commit $x \in \mathbb{Z}_p$, one chooses randomness $S = (\eta, \theta) \xleftarrow{\$} \mathbb{Z}_p^2$ and sets $\mathbf{d}_x := \iota'(x) \odot \mathbf{u}_1^\eta \odot \mathbf{u}_2^\theta$. We define the 3×2 matrix $\mathbf{v} = (\mathbf{u}_1, \mathbf{u}_2)$.

For that we have:

- $\mathsf{Prove}(\vec{\mathcal{X}}, \mathsf{ck}, E; R, T \in \mathbb{Z}_p^3)$ outputs a proof π, together with $\vec{\mathbf{c}} \in \mathbb{G}^{3\times n}$ matrix of commitments to the witnesses. The proof is at most composed of nine group elements in $\mathbb{G}$ for pairing product equations, multi-scalar equations and 6 for quadratic equations.

  (2.1) $\pi = R^\top \iota(\vec{\mathcal{A}}) + R^\top(\Gamma + \Gamma^\top)\iota(\vec{\mathcal{X}}) + R^\top \Gamma R\mathbf{u} + TH\mathbf{u}$. Where TH is the sum of the $T_i H_i$.

  (2.2) $\pi = R^\top \iota(\vec{\mathcal{B}}) + R^\top \Gamma \iota(\vec{\mathcal{Y}}) + {S'}^\top \iota'(\vec{a}) + {S'}^\top \Gamma^\top \iota'(\vec{x}) + R^\top \Gamma S'\mathbf{u} + tH_1\mathbf{u}$.

  (2.3) $\pi = {S'}^\top \iota'(\vec{b}) + {S'}^\top(\Gamma + \Gamma^\top)\iota'(\vec{x}) + S^\top \Gamma S'\mathbf{u} + tH_1\mathbf{v}$.

- $\mathsf{Verify}(\vec{\mathbf{c}}, \mathsf{ck}, E, \pi)$ checks if the proof is valid

  (2.1) $\left(\iota(\vec{\mathcal{A}}) \overset{s}{\bullet} \vec{\mathbf{d}}\right) \odot \left(\vec{\mathbf{d}} \overset{s}{\bullet} \Gamma\vec{\mathbf{d}}\right) = \iota_T(t_T) \odot \left(\vec{\mathbf{u}} \overset{s}{\bullet} \vec{\pi}\right)$

  (2.2) $\left(\iota'(\vec{a}) \overset{s}{\bullet} \vec{\mathbf{d}}\right) \odot \left(\vec{\mathbf{c}} \overset{s}{\bullet} \iota(\vec{\mathcal{B}})\right) \odot \left(\vec{\mathbf{c}} \overset{s}{\bullet} \Gamma\vec{\mathbf{d}}\right) = \hat{\iota}_T(T) \odot \left(\vec{\mathbf{u}} \overset{s}{\bullet} \vec{\pi}\right)$

  (2.3) $\left(\vec{\mathbf{c}} \overset{s}{\bullet} \iota'(\vec{b})\right) \odot \left(\vec{\mathbf{c}} \overset{s}{\bullet} \Gamma\vec{\mathbf{c}}\right) = {\iota'}_T(t) \odot \left(\vec{\mathbf{v}} \overset{s}{\bullet} \vec{\pi}\right)$

- $\mathsf{RandProof}(\vec{\mathbf{c}}, \pi, \mathsf{ck}; R', T')$ for some new randomness (R', T'). Given the definition of π this randomization is quite straightforward: $\vec{\mathbf{c}'} = \vec{\mathcal{X}} \odot \vec{\mathbf{u}_1}^{R'_1} \odot \vec{\mathbf{u}_2}^{R'_2} \odot \vec{\mathbf{u}_3}^{R'_3}$ and then:

$$\begin{aligned}
\pi' &= \pi + {R'}^\top \iota(\vec{\mathcal{A}}) + {R'}^\top(\Gamma + \Gamma^\top)\iota(\vec{\mathbf{c}}) + {R'}^\top \Gamma R'\mathbf{u} + T'H\mathbf{u} \\
&= (R^\top \iota(\vec{\mathcal{A}}) + R^\top(\Gamma + \Gamma^\top)\iota(\vec{\mathcal{X}}) + R^\top \Gamma R\mathbf{u} + TH\mathbf{u}) \\
&\quad + {R'}^\top \iota(\vec{\mathcal{A}}) + {R'}^\top(\Gamma + \Gamma^\top)(\iota(\vec{\mathcal{X}}) + R\mathbf{u}) + {R'}^\top \Gamma R'\mathbf{u} + T'H\mathbf{u} \\
&= (R + R')^\top \iota(\vec{\mathcal{A}}) + (R + R')^\top(\Gamma + \Gamma^\top)\iota(\vec{\mathcal{X}}) + (R + R')^\top \Gamma(R + R')\mathbf{u} \\
&\quad + ({R'}^\top \Gamma^\top R - R^\top \Gamma R')\mathbf{u} + (T + T')H\mathbf{u} \\
&= S^\top \iota(\vec{\mathcal{A}}) + S^\top(\Gamma + \Gamma^\top)\iota(\vec{\mathcal{X}}) + S^\top \Gamma S\mathbf{u} + (T^* H)\mathbf{u}
\end{aligned}$$

  The last transformation is possible thanks to the antisymmetry of the matrices H, and of $({R'}^\top \Gamma^\top R - R^\top \Gamma R')\mathbf{u}$.

Therefore a use of $\mathsf{RandProof}(\mathsf{Prove}(\vec{\mathcal{X}}, \mathsf{ck}, E; R, T), \pi, \mathsf{ck}; R', T')$ is indeed equivalent to a fresh proof generated by $\mathsf{Prove}(\vec{\mathcal{X}}, \mathsf{ck}, E; S, T^*)$. The same can be done for $\vec{\mathbf{d}}$ and the associated proofs.

The proof π may seem more complicated than in the SXDH case, however its logic is the same, the main difference is the term used to compensate the quadratic term, as in this case both vectors $\vec{\mathcal{X}}$ are the same.

Examples

1. *Proof of equality:* Let's consider an equation like $e(\underline{\mathcal{X}_1}, g)/e(\underline{\mathcal{X}_2}, g) = 1_T$. We commit to $\mathcal{X}_i$ in $\mathbb{G}$ by computing $\mathbf{c}_i = \iota(\mathcal{X}_i) + R_i\mathbf{u}$. The proof is then: $\pi = R^\top \iota(\vec{\mathcal{A}}) + TH\mathbf{u}$. As noted in the original paper, we can use the asymmetric $\bullet$, in this case there aren't any non-trivial solution to $\mathbf{u} \bullet H\mathbf{u}$, and so π can be reduced to $R^\top \iota(\vec{\mathcal{A}})$ and so to $R\vec{\mathcal{A}} = (g^{R_{11}-R_{21}}, g^{R_{12}-R_{22}}, g^{R_{13}-R_{23}})$, which is composed of only three group elements. (The initial equation without any Γ is called a linear pairing product equation and only leads to reduced proofs)

2. *Proof of a Diffie Hellman tuple:* Let's consider an equation like $e(\underline{\mathcal{X}_1}, g)/e(\underline{\mathcal{X}_2}, h) = 1_T$. We commit to $\mathcal{X}_i$ in $\mathbb{G}$ by computing $\mathbf{c}_i = \iota(\mathcal{X}_i) + R_i\mathbf{u}$, and then the proof is: $\pi = R^\top \iota(\vec{\mathcal{A}}) + TH\mathbf{u}$. But like in the previous example, we can use the other map, to achieve a 3 elements proof. This is a huge difference with the SXDH instantiation, where such equation wasn't linear and so the proof was bigger.

2.4.2 Initial Optimization of Groth-Sahai Proofs

From what we have just seen, Groth-Sahai proofs are quite efficient: at most 9 group elements in the linear case, 4 in each groups in the asymmetrical one. There are several type of equation, and as we will briefly sum-up, all alternate type leads to a simplification of the proofs, in Section 2.6, page 35, we will present further optimizations.

Weaker types of pairing equation

We have seen in the examples, that pairing product equations come into two forms, the quadratic ones, and the linear ones. Linear pairing product equations drastically reduce the size of the proofs, for all the types of equations:

In the DLin instantiation, those linear equations are:

$$\langle \vec{\mathcal{A}}, \underline{\vec{\mathcal{Y}}} \rangle = t_T \ , \quad \langle \vec{a}, \underline{\vec{\mathcal{Y}}} \rangle = T \ , \quad \langle \underline{\vec{x}}, \vec{b} \rangle = t \ . \qquad \langle \underline{\vec{x}}, \vec{\mathcal{B}} \rangle = T \ ,$$

Whereas in the SXDH setting, those in $\mathbb{G}_1$ are:

$$\langle \underline{\vec{\mathcal{X}}}, \vec{\mathcal{B}} \rangle = t_T \ , \quad \langle \underline{\vec{y}}, \vec{\mathcal{A}} \rangle = T \ , \quad \langle \underline{\vec{x}}, \vec{b} \rangle = t \ . \qquad \langle \vec{b}, \underline{\vec{\mathcal{X}}} \rangle = T \ ,$$

Those different simplifications help to reduce the size of the proofs, and so the number of exponentiations required.

Assumption: SXDH	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{Z}_p$
Variables $x \in \mathbb{Z}_p, X \in \mathbb{G}_1$	2	0	0
Variables $y \in \mathbb{Z}_p, Y \in \mathbb{G}_2$	0	2	0
Pairing-Product Equation:	4	4	0
$\vec{\mathcal{A}} \cdot \vec{\mathcal{Y}} = d_T$	2	0	0
$\vec{\mathcal{X}} \cdot \vec{\mathcal{B}} = d_T$	0	2	0
Multi-Scalar Equation in $\mathbb{G}_1$:	2	4	0
$\vec{\mathcal{A}} \cdot \vec{y} = d_1$	1	0	0
$\vec{\mathcal{X}} \cdot \vec{b} = d_1$	0	0	2
Multi-Scalar Equation in $\mathbb{G}_2$:	4	2	0
$\vec{x} \cdot \vec{\mathcal{B}} = d_2$	0	1	0
$\vec{a} \cdot \vec{\mathcal{Y}} = d_2$	0	0	2
Quadratic equations in $\mathbb{Z}_p$:	2	2	0
$\vec{a} \cdot \vec{y} = d$	0	0	1
$\vec{x} \cdot \vec{b} = d$	0	0	1

Assumption: DLin	$\mathbb{G}$	$\mathbb{Z}_p$
Variables $x \in \mathbb{Z}_p, Y \in \mathbb{G}$	2	0
Pairing-Product Equation:	9	0
$\vec{\mathcal{A}} \cdot \vec{\mathcal{Y}} = d_T$	3	0
Multi-Scalar Equation in $\mathbb{G}$:	9	0
$\vec{a} \cdot \vec{\mathcal{Y}} = d_1$	0	3
$\vec{x} \cdot \vec{B} = d_1$	2	0
Quadratic equations in $\mathbb{Z}_p$:	6	0
$\vec{x} \cdot \vec{b} = d$	0	2

2.5 Smooth Projective Hash Functions

In this section, we will explain how to instantiate a smooth projective hash functions for a language $\mathcal{L}$ composed of valid encryptions of M. This is the foundation of our methodology to build SPHF on a broaden set of languages presented in section 5.2, page 97.

2.5.1 On a Linear Encryption

In the following, we will denote $\mathsf{Lin}(\mathsf{pk}, M)$ the language of the linear encryptions $\mathbf{c}$ of the message M under the encryption key $\mathsf{pk} = (Y_1, Y_2)$. Clearly, for $M = 1_{\mathbb{G}}$, the language contains the linear tuples in basis (Y_1, Y_2, g). The SPHF system is defined by, for $\mathsf{pk} = (Y_1, Y_2)$ and $\mathbf{c} = (c_1 = Y_1^{r_1}, c_2 = Y_2^{r_2}, c_3 = g^{r_1+r_2} \times M)$

$$\mathsf{HashKG}(\mathsf{Lin}(\mathsf{pk}, M)) = \mathsf{hk} = (x_1, x_2, x_3) \xleftarrow{\$} \mathbb{Z}_p^3 \qquad \mathsf{Hash}(\mathsf{hk}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c}) = c_1^{x_1} c_2^{x_2} (c_3/M)^{x_3}$$
$$\mathsf{ProjKG}(\mathsf{hk}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c}) = \mathsf{hp} = (Y_1^{x_1} g^{x_3}, Y_2^{x_2} g^{x_3}) \qquad \mathsf{ProjHash}(\mathsf{hp}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c}, r) = \mathsf{hp}_1^{r_1} \mathsf{hp}_2^{r_2}$$

Theorem 2.5.1 *This Smooth Projective Hash Function is correct.*

Proof: With the above notations:

- $\mathsf{Hash}(\mathsf{hk}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c}) = c_1^{x_1} c_2^{x_2} (c_3/M)^{x_3} = Y_1^{r_1 x_1} Y_2^{r_2 x_2} g^{(r_1+r_2)x_3}$
- $\mathsf{ProjHash}(\mathsf{hp}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c}, r) = \mathsf{hp}_1^{r_1} \mathsf{hp}_2^{r_2} = (Y_1^{x_1} g^{x_3})^{r_1} (Y_2^{x_2} g^{x_3})^{r_2} = Y_1^{r_1 x_1} Y_2^{r_2 x_2} g^{(r_1+r_2)x_3}$

□

Theorem 2.5.2 *This Smooth Projective Hash Function is smooth.*

Proof: Let us show that from an information theoretic point of view, $v = \mathsf{Hash}(\mathsf{hk}, \mathcal{L}(\mathsf{pk}, M), \mathbf{c})$ is unpredictable, even knowing hp, when $\mathbf{c}$ is not a correct ciphertext: $\mathbf{c} = (c_1 = Y_1^{r_1}, c_2 = Y_2^{r_2}, c_3 = g^{r_3} \times M)$, for $r_3 \neq r_1 + r_2$. We recall that $\mathsf{Hash}(\mathsf{hk}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c}) = Y_1^{r_1 x_1} Y_2^{r_2 x_2} g^{r_3 x_3}$ and $\mathsf{hp} = (Y_1^{x_1} g^{x_3}, Y_2^{x_2} g^{x_3})$: If we denote $Y_1 = g^{y_1}$ and $Y_2 = g^{y_2}$, we have:

$$\begin{pmatrix} \log \mathsf{hp}_1 \\ \log \mathsf{hp}_2 \\ \log v \end{pmatrix} = \begin{pmatrix} y_1 & 0 & 1 \\ 0 & y_2 & 1 \\ y_1 r_1 & y_2 r_2 & r_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

The determinant of this matrix is $y_1 y_2 (r_3 - r_1 - r_2)$, which is non-zero if $\mathbf{c}$ does not belong to the language ($r_3 \neq r_1 + r_2$). So v is independent from hp and $\mathbf{c}$. □

Theorem 2.5.3 *This Smooth Projective Hash Function is pseudo-random under the* DLin *assumption (the semantic security of the Linear encryption).*

Proof: As shown above, when $\mathbf{c}$ encrypts $M' \neq M$, then the distributions

$$\mathcal{D}_1 = \{\mathsf{Lin}(\mathsf{pk}, M), \mathbf{c} = \mathcal{E}_{\mathsf{pk}}(M'), \mathsf{hp}, v \xleftarrow{\$} \mathbb{G}\} \quad \mathcal{D}_2 = \{\mathsf{Lin}(\mathsf{pk}, M), \mathbf{c} = \mathcal{E}_{\mathsf{pk}}(M'), \mathsf{hp}, v = \mathsf{Hash}(\mathsf{hk}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c})\}$$

are perfectly indistinguishable. Under the semantic security of the Linear encryption, $\mathcal{E}_{\mathsf{pk}}(M)$ and $\mathcal{E}_{\mathsf{pk}}(M')$ are computationally indistinguishable, and so are the distributions

$$\begin{aligned} \mathcal{D}_0 &= \{\mathsf{Lin}(\mathsf{pk}, M), \mathbf{c} = \mathcal{E}_{\mathsf{pk}}(M), \mathsf{hp}, v \xleftarrow{\$} \mathbb{G}\} \\ \mathcal{D}_1 &= \{\mathsf{Lin}(\mathsf{pk}, M), \mathbf{c} = \mathcal{E}_{\mathsf{pk}}(M'), \mathsf{hp}, v \xleftarrow{\$} \mathbb{G}\} \end{aligned}$$

and the distributions

$$\begin{aligned} \mathcal{D}_2 &= \{\mathsf{Lin}(\mathsf{pk}, M), \mathbf{c} = \mathcal{E}_{\mathsf{pk}}(M'), \mathsf{hp}, v = \mathsf{Hash}(\mathsf{hk}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c})\} \\ \mathcal{D}_3 &= \{\mathsf{Lin}(\mathsf{pk}, M), \mathbf{c} = \mathcal{E}_{\mathsf{pk}}(M), \mathsf{hp}, v = \mathsf{Hash}(\mathsf{hk}, \mathsf{Lin}(\mathsf{pk}, M), \mathbf{c})\} \end{aligned}$$

As a consequence, $\mathcal{D}_0$ and $\mathcal{D}_3$ are computationally indistinguishable, which proves the result. □

2.5.2 On an ElGamal Encryption

In the following, we will denote $\mathsf{EG}(\mathsf{pk}, M)$ the language of ElGamal encryptions C of the message M under the encryption key $\mathsf{pk} = u$. Clearly, for $M = 1_{\mathbb{G}}$, the language contains the Diffie Hellman pairs in basis (u, g_1). The SPHF system is defined by, for $\mathsf{pk} = u$ and $C = (c_1 = u^r, c_2 = g_1^r \times M)$

$$\begin{array}{ll} \mathsf{HashKG}(\mathsf{EG}(\mathsf{pk}, M)) = \mathsf{hk} = (x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2 & \mathsf{Hash}(\mathsf{hk}, \mathsf{EG}(\mathsf{pk}, M), C) = c_1^{x_1} (c_2/M)^{x_2} \\ \mathsf{ProjKG}(\mathsf{hk}, \mathsf{EG}(\mathsf{pk}, M), C) = \mathsf{hp} = (u^{x_1} g_1^{x_2}) & \mathsf{ProjHash}(\mathsf{hp}, \mathsf{EG}(\mathsf{pk}, M), C, r) = \mathsf{hp}^r \end{array}$$

The security of this SPHF can be proven like before. With $M = 1_{\mathbb{G}}$, this boils down to check that we have a Diffie-Hellman tuple

2.6 New Results on Standard Primitives

In this section, we present several useful results on classical tools we discovered throughout this thesis. While not being crucial to understand our protocols, they allow many optimizations, allowing sometimes huge improvement in the final results, and they may be used directly in a lot of previous papers to improve their efficiency without decreasing their security.

Even if Groth-Sahai proofs have presented an impressive improvement in the efficiency of Witness-Indistinguishable, Zero-Knowledge proofs in the Standard Model, they still require a huge amount of computation. Think to what happens when we have to verify n equations sequentially, or simply if we have an equation with a large number of variables. In those cases exponentiations are not the main concern, the verification required a lot of pairing computations which are really costful. So we are going to emphasize an optimization we proposed on the Groth-Sahai framework. It allows to improve the efficiency of verification, we propose a batching technique to allow a verifier to check several proofs simultaneously. We even show that this technique improve the verification efficiency of one single proof.

We are then going to focus around Waters signature / function, first we propose and prove an asymmetric version of the Waters signature, usable in asymmetric groups. We are then going to study a variation of Waters function, with the help of the methodology followed in [HK08], this let us show that Waters may be used on some non-binary alphabets.

We then focus on Cramer-Shoup to consider instantiations where we compute a single-hash for multiple encryptions, and some partial encryption. This will be primordial for our protocol in the UC framework. We then present a variant of the Linear Cramer-Shoup commitment, such that it becomes both extractable and equivocable. This approach is inspired by the UC-commitment proposed by Lindell in [Lin11], however we increased its efficiency. We do not claim our result is a UC-commitment, as our protocol has inherited a weakness from the original construction and so does not completely fulfill the UC functionality.

2.6.1 Batch Groth-Sahai

In [BFI+10], we proposed a way to batch the verification of several Groth-Sahai proofs. Our first goal was to consider the cases where we have to verify n similar equations, like when someone wants to verify a bunch of signatures. We followed the steps of [Fia90], [BGR98], and [FGHP09]. We decided to batch those expensive tasks all at once.

In order to do so, we used the *small exponent test* from [BGR98]. We picked small random exponents r_i, raised the ith-equation to power r_i and checked if the product of the left-hand sides of those randomized equations was equal to the product of the right ones. This induces a tiny soundness error (It was shown in [FGHP09] that it is bounded by $2^{-\ell}$, when r_i are ℓ-bits strings), but drastically improve the efficiency. We also follow simple rules, to avoid costly exponentiations in $\mathbb{G}_T$ by moving the exponent inside the pairing on the element in $\mathbb{G}_1$ when possible ([GPS08] explained that an exponentiation in $\mathbb{G}_2$ may be more costly).

1. **Move the exponent into the pairing:** $e(f_i, h_i)^{\delta_i} \to e(f_i^{\delta_i}, h_i)$
2. **Move the product into the pairing:** $\prod_{j=1}^{m} e(f_j^{\delta_j}, h_i) \to e\left(\prod_{j=1}^{m} f_j^{\delta_j}, h_i\right)$
3. **Switch two products:** $\prod_{j=1}^{m} e\left(f_j, \prod_{i=1}^{k} h_i^{\delta_{i,j}}\right) \leftrightarrow \prod_{i=1}^{k} e\left(\prod_{j=1}^{m} f_j^{\delta_{i,j}}, h_i\right)$

δ_i can be any exponent involved in the i-th equation, so of course the power r_i but also the exponent $\gamma_{i,k}$ associated with the quadratic pairing product, and public scalars in both multi-scalar multiplication equations and quadratic equations.

Batch Verification: We applied these batch-verification techniques to the verification equations for Groth-Sahai proofs, and obtained some nice improvements, one can see that even for $n = 1$, our verification technique provides some good results presented in Table 2.1, page 36.

In the following we are going to focus on DLin, and show how we obtain those results. A similar approach was done in SXDH, however as we will primarily focus on the DLin instantiations of our protocols in the whole thesis, so we will skip the details on how to obtain the results in SXDH.

	Naive computation	Batch computation
SXDH		
Pairing-product equation	$5\,m + 3\,n + 16$	$m + 2\,n + 8$
Multi-scalar multiplication equation in $\mathbb{G}_1$	$8\,m + 2\,n + 14$	$\min(2\,n + 9, 2\,m + n + 7)$
Quadratic equation	$8\,m + 8\,n + 12$	$2\min(m, n) + 8$
DLin		
Pairing-product equation	$12\,n + 27$	$3\,n + 6$
Multi-scalar multiplication equation	$9\,n + 12\,m + 27$	$3\,n + 3\,m + 6$
Quadratic equation	$18\,n + 24$	$3\,n + 6$

Table 2.1: Number of pairings per verification, where n and m stand for the number of variables.

Pairing-Product Equation

The verification relation of a proof $(\vec{\mathbf{d}}, \phi) \in \mathbb{G}^{n\times 3} \times \mathbb{G}^{3\times 3}$ for a pairing equation is the following:

$$\left[\iota(\vec{\mathcal{A}}) \overset{s}{\bullet} \vec{\mathbf{d}}\right] \odot \left[\vec{\mathbf{d}} \overset{s}{\bullet} \Gamma\vec{\mathbf{d}}\right] = \iota_T(t_T) \odot \left[\vec{\mathbf{u}} \overset{s}{\bullet} \phi\right]$$

For simplicity, we consider the squares of all $\mathbb{G}_T$ elements in both sides of the equation, this avoids the square roots introduced by the $\overset{s}{\bullet}$ operator. Writing $\Gamma\vec{\mathbf{d}}$ as $\left(\prod_{k=1}^{n} d_{k,j}^{\gamma_{i,k}}\right)_{\substack{1\le i\le n\\ 1\le j\le 3}}$ and replacing the bilinear product $\overset{s}{\bullet}$ by its definition, we get for the left-hand side:

$$\begin{pmatrix}
\prod_{i=1}^{n} e(d_{i,1}, \prod d_{k,1}^{\gamma_{i,k}})^2 & \prod_{i=1}^{n} e(d_{i,1}, \prod d_{k,2}^{\gamma_{i,k}}) e(d_{i,2}, \prod d_{k,1}^{\gamma_{i,k}}) & \prod_{i=1}^{n} e(\mathcal{A}_i, d_{i,1}) e(d_{i,1}, \prod d_{k,3}^{\gamma_{i,k}}) e(d_{i,3}, \prod d_{k,1}^{\gamma_{i,k}}) \\
\prod_{i=1}^{n} e(d_{i,2}, \prod d_{k,1}^{\gamma_{i,k}}) e(d_{i,1}, \prod d_{k,2}^{\gamma_{i,k}}) & \prod_{i=1}^{n} e(d_{i,2}, \prod d_{k,2}^{\gamma_{i,k}})^2 & \prod_{i=1}^{n} e(\mathcal{A}_i, d_{i,2}) e(d_{i,2}, \prod d_{k,3}^{\gamma_{i,k}}) e(d_{i,3}, \prod d_{k,2}^{\gamma_{i,k}}) \\
\prod_{i=1}^{n} e(\mathcal{A}_i, d_{i,1}) e(d_{i,3}, \prod d_{k,1}^{\gamma_{i,k}}) \cdot e(d_{i,1}, \prod d_{k,3}^{\gamma_{i,k}}) & \prod_{i=1}^{n} e(\mathcal{A}_i, d_{i,2}) e(d_{i,3}, \prod d_{k,2}^{\gamma_{i,k}}) \cdot e(d_{i,2}, \prod d_{k,3}^{\gamma_{i,k}}) & \prod_{i=1}^{n} e(\mathcal{A}_i, d_{i,3})^2 e(d_{i,3}, \prod d_{k,3}^{\gamma_{i,k}})^2
\end{pmatrix}$$

and for the right-hand side:

$$\begin{pmatrix}
\prod_{i=1}^{3} e(u_{i,1}, \phi_{i,1})^2 & \prod_{i=1}^{3} e(u_{i,1}, \phi_{i,2}) e(u_{i,2}, \phi_{i,1}) & \prod_{i=1}^{3} e(u_{i,1}, \phi_{i,3}) e(u_{i,3}, \phi_{i,1}) \\
\prod_{i=1}^{3} e(u_{i,2}, \phi_{i,1}) e(u_{i,1}, \phi_{i,2}) & \prod_{i=1}^{3} e(u_{i,2}, \phi_{i,2})^2 & \prod_{i=1}^{3} e(u_{i,2}, \phi_{i,3}) e(u_{i,3}, \phi_{i,2}) \\
\prod_{i=1}^{3} e(u_{i,3}, \phi_{i,1}) e(u_{i,1}, \phi_{i,3}) & \prod_{i=1}^{3} e(u_{i,3}, \phi_{i,2}) e(u_{i,2}, \phi_{i,3}) & t_T^2 \prod_{i=1}^{3} e(u_{i,3}, \phi_{i,3})^2
\end{pmatrix}$$

Taking each element $M_{i,j}$ of the equation to the power of $r_{i,j}$, multiplying everything, and regrouping pairings, we get the following for the left-hand side:

$$\prod_{i=1}^{n} e\big(d_{i,1},\ \mathcal{A}_i^{r_{1,3}+r_{3,1}} \prod d_{k,1}^{2\cdot\gamma_{i,k}\cdot r_{1,1}} d_{k,2}^{\gamma_{i,k}(r_{1,2}+r_{2,1})} d_{k,3}^{\gamma_{i,k}(r_{1,3}+r_{3,1})}\big) \cdot$$
$$e\big(d_{i,2},\ \mathcal{A}_i^{r_{2,3}+r_{3,2}} \prod d_{k,1}^{\gamma_{i,k}(r_{1,2}+r_{2,1})} d_{k,2}^{2\cdot\gamma_{i,k}\cdot r_{2,2}} d_{k,3}^{\gamma_{i,k}(r_{2,3}+r_{3,2})}\big) \cdot$$
$$e\big(d_{i,3},\ \mathcal{A}_i^{2\cdot r_{3,3}} \prod d_{k,1}^{\gamma_{i,k}(r_{1,3}+r_{3,1})} d_{k,2}^{\gamma_{i,k}(r_{2,3}+r_{3,2})} d_{k,3}^{2\cdot\gamma_{i,k}\cdot r_{3,3}}\big)$$

and for the right-hand side:

$$\prod_{i=1}^{3} e\big(u_{i,1},\ \phi_{i,1}^{2\cdot r_{1,1}} \phi_{i,2}^{r_{1,2}+r_{2,1}} \phi_{i,3}^{r_{1,3}+r_{3,1}}\big) \cdot e\big(u_{i,2},\ \phi_{i,1}^{r_{1,2}+r_{2,1}} \phi_{i,2}^{2\cdot r_{2,2}} \phi_{i,3}^{r_{2,3}+r_{3,2}}\big)$$
$$\cdot\, e\big(u_{i,3},\ \phi_{i,1}^{r_{1,3}+r_{3,1}} \phi_{i,2}^{r_{2,3}+r_{3,2}} \phi_{i,3}^{2\cdot r_{3,3}}\big) \cdot t_T^{2 r_{3,3}}$$

By definition, we also have $u_{1,2} = u_{2,1} = 1$, and $u_{1,3} = u_{2,3}$ this simplifies to:

$$e\big(u_{1,1},\ \phi_{1,1}^{2\cdot r_{1,1}}\phi_{1,2}^{r_{1,2}+r_{2,1}}\phi_{1,3}^{r_{1,3}+r_{3,1}}\big)\cdot e\big(u_{1,3},\ (\phi_{1,1}\phi_{2,1})^{r_{1,3}+r_{3,1}}(\phi_{1,2}\phi_{2,2})^{r_{2,3}+r_{3,2}}(\phi_{1,3}\phi_{2,3})^{2\cdot r_{3,3}}\big)$$
$$e\big(u_{2,2},\ \phi_{2,1}^{r_{1,2}+r_{2,1}}\phi_{2,2}^{2\cdot r_{2,2}}\phi_{2,3}^{r_{2,3}+r_{3,2}}\big)\cdot e\big(u_{3,1},\ \phi_{3,1}^{2\cdot r_{1,1}}\phi_{3,2}^{r_{1,2}+r_{2,1}}\phi_{3,3}^{r_{1,3}+r_{3,1}}\big)\cdot e\big(u_{3,2},\ \phi_{3,1}^{r_{1,2}+r_{2,1}}\phi_{3,2}^{2\cdot r_{2,2}}\phi_{3,3}^{r_{2,3}+r_{3,2}}\big)\cdot$$
$$e\big(u_{3,3},\ \phi_{3,1}^{r_{1,3}+r_{3,1}}\phi_{3,2}^{r_{2,3}+r_{3,2}}\phi_{3,3}^{2\cdot r_{3,3}}\big)\cdot t_T^{2r_{3,3}}$$

In total we reduced the number of pairings from $12\,n + 27$ to $3\,n + 6$ pairings at the expense of adding $9\,n^2 + 3\,n$ exponentiations in $\mathbb{G}$ and one exponentiation in $\mathbb{G}_T$.

Multi-Scalar Multiplication Equation

The verification equation of a proof $(\vec{\mathbf{c}}, \vec{\mathbf{d}}, \phi)$, with $\phi \in \mathbb{G}^{3\times 3}$, of a multi-scalar multiplication equation is the following:

$$\Big[\iota'(\vec{a}) \overset{s}{\bullet} \vec{\mathbf{d}}\Big] \odot \Big[\vec{\mathbf{c}} \overset{s}{\bullet} \iota(\vec{\mathcal{B}})\Big] \odot \Big[\vec{\mathbf{c}} \overset{s}{\bullet} \Gamma\vec{\mathbf{d}}\Big] = \hat{\imath}_T(\mathcal{T}) \odot \Big[\mathbf{u} \overset{s}{\bullet} \phi\Big]$$

The left-hand side of the equation is thus

$$\Big[\big(u_{3,1}^{a_i}, u_{3,2}^{a_i}, (u_{3,3}g)^{a_i}\big)_{1\le i\le n} \overset{s}{\bullet} \big(d_{i,1}, d_{i,2}, d_{i,3}\big)_{1\le i\le n}\Big] \odot \Big[\big(c_{i,1}, c_{i,2}, c_{i,3}\big)_{1\le i\le m} \overset{s}{\bullet} \big(1, 1, \mathcal{B}_i\big)_{1\le i\le m}\Big]$$
$$\odot \Big[\big(c_{i,1}, c_{i,2}, c_{i,3}\big)_{1\le i\le m} \overset{s}{\bullet} \big((\Gamma\vec{\mathbf{d}})_{i,1}, (\Gamma\vec{\mathbf{d}})_{i,2}, (\Gamma\vec{\mathbf{d}})_{i,3}\big)_{1\le i\le m}\Big]$$

Considering the squares of all matrix elements, this is written out as

$$\begin{pmatrix} \prod_{i=1}^n e\big(u_{3,1}^{a_i}, d_{i,1}\big)^2 & \prod_{i=1}^n e\big(u_{3,1}^{a_i}, d_{i,2}\big)e\big(u_{3,2}^{a_i}, d_{i,1}\big) & \prod_{i=1}^n e\big(u_{3,1}^{a_i}, d_{i,3}\big)e\big((u_{3,3}g)^{a_i}, d_{i,1}\big) \\ \prod_{i=1}^n e\big(u_{3,2}^{a_i}, d_{i,1}\big)e\big(u_{3,1}^{a_i}, d_{i,2}\big) & \prod_{i=1}^n e\big(u_{3,2}^{a_i}, d_{i,2}\big)^2 & \prod_{i=1}^n e\big(u_{3,2}^{a_i}, d_{i,3}\big)e\big((u_{3,3}g)^{a_i}, d_{i,2}\big) \\ \prod_{i=1}^n e\big((u_{3,3}g)^{a_i}, d_{i,1}\big)e\big(u_{3,1}^{a_i}, d_{i,3}\big) & \prod_{i=1}^n e\big((u_{3,3}g)^{a_i}, d_{i,2}\big)e\big(u_{3,2}^{a_i}, d_{i,3}\big) & \prod_{i=1}^n e\big((u_{3,3}g)^{a_i}, d_{i,3}\big)^2 \end{pmatrix}$$
$$\odot \begin{pmatrix} 1 & 1 & \prod_{i=1}^m e(c_{i,1}, \mathcal{B}_i) \\ 1 & 1 & \prod_{i=1}^m e(c_{i,2}, \mathcal{B}_i) \\ \prod_{i=1}^m e(c_{i,1}, \mathcal{B}_i) & \prod_{i=1}^m e(c_{i,2}, \mathcal{B}_i) & \prod_{i=1}^m e(c_{i,3}, \mathcal{B}_i) \end{pmatrix}$$
$$\odot \begin{pmatrix} \prod_{i=1}^m e(c_{i,1}, \prod d_{k,1}^{\gamma_{i,k}})^2 & \prod_{i=1}^m e(c_{i,1}, \prod d_{k,2}^{\gamma_{i,k}})e(c_{i,2}, \prod d_{k,1}^{\gamma_{i,k}}) & \prod_{i=1}^m e(c_{i,1}, \prod d_{k,3}^{\gamma_{i,k}})e(c_{i,3}, \prod d_{k,1}^{\gamma_{i,k}}) \\ \prod_{i=1}^m e(c_{i,2}, \prod d_{k,1}^{\gamma_{i,k}})e(c_{i,1}, \prod d_{k,2}^{\gamma_{i,k}}) & \prod_{i=1}^m e(c_{i,2}, \prod d_{k,2}^{\gamma_{i,k}})^2 & \prod_{i=1}^m e(c_{i,2}, \prod d_{k,3}^{\gamma_{i,k}})e(c_{i,3}, \prod d_{k,2}^{\gamma_{i,k}}) \\ \prod_{i=1}^m e(c_{i,3}, \prod d_{k,1}^{\gamma_{i,k}})e(c_{i,1}, \prod d_{k,3}^{\gamma_{i,k}}) & \prod_{i=1}^m e(c_{i,3}, \prod d_{k,2}^{\gamma_{i,k}})e(c_{i,2}, \prod d_{k,3}^{\gamma_{i,k}}) & \prod_{i=1}^m e(c_{i,3}, \prod d_{k,3}^{\gamma_{i,k}})^2 \end{pmatrix}$$

Using the batching technique, we get the following left-hand side:

$$\prod_{i=1}^n e\big(u_{3,1}^{2\cdot a_i\cdot r_{1,1}}u_{3,2}^{a_i(r_{1,2}+r_{2,1})}(u_{3,3}g)^{a_i\cdot(r_{1,3}+r_{3,1})},\ d_{i,1}\big)\cdot e\big(u_{3,1}^{a_i(r_{1,2}+r_{2,1})}u_{3,2}^{2\cdot a_i\cdot r_{2,2}}(u_{3,3}g)^{a_i\cdot(r_{2,3}+r_{3,2})},\ d_{i,2}\big)$$
$$\cdot e\big(u_{3,1}^{a_i(r_{1,3}+r_{3,1})}u_{3,2}^{a_i(r_{2,3}+r_{3,2})}(u_{3,3}g)^{2\cdot a_i\cdot r_{3,3}},\ d_{i,3}\big)$$
$$\cdot \prod_{i=1}^m e\big(c_{i,1}, \mathcal{B}_i^{r_{1,3}+r_{3,1}}(\prod d_{k,1}^{\gamma_{i,k}})^{2\cdot r_{1,1}}(\prod d_{k,2}^{\gamma_{i,k}})^{r_{1,2}+r_{2,1}}(\prod d_{k,3}^{\gamma_{i,k}})^{r_{1,3}+r_{3,1}}\big)$$
$$\cdot e\big(c_{i,2}, \mathcal{B}_i^{r_{2,3}+r_{3,2}}(\prod d_{k,1}^{\gamma_{i,k}})^{r_{1,2}+r_{2,1}}(\prod d_{k,2}^{\gamma_{i,k}})^{2\cdot r_{2,2}}(\prod d_{k,3}^{\gamma_{i,k}})^{r_{2,3}+r_{3,2}}\big)$$
$$\cdot e\big(c_{i,3}, \mathcal{B}_i^{2\cdot r_{3,3}}(\prod d_{k,1}^{\gamma_{i,k}})^{r_{3,1}+r_{1,3}}(\prod d_{k,2}^{\gamma_{i,k}})^{r_{3,2}+r_{2,3}}(\prod d_{k,3}^{\gamma_{i,k}})^{2\cdot r_{3,3}}\big)$$

While the right-hand side is similar to the previous case.

In total, we reduced from $9\,n + 12\,m + 27$ to $3\,n + 3\,m + 6$ pairings.

Quadratic Equation

The verification of a proof $(\vec{\mathbf{c}}, \phi) \in \mathbb{G}^{n\times 3} \times \mathbb{G}^{2\times 3}$ consists in checking the following:

$$\big[\vec{\mathbf{c}} \overset{s}{\bullet} (\iota'(\vec{b}))\big] \odot \big[\vec{\mathbf{c}} \overset{s}{\bullet} \Gamma\vec{\mathbf{c}}\big] = \iota'_T(t) \odot \big[\mathbf{v} \overset{s}{\bullet} \phi\big]\ .$$

We develop the left-hand side of the equation:

$$\left[\left(c_{i,1},c_{i,2},c_{i,3}\right)_{1\leq i\leq n} \overset{s}{\bullet} \left(u_{3,1}^{b_i},u_{3,2}^{b_i},\left(u_{3,3}g\right)^{b_i}\right)_{1\leq i\leq n}\right] \odot \left[\left(c_{i,1},c_{i,2},c_{i,3}\right)_{1\leq i\leq n} \overset{s}{\bullet} \left((\Gamma\vec{c})_{i,1},(\Gamma\vec{c})_{i,2},(\Gamma\vec{c})_{i,3}\right)_{1\leq i\leq n}\right]$$

Once squared, this becomes:

$$\begin{pmatrix} \prod_{i=1}^n e(c_{i,1},u_{3,1}^{b_i})^2 & \prod_{i=1}^n e(c_{i,1},u_{3,2}^{b_i})e(c_{i,2},u_{3,1}^{b_i}) & \prod_{i=1}^n e(c_{i,1},(u_{3,3}g)^{b_i})e(c_{i,3},u_{3,1}^{b_i}) \\ \prod_{i=1}^n e(c_{i,2},u_{3,1}^{b_i})e(c_{i,1},u_{3,2}^{b_i}) & \prod_{i=1}^n e(c_{i,2},u_{3,2}^{b_i})^2 & \prod_{i=1}^n e(c_{i,2},(u_{3,3}g)^{b_i})e(c_{i,3},u_{3,2}^{b_i}) \\ \prod_{i=1}^n e(c_{i,3},u_{3,1}^{b_i})e(c_{i,1},(u_{3,3}g)^{b_i}) & \prod_{i=1}^n e(c_{i,3},u_{3,2}^{b_i})e(c_{i,2},(u_{3,3}g)^{b_i}) & \prod_{i=1}^n e(c_{i,3},(u_{3,3}g)^{b_i})^2 \end{pmatrix}$$

$$\odot \begin{pmatrix} \prod_{i=1}^n e(c_{i,1},\prod c_{k,1}^{\gamma_{i,k}})^2 & \prod_{i=1}^n e(c_{i,1},\prod c_{k,2}^{\gamma_{i,k}})e(c_{i,2},\prod c_{k,1}^{\gamma_{i,k}}) & \prod_{i=1}^n e(c_{i,1},\prod c_{k,3}^{\gamma_{i,k}})e(c_{i,3},\prod c_{k,1}^{\gamma_{i,k}}) \\ \prod_{i=1}^n e(c_{i,2},\prod c_{k,1}^{\gamma_{i,k}})e(c_{i,1},\prod c_{k,2}^{\gamma_{i,k}}) & \prod_{i=1}^n e(c_{i,2},\prod c_{k,2}^{\gamma_{i,k}})^2 & \prod_{i=1}^n e(c_{i,2},\prod c_{k,3}^{\gamma_{i,k}})e(c_{i,3},\prod c_{k,2}^{\gamma_{i,k}}) \\ \prod_{i=1}^n e(c_{i,3},\prod c_{k,1}^{\gamma_{i,k}})e(c_{i,1},\prod c_{k,3}^{\gamma_{i,k}}) & \prod_{i=1}^n e(c_{i,3},\prod c_{k,2}^{\gamma_{i,k}})e(c_{i,2},\prod c_{k,3}^{\gamma_{i,k}}) & \prod_{i=1}^n e(c_{i,3},\prod c_{k,3}^{\gamma_{i,k}})^2 \end{pmatrix}$$

Multiplying all matrix elements after taking them to a random power, we obtain the following:

$$\prod_{i=1}^n e\Big(c_{i,1},\, u_{3,1}^{2b_i\cdot r_{1,1}}u_{3,2}^{b_i\cdot(r_{1,2}+r_{2,1})}(u_{3,3}g)^{b_i\cdot(r_{1,3}+r_{3,1})}\prod\left(c_{k,1}^{2\gamma_{i,k}\cdot r_{1,1}}c_{k,2}^{\gamma_{i,k}\cdot(r_{1,2}+r_{2,1})}c_{k,3}^{\gamma_{i,k}\cdot(r_{1,3}+r_{3,1})}\right)\Big)$$
$$e\Big(c_{i,2},\, u_{3,1}^{b_i\cdot(r_{1,2}+r_{2,1})}u_{3,2}^{2b_i\cdot r_{2,2}\cdot}(u_{3,3}g)^{b_i\cdot(r_{2,3}+r_{3,2})}\prod\left(c_{k,1}^{\gamma_{i,k}\cdot(r_{1,2}+r_{2,1})}c_{k,2}^{2\gamma_{i,k}\cdot r_{2,2}}c_{k,3}^{\gamma_{i,k}\cdot(r_{2,3}+r_{3,2})}\right)\Big)$$
$$e\Big(c_{i,3},\, u_{3,1}^{b_i\cdot(r_{1,3}+r_{3,1})}u_{3,2}^{b_i\cdot(r_{2,3}+r_{3,2})}(u_{3,3}g)^{2b_i\cdot r_{3,3}}\prod\left(c_{k,1}^{\gamma_{i,k}\cdot(r_{1,3}+r_{3,1})}c_{k,2}^{\gamma_{i,k}\cdot(r_{2,3}+r_{3,2})}c_{k,3}^{2\gamma_{i,k}\cdot r_{3,3}}\right)\Big)$$

On the right hand side we have something similar to the previous cases, therefore we have an overall reduction from $18\,n+24$ pairings to $3\,n+6$ pairings.

Those results can further be improved on linear equations, the $\gamma_{i,j}$ in the left-hand side being all null in this case.

2.6.2 Application to Existing Protocols

We applied those batching techniques to classical schemes.

The first scheme we consider was proposed by Groth in 2007 [Gro07]. It is a constant-size group signature scheme whose security can be proved in the standard model. (Group signature are defined later, in section 3.1, page 55) For illustrative purposes, we concentrate on the (simpler) variant of the scheme. Even this variant does not achieve satisfactory efficiency: the verification of a signature requires the computation of 68 expensive pairing operations. In Section 2.6.2, page 38, we propose an improved verification procedure in which the total number of bilinear map evaluations drops to 11. In addition, if $n \geq 2$ signatures (for the same group) have to be verified at once, we manage to decrease this number from $11\,n$ to $4\,n+7$.

In Section 2.6.2, page 40, we study the *P-signature* scheme[1] proposed by Belenkiy, Chase, Kohlweiss and Lysyanskaya [BCKL08]. Since anonymous credentials are an immediate consequence of P-signatures, we thereby apply our techniques to privacy-preserving authentication mechanisms. Belenkiy *et al.* proposed two instantiations of their protocol (based on SXDH and DLin). They evaluated that the verification of the proof of possession of a signature would involve respectively 68 and 128 pairing evaluations. We show that this can be reduced to 15 and 12, respectively. Moreover, the number of pairing operations required to verify $n \geq 2$ signatures is reduced to $2\,n+13$ and $3\,n+9$, respectively, by using our techniques which is a huge improvement when compared with the original cost of $68\,n$ and $128\,n$.

Application 1: Groth's Group Signatures

Description: We demonstrate our techniques by applying them to one of the most practical group signature schemes in the standard model to date: Groth's construction [Gro07]. Groth proposed a methodology of transforming *certified signatures* [BFPW07] that respect a certain structure into group signatures using Groth-Sahai NIWI proofs:

- a member picks keys for a certified signature scheme and asks the issuer to certify her public verification key for the signature scheme;

[1] A *P-signature* scheme is a digital signature scheme with an additional non-interactive proof of signature possession.

- to produce a group signature, the member will make a certified signature, encrypt it and then use NIWI proofs to demonstrate that the ciphertext contains a valid certified signature.

Groth proposed an efficient certified signature scheme based on the so called q-**U** assumption (see [Gro07] for details). In the simple version of the scheme, the issuer's public key is a triple $(f, h, T) \in \mathbb{G}^2 \times \mathbb{G}_T$ (and its private key is $z \in \mathbb{G}$ such that $e(f, z) = T$) and the certificate of a group member with public key $v = g^x \in \mathbb{G}$ is a pair (a, b) satisfying $e(a, vh)\, e(f, b) = T$. To sign a message $m \in \mathbb{Z}_p$, the group member first computes a weak Boneh-Boyen signature [BB08] $\sigma = g^{1/(x+m)}$ using her private key x; then she forms Groth-Sahai commitments $\mathbf{d}_v$, $\mathbf{d}_b$ and $\mathbf{d}_\sigma$ to the group elements v, b and σ, resp., and makes a proof that they satisfy the following:

$$e(a, vh)\, e(f, b) = T \quad \text{and} \quad e(\sigma, g^m v) = e(g, g)$$

The fact that a is given in the clear is not a problem since the certificate is malleable, so the group member can unlinkably re-randomize it each time she signs a message. A group signature is thus of the form $(a, \mathbf{d}_b, \mathbf{d}_v, \mathbf{d}_\sigma, \psi, \phi)$, where ψ and ϕ denote the Groth-Sahai proofs for the two equations, respectively.

We first instantiate our generic batch construction to verify a single signature more efficiently and then show how to verify multiple signatures at once.

1st Equation: The first equation is a linear pairing equation, so we can use an improved version of our pairing equation batch. If we consider an equation $\langle \vec{\mathcal{A}}, \vec{\mathcal{Y}} \rangle = t_T$ we obtain

$$\prod_{i=1}^{n} e(\mathcal{A}_i, d_{i,1}^{s_1} d_{i,2}^{s_2} d_{i,3}^{s_3}) = t_T^{s_3}\, e(u_{11}, \psi_1^{s_1})\, e(u_{13}, (\psi_1\psi_2)^{s_3})\, e(u_{22}, \psi_2^{s_2})\, e(u_{31}, \psi_3^{s_1})\, e(u_{32}, \psi_3^{s_2})\, e(u_{33}, \psi_3^{s_3})\ .$$

So, in our case, after some more optimization (shifting $e(a, h^{-1}))^{s_3}$ to the left-hand side of the equation) we obtain:

$$\begin{aligned} &e(d_{v,1}^{s_1} d_{v,2}^{s_2} (d_{v,3} h)^{s_3},\, a)\, e(d_{b,1}^{s_1} d_{b,2}^{s_2} d_{b,3}^{s_3},\, f) \;= \\ &\qquad = T^{s_3}\, e(u_{11}, \psi_1^{s_1})\, e(u_{13}, (\psi_1\psi_2)^{s_3})\, e(u_{22}, \psi_2^{s_2})\, e(u_{31}, \psi_3^{s_1})\, e(u_{32}, \psi_3^{s_2})\, e(u_{33}, \psi_3^{s_3})\ . \end{aligned}$$

2nd Equation: We get:

$$\begin{aligned} &e\big(d_{\sigma 1},\ (g^m d_{v3})^{(r_{13}+r_{31})} d_{v1}^{2\cdot r_{11}} d_{v2}^{(r_{12}+r_{21})}\big)\, e\big(d_{\sigma 2},\ (g^m d_{v3})^{(r_{23}+r_{32})} d_{v1}^{(r_{12}+r_{21})} d_{v2}^{2\cdot r_{22}}\big) \\ &\cdot e\big(d_{\sigma 3},\ (g^m d_{v3})^{2\cdot r_{33}} d_{v1}^{(r_{13}+r_{31})} d_{v2}^{(r_{23}+r_{32})}\big) \;= \\ &\qquad = e\big(u_{11},\ \phi_{11}^{2\cdot r_{11}} \phi_{12}^{r_{12}+r_{21}} \phi_{13}^{r_{13}+r_{31}}\big)\, e\big(u_{13},\ (\phi_{11}\phi_{21})^{r_{13}+r_{31}} (\phi_{12}\phi_{22})^{r_{23}+r_{32}} (\phi_{13}\phi_{23})^{2\cdot r_{33}}\big) \\ &\qquad\quad \cdot e\big(u_{22},\ \phi_{21}^{r_{12}+r_{21}} \phi_{22}^{2\cdot r_{22}} \phi_{23}^{r_{23}+r_{32}}\big)\, e\big(u_{31},\ \phi_{31}^{2\cdot r_{11}} \phi_{32}^{r_{12}+r_{21}} \phi_{33}^{r_{13}+r_{31}}\big) \\ &\qquad\quad \cdot e\big(u_{32},\ \phi_{31}^{r_{12}+r_{21}} \phi_{32}^{2\cdot r_{22}} \phi_{33}^{r_{23}+r_{32}}\big)\, e\big(u_{33},\ \phi_{31}^{r_{13}+r_{31}} \phi_{32}^{r_{23}+r_{32}} \phi_{33}^{2\cdot r_{33}}\big)\, e(g, g^{2 r_{33}})\ . \end{aligned}$$

Multiplying the two equations we get a single verification relation of the following form:

$$\begin{aligned} &e(d_{v,1}^{s_1} d_{v,2}^{s_2} (d_{v,3} h)^{s_3}, a)\, e(d_{b,1}^{s_1} d_{b,2}^{s_2} d_{b,3}^{s_3}, f) e\big(d_{\sigma 1},\ (g^m d_{v3})^{(r_{13}+r_{31})} d_{v1}^{2\cdot r_{11}} d_{v2}^{(r_{12}+r_{21})}\big) \\ &\cdot e\big(d_{\sigma 2},\ (g^m d_{v3})^{(r_{23}+r_{32})} d_{v1}^{(r_{12}+r_{21})} d_{v2}^{2\cdot r_{22}}\big)\, e\big(d_{\sigma 3},\ (g^m d_{v3})^{2\cdot r_{33}} d_{v1}^{(r_{13}+r_{31})} d_{v2}^{(r_{23}+r_{32})}\big) \;= \\ &\qquad = e\big(u_{11},\ \phi_{11}^{2\cdot r_{11}} \phi_{12}^{r_{12}+r_{21}} \phi_{13}^{r_{13}+r_{31}} \psi_1^{s_1}\big)\, e\big(u_{13},\ (\phi_{11}\phi_{21})^{r_{13}+r_{31}} (\phi_{12}\phi_{22})^{r_{23}+r_{32}} (\phi_{13}\phi_{23})^{2\cdot r_{33}} (\psi_1\psi_2)^{s_3}\big) \\ &\qquad\quad \cdot e\big(u_{22},\ \phi_{21}^{r_{12}+r_{21}} \phi_{22}^{2\cdot r_{22}} \phi_{23}^{r_{23}+r_{32}} \psi_2^{s_2}\big)\, e\big(u_{31},\ \phi_{31}^{2\cdot r_{11}} \phi_{32}^{r_{12}+r_{21}} \phi_{33}^{r_{13}+r_{31}} \psi_3^{s_1}\big) \\ &\qquad\quad \cdot e\big(u_{32},\ \phi_{31}^{r_{12}+r_{21}} \phi_{32}^{2\cdot r_{22}} \phi_{33}^{r_{23}+r_{32}} \psi_3^{s_2}\big)\, e\big(u_{33},\ \phi_{31}^{r_{13}+r_{31}} \phi_{32}^{r_{23}+r_{32}} \phi_{33}^{2\cdot r_{33}} \psi_3^{s_3}\big)\, \big(T^{s_3} e(g, g^{2 r_{33}})\big) \end{aligned}$$

Analysis: With no use of batching techniques, the verification of a single signature takes for the first equation 13 pairings and for the second 20 pairings for the left-hand side and 35 for its right-hand side. This is an overall of 68 pairing evaluations, compared to 11 for the batched verification, so even on a single signature we have a significant improvement.

Batching Several Group Signatures:

Consider the situation where we want to verify multiple group signatures at once. That is given a group public key $(f, h, T, u_{11}, u_{13}, u_{22}, u_{31}, u_{32}, u_{33})$ and n group signatures

$$\left(a^{(k)}, \mathbf{d}_b^{(k)}, \mathbf{d}_v^{(k)}, \mathbf{d}_\sigma^{(k)}, (\psi_i^{(k)})_{1\leq i\leq 3}, (\phi_{ij}^{(k)})_{1\leq i,j\leq 3}\right)$$

Using the same technique of taking each of the (new) equations to the power of some randomness and multiplying them, we can unify the pairings $e(\cdot, f)$ on the left-hand side and all pairings (which are of the form $e(u_{ij}, \cdot)$) on the right-hand side.

Instead of $11\,n$ pairings needed when checking each equation, the batched version only requires $4\,n+7$ pairings.

Application 2: Belenkiy-Chase-Kohlweiss-Lysyanskaya's P-signatures

Description: Belenkiy *et al.* formalized in [BCKL08] digital signature schemes with an additional non-interactive proof of signature possession that they called *P-signature schemes*. They proposed two constructions: the first one relies on the weak Boneh-Boyen signature scheme [BB08] while the second one is inspired by its full version.

Since Belenkiy *et al.*'s first scheme relies on a rather strong assumption, we consider only their second proposal: a signature σ on a message $m \in \mathbb{Z}_p$ is a triple $\sigma = (C_1, C_2, C_3) \in \mathbb{G}^3$ such that $e(C_1, vh^m C_2) = e(g, h)$ and $e(f, C_2) = e(C_3, w)$, where f, g, h are (public) generators of $\mathbb{G}$ and $v, w \in \mathbb{G}$ are parts of the signer's public key. To prove the possession of such a signature, a prover forms the Groth-Sahai commitments $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$ for the group elements $C_1, M_1 = f^m, C_3$ and $\mathbf{d_1}$ and $\mathbf{d_2}$ for the group elements $M_2 = h^m$ and C_2 in $\mathbb{G}$ and provides a proof that they satisfy:

$$e(C_1, vM_2C_2) = e(g, h), \quad e(f, C_2) = e(C_3, w) \quad \text{and} \quad e(f, M_2) = e(M_1, h)$$

SXDH **Instantiation:** In [BCKL08], the authors evaluated that the verification of the proof in the SXDH instantiation requires the computation of 68 pairings. We have shown that it can be reduced to 15.

DLin **Instantiation:** As with the previous scheme, the last two pairing-product equations from are actually linear pairing-product equations. We denote the Groth-Sahai commitments for the group elements $C_1, C_2, C_3, M_1 = f^m, M_2 = h^m$ in $\mathbb{G}$ by $\mathbf{d_1}$,$\mathbf{d_2}$, $\mathbf{d_3}$, $\mathbf{d_4}$ and $\mathbf{d_5}$ (respectively) and ϕ, ψ and θ the proofs that they satisfy the first, the second and the third equation (respectively).

For the first equation, we get:

$$\begin{aligned}
&e\big(d_{1,1},\ (v\,d_{2,3}\,d_{5,3})^{r_{1,3}+r_{3,1}}(d_{2,1}\,d_{5,1})^{2r_{1,1}}(d_{2,2}\,d_{5,2})^{(r_{1,2}+r_{2,1})}\big)\\
&\qquad\qquad\cdot e\big(d_{1,2},\ (v\,d_{2,3}\,d_{5,3})^{r_{2,3}+r_{3,2}}(d_{2,1}\,d_{5,1})^{r_{1,2}+r_{2,1}}(d_{2,2}\,d_{5,2})^{2r_{2,2}}\big)\\
&\qquad\qquad\cdot e\big(d_{1,3},\ (v\,d_{2,3}\,d_{5,3})^{2r_{3,3}}(d_{2,1}\,d_{5,1})^{r_{1,3}+r_{3,1}}(d_{2,2}\,d_{5,2})^{r_{2,3}+r_{3,2}}\big)\\
&\quad= e\big(u_{1,1},\ \phi_{1,1}^{2\cdot r_{1,1}}\phi_{1,2}^{r_{1,2}+r_{2,1}}\phi_{1,3}^{r_{1,3}+r_{3,1}}\big)\cdot e\big(u_{1,3},\ (\phi_{1,1}\phi_{2,1})^{r_{1,3}+r_{3,1}}(\phi_{1,2}\phi_{2,2})^{r_{2,3}+r_{3,2}}(\phi_{1,3}\phi_{2,3})^{2\cdot r_{3,3}}\big)\\
&e\big(u_{2,2},\ \phi_{2,1}^{r_{1,2}+r_{2,1}}\phi_{2,2}^{2\cdot r_{2,2}}\phi_{2,3}^{r_{2,3}+r_{3,2}}\big)\cdot e\big(u_{3,1},\ \phi_{3,1}^{2\cdot r_{1,1}}\phi_{3,2}^{r_{1,2}+r_{2,1}}\phi_{3,3}^{r_{1,3}+r_{3,1}}\big)\cdot e\big(u_{3,2},\ \phi_{3,1}^{r_{1,2}+r_{2,1}}\phi_{3,2}^{2\cdot r_{2,2}}\phi_{3,3}^{r_{2,3}+r_{3,2}}\big)\cdot\\
&\qquad\qquad e\big(u_{3,3},\ \phi_{3,1}^{r_{1,3}+r_{3,1}}\phi_{3,2}^{r_{2,3}+r_{3,2}}\phi_{3,3}^{2\cdot r_{3,3}}\big)\cdot e(g,g)^{2r_{3,3}}\ .
\end{aligned}$$

With the substitution $\vec{\mathcal{A}} = \begin{pmatrix} f \\ w^{-1} \end{pmatrix}, \vec{\mathbf{d}} = \begin{pmatrix} \mathbf{d_2} \\ \mathbf{d_3} \end{pmatrix}$, and $t_T = 1$ and $\vec{\mathcal{A}} = \begin{pmatrix} f \\ h^{-1} \end{pmatrix}, \vec{\mathbf{d}} = \begin{pmatrix} \mathbf{d_5} \\ \mathbf{d_4} \end{pmatrix}$, and $t_T = 1$ (respectively) in (2.6.2), we obtain the second and third equation. Once the three equations multiplied, we obtain:

$$
\begin{aligned}
e\big(d_{1,1},\ (v\,d_{2,3}\,d_{5,3})^{r_{1,3}+r_{3,1}}(d_{2,1}\,d_{5,1})^{2r_{1,1}}(d_{2,2}\,d_{5,2})^{(r_{1,2}+r_{2,1})}\big)\\
\cdot e\big(d_{1,2},\ (v\,d_{2,3}\,d_{5,3})^{r_{2,3}+r_{3,2}}(d_{2,1}\,d_{5,1})^{r_{1,2}+r_{2,1}}(d_{2,2}\,d_{5,2})^{2r_{2,2}}\big)\\
e\big(d_{1,3},\ (v\,d_{2,3}\,d_{5,3})^{2r_{3,3}}(d_{2,1}\,d_{5,1})^{r_{1,3}+r_{3,1}}(d_{2,2}\,d_{5,2})^{r_{2,3}+r_{3,2}}\big)\\
\cdot e(f, d_{2,1}^{s_1}d_{2,2}^{s_2}d_{2,3}^{s_3}d_{5,1}^{t_1}d_{5,2}^{t_2}d_{5,3}^{t_3})\cdot e(w^{-1}, d_{3,1}^{s_1}d_{3,2}^{s_2}d_{3,3}^{s_3})\cdot e(h^{-1}, d_{4,1}^{t_1}d_{4,2}^{t_2}d_{4,3}^{t_3})\\
= e\big(u_{1,1},\ \phi_{1,1}^{2\cdot r_{1,1}}\phi_{1,2}^{r_{1,2}+r_{2,1}}\phi_{1,3}^{r_{1,3}+r_{3,1}}\psi_1^{s_1}\theta_1^{t_1}\big)\\
\cdot e\big(u_{1,3},\ (\phi_{1,1}\phi_{2,1})^{r_{1,3}+r_{3,1}}(\phi_{1,2}\phi_{2,2})^{r_{2,3}+r_{3,2}}(\phi_{1,3}\phi_{2,3})^{2\cdot r_{3,3}}(\psi_1\psi_2)^{s_3}(\theta_1\theta_2)^{t_3}\big)\\
\cdot e\big(u_{2,2},\ \phi_{2,1}^{r_{1,2}+r_{2,1}}\phi_{2,2}^{2\cdot r_{2,2}}\phi_{2,3}^{r_{2,3}+r_{3,2}}\psi_2^{s_2}\theta_2^{t_2}\big)e\big(u_{3,1},\ \phi_{3,1}^{2\cdot r_{1,1}}\phi_{3,2}^{r_{1,2}+r_{2,1}}\phi_{3,3}^{r_{1,3}+r_{3,1}}\psi_3^{s_1}\theta_3^{t_1}\big)\\
\cdot e\big(u_{3,2},\ \phi_{3,1}^{r_{1,2}+r_{2,1}}\phi_{3,2}^{2\cdot r_{2,2}}\phi_{3,3}^{r_{2,3}+r_{3,2}}\psi_3^{s_2}\theta_3^{t_2}\big)e\big(u_{3,3},\ \phi_{3,1}^{r_{1,3}+r_{3,1}}\phi_{3,2}^{r_{2,3}+r_{3,2}}\phi_{3,3}^{2\cdot r_{3,3}}\psi_3^{s_3}\theta_3^{t_3}\big)e(g,g)^{2r_{3,3}}
\end{aligned}
$$

In [BCKL08], the authors evaluated that the verification of the proof in the DLin instantiation requires the computation of 126 pairings. With our result, we prove it can be reduced to 12.

Batching Several P-Signatures: Like with the previous scheme, in the situation where we want to verify multiple P-signatures at once, we can further unify some extra pairings (here those containing f, h and w on the left-hand side and all pairings (which are of the form $e(u_{i,j}, \cdot)$) on the right-hand side). Instead of $15\,n$ (*resp.* $12\,n$) pairings needed when checking each equation, the batched version only requires $2\,n + 13$ (*resp.* $3\,n + 9$) pairings.

This last result emphasizes the improvement our batching verification technique proposes. From 126 pairings computation needed for a single signature, we successfully reduced it to $3\,n + 9$ for n signatures. The two previous scheme were not conceived with those batching techniques in mind, and however benefits greatly from our approach, which leads to think that our techniques on many of today's existing pairing-based schemes, using the Groth-Sahai framework might yield huge improvements without a drastic reduction of the security.

2.6.3 Asymmetric Waters Signature

In this section, we briefly present an asymmetric version of the Waters Signature. This version is useful in instantiations in asymmetric groups. To prove its security we need to rely on the CDH^+ hypothesis recalled in Section 2.2.2, page 17.

Waters$^+$ Signature Scheme
$\ulcorner\, \mathcal{S} = (\mathsf{Setup}_{\mathcal{S}}, \mathsf{KeyGen}_{\mathcal{S}}, \mathsf{Sign}, \mathsf{Verif})$:

- $\mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and more specifically the bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, an extra generator $h_1 \in \mathbb{G}_1$, and generators $(u_i)_{[\![0,k]\!]} \in \mathbb{G}_1^{k+1}$ for the Waters function, where once again k is a polynomial in $\mathfrak{K}$, $\mathcal{F}(m) = u_0 \prod_{i\in[\![1,k]\!]} u_i^{m_i}$, where $m = (m_1, \ldots, m_k) \in \{0,1\}^k$.
- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$ picks a random $x \xleftarrow{\$} \mathbb{Z}_p$ and outputs the secret key $\mathsf{sk} = Y = h_1^x$, and the verification key $\mathsf{vk} = X = (g_1^x, g_2^x)$;
- $\mathsf{Sign}(\mathsf{sk}, m; \mu)$ outputs a signature $\sigma(m) = (Y\mathcal{F}(m)^{\mu}, g_1^{-\mu}, g_2^{-\mu})$;
- $\mathsf{Verif}(\mathsf{vk}, m, \sigma)$ checks the validity of σ, by checking if the following pairing equations hold: $e(\sigma_1, g_2)\cdot e(\mathcal{F}(m), \sigma_3) \stackrel{?}{=} e(h_1, X_2)$ and $e(\sigma_2, g_2) \stackrel{?}{=} e(g_1, \sigma_3)$.

$\lrcorner$

Theorem 2.6.1 *The Asymmetric Waters signature scheme is randomizable and existentially unforgeable under the* CDH^+ *assumption.*

Proof: First, let us define $\mathsf{Random}(\mathsf{vk}, (F, \Pi_M), \sigma = (\sigma_1, \sigma_2, \sigma_3); s')$ to output $\sigma' = (\sigma_1 \cdot F^{s'}, \sigma_2 \cdot g_1^{-s'}, \sigma_3 \cdot g_2^{-s'})$, for random $s' \xleftarrow{\$} \mathbb{Z}_p$. We easily see it corresponds to $\mathsf{Sign}(\mathsf{sk}, F, \Pi_M; s + s' \bmod p)$. Because of the group structure of $\mathbb{Z}_p$, we get appropriate distributions.

Let $\mathcal{A}$ be an adversary breaking the existential unforgeability of the above signature scheme, i.e. after at most q_s signing queries, it succeeds in building a new signature with probability ϵ. Let $(g_1, g_2, \lambda = (g_1^a, g_2^a), \mu = g_1^b)$ be a CDH^+-instance. We show how an adversary $\mathcal{B}$ can compute g_1^{ab} thanks to $\mathcal{A}$.

Setup$_{\mathcal{S}}$. Pick a random position $j \xleftarrow{\$} \{0,\ldots,k\}$, choose random indices $y_0, y_1, \ldots, y_k \xleftarrow{\$} \{0,\ldots,2q_s-1\}$, and random scalars $z_0, z_1, \ldots, z_k \xleftarrow{\$} \mathbb{Z}_p$. One defines $X_1 = \lambda_1$, $X_2 = \lambda_2$, $h_1 = g_1^b$, $u_0 = h_1^{y_0-2jq_s} g^{z_0}$, $u_i = h_1^{y_i} g_1^{z_i}$.

Signing queries. To answer a signing query on m, with a message $M = (M_i)$, we define

$$H = -2jq_s + y_0 + \sum_i y_i M_i, \quad J = z_0 + \sum_i z_i M_i : \mathcal{F}(M) = h_1^H g_1^J.$$

If $H \equiv 0 \pmod p$ then abort, otherwise set $\sigma = (X_1^{-J/H}\mathcal{F}(M)^s, X_1^{1/H} g_1^{-s}, X_2^{1/H} g_2^{-s})$. Defining $\tilde{\mu} = s - a/H$, we have:

$$\begin{aligned}\sigma &:= \left(X_1^{-J/H}(h_1^H g_1^J)^s,\, X_1^{1/H} g_1^{-s},\, X_2^{1/H} g_2^{-s}\right)\\ &= \left(X_1^{-J/H}(h_1^a g_1^{Ja/H})(\mathcal{F}(M))^{\tilde{\mu}},\, X_1^{1/H} g_1^{-a/H} g_1^{-\tilde{\mu}},\, X_2^{1/H} g_2^{-a/H} g_2^{-\tilde{\mu}}\right)\\ &= \left(h_1^a(\mathcal{F}(M))^{\tilde{\mu}},\, g_1^{-\tilde{\mu}},\, g_2^{-\tilde{\mu}}\right).\end{aligned}$$

After at most q_s signing queries $\mathcal{A}$ outputs a forgery $\sigma^* = (\sigma_1^*, \sigma_2^*, \sigma_3^*)$ on M^*. As before, we define

$$H^* = -2jq_s + y_0 + \sum_i y_i M_i^*, \quad J^* = z_0 + \sum_i z_i M_i^* : \mathcal{F}(M^*) = h_1^{H^*} g_1^{J^*}.$$

If $H^* \not\equiv 0 \pmod p$ then abort, otherwise, as shown above, for some μ^*, $\sigma^* = (h_1^a \mathcal{F}(M^*)^{\mu^*}, g_1^{-\mu^*}, g_2^{-\mu^*})$, and thus $\sigma^* = (h_1^a g_1^{J^*\mu^*}, g_1^{-\mu^*}, g_2^{-\mu^*})$. As a consequence, $\sigma_1^*(\sigma_2^*)^{J^*} = h_1^a = g_1^{ab}$: one has solved the CDH^+ problem. As shown in [HK08], it occurs with non-negligible probability that all the H are null except H^*. □

2.6.4 Waters Function Programmability

In this section, we prove that for a polynomial alphabet size, the Waters function remains programmable. We recall some notations introduced in [HK08] and show our result which can be seen as a generalization of a result presented by Naccache [Nac05] where he considered a variant of Waters identity-based encryption [Wat05] with shorter public parameters.

Definitions

Let us recall some basic definitions. A family of cyclic groups $G = (\mathbb{G}_{\mathfrak{K}})_{\mathfrak{K}\in\mathbb{N}}$, indexed by a security parameter $\mathfrak{K}$, is called a *group family*. A *group hash function* H for G, an alphabet $\Sigma = \Sigma(\mathfrak{K})$ and an input length $\ell = \ell(\mathfrak{K})$ is a pair of probabilistic polynomial-time algorithms $(\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ such that:

- $\mathsf{PHF.Gen}$ takes as input a security parameter $\mathfrak{K}$ and outputs a key κ
- $\mathsf{PHF.Eval}$ takes as input a key κ output by $\mathsf{PHF.Gen}$ and a string $X \in \Sigma^\ell$ and outputs an element of $\mathbb{G}_{\mathfrak{K}}$.

Group Hash Function [HK08]
⌜ A group hash function $(\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ is (m, n, δ)-programmable, if there exist two probabilistic polynomial-time algorithms $(\mathsf{PHF.TrapGen}, \mathsf{PHF.TrapEval})$ such that

- **Syntactic:** For $g, h \in \mathbb{G}, \mathsf{PHF.TrapGen}(1^{\mathfrak{K}}, g, h)$ generates a key κ' and a trapdoor t such that $\mathsf{PHF.TrapEval}(t, X)$ produces integers a_X, b_X for any $X \in \Sigma^\ell$
- **Correctness:** For all generators $g, h \in \mathbb{G}$, all (κ', t) output by $\mathsf{PHF.TrapGen}(1^{\mathfrak{K}}, g, h)$ and all $X \in \Sigma^\ell$, $H_{\kappa'}(X) := \mathsf{PHF.Eval}(\kappa', X)$ satisfies $H_{\kappa'}(X) = g^{a_X} h^{b_X}$ where $(a_X, b_X) := \mathsf{PHF.TrapEval}(t, X)$.
- **Statistically close trapdoor keys:** For all $g, h \in \mathbb{G}^2, \mathsf{PHF.Gen}(1^{\mathfrak{K}})$ and $\mathsf{PHF.TrapGen}(1^{\mathfrak{K}}, g, h)$ output keys κ and κ' statistically close.

- **Well-distributed logarithms:** For all $g, h \in \mathbb{G}$, all (κ', t) output by $\mathsf{PHF.TrapGen}(1^{\mathfrak{K}}, g, h)$ and all bit-strings $(X_i)_{1,\ldots,m}, (Z_i)_{1,\ldots,n} \in \Sigma^{\ell}$ such that $\forall i, j, X_i \neq Z_j$, we have

$$\Pr[a_{X_1} = \ldots = a_{X_m} = 0 \wedge a_{Z_1}, \ldots, a_{Z_n} \neq 0] \geq \delta$$

where the probability is taken over the random coins used by $\mathsf{PHF.TrapGen}$ and $(a_{X_i}, b_{X_i}) := \mathsf{PHF.TrapEval}(t, X_i)$ and $(a_{Z_i}, b_{Z_i}) := \mathsf{PHF.TrapEval}(t, Z_i)$.

⌟

Instantiation with Waters function

Let us consider the Waters function presented in [Wat05].

Multi-Generator PHF
⌜ Let $G = (\mathbb{G}_{\mathfrak{K}})$ be a group family, and $\ell = \ell(\mathfrak{K})$ a polynomial. We define $\mathcal{F} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ as the following group hash function:

- $\mathsf{PHF.Gen}(1^{\mathfrak{K}})$ outputs a uniformly and independently sampled $\kappa = (h_0, \ldots, h_\ell) \in \mathbb{G}^{\ell+1}$
- $\mathsf{PHF.Eval}(\kappa, X)$ parses κ and $X = (x_1, \ldots, x_\ell) \in \{0,1\}^{\ell}$ and outputs $\mathcal{F}_\kappa(X) = h_0 \prod_{i=1}^{\ell} h_i^{x_i}$.

⌟

This function was shown to be $(1, q, \delta)$-programmable with a $\delta = O(1/(q\sqrt{\ell}))$ and $(2, 1, \delta)$-programmable with a $\delta = O(1/\ell)$ (*cf.* [HK08]).

However this definition requires to generate and store $n+1$ group generators where n is the bit-length of the messages you want to hash. We consider a more general case where instead of hashing bit-per-bit we decide to hash blocks of bits.

Improved Multi-Generator PHF
⌜ Let $G = (\mathbb{G}_{\mathfrak{K}})$ be a group family, $\Sigma = \{0, \ldots, \tau\}$ a finite alphabet and $\ell = \ell(\mathfrak{K})$ a polynomial. We define $\mathcal{F} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ as the following group hash function:

- $\mathsf{PHF.Gen}(1^{\mathfrak{K}})$ returns a uniformly and independently sampled $\kappa = (h_0, \ldots, h_\ell) \in \mathbb{G}^{\ell+1}$
- $\mathsf{PHF.Eval}(\kappa, X)$ parses κ and $X = (x_1, \ldots, x_\ell) \in \Sigma^{\ell}$ and returns ${\mathcal{F}^+}_\kappa(X) = h_0 \prod_{i=1}^{\ell} h_i^{x_i}$.

⌟

Using a larger alphabet allows to hash from a larger domain with a smaller hash key, but it comes at a price since one can easily prove that the function is no longer $(2,1)$-programmable:

Theorem 2.6.2 ((2,1)-Programmability) *For any group family G with known order and $\tau > 1$, the function $\mathcal{F}^+$ is not a (2,1)-programmable hash function if the discrete logarithm problem is hard in G.*

Proof: Consider a discrete logarithm challenge (g, h) in a group $\mathbb{G}$ and suppose by contradiction that the function $\mathcal{F}^+$ is $(2,1)$-programmable with $\tau \geq 2$ (i.e. , we suppose that there exist two probabilistic polynomial-time algorithms $(\mathsf{PHF.TrapGen}, \mathsf{PHF.TrapEval})$ satisfying the definition 2.6.4 for a non-negligible δ).

For any hash key κ' and trapdoor t generated by $\mathsf{PHF.TrapGen}(1^{\lambda}, g, h)$, we can consider the messages $X_1 = (2,0), X_2 = (1,1), Z = (0,2)$ and with non-negligible probability over the random coins used by $\mathsf{PHF.TrapGen}$ we have $a_{X_1} = a_{X_2} = 0$ and $a_Z \neq 0$ where $(a_{X_1}, b_{X_1}) := \mathsf{PHF.TrapEval}(t, X_1)$, $(a_{X_2}, b_{X_2}) := \mathsf{PHF.TrapEval}(t, X_2)$ and $(a_Z, b_Z) := \mathsf{PHF.TrapEval}(t, Z)$. By the correctness property, we have $g^{a_Z} h^{b_z} = h_0 h_2^2 = h^{2b_{X_2}} / h^{b_{X_1}}$ and we can extract the discrete logarithm of g in base h as follows:

$$\log_h(g) = \frac{2b_{X_2} - b_{X_1} - b_Z}{a_Z} \mod |\mathbb{G}_\lambda|.$$

□
□

However we still have the interesting property:

Theorem 2.6.3 ((1,poly)-Programmability) *For any polynomial q and a group family G with groups of known order, the function $\mathcal{F}^+$ is a $(1, q, \delta)$-programmable hash function with a $\delta = \Omega(1/\tau q \ell)$.*

Remark In order to be able to sign all messages in a set $\mathcal{M}$, we have to consider parameters τ and ℓ such that $\tau^\ell \geq \#\mathcal{M}$, but the security is proved only if the value δ is non-negligible (*i.e.* if $\ell = \lambda^{O(1)}$ and $\tau = \lambda^{O(1)}$). In particular if $\mathcal{M}$ is of polynomial size in λ (which is the case in our WSN application with data aggregation), one can use $\tau = \#\mathcal{M}$ and $\ell = 1$ (namely, the Boneh-Boyen hash function), and therefore get data confidentiality.

Proof: Let us first introduce some notations. Let $n \in \mathbb{N}^*$, let A_j be independent and uniform random variables in $\{-1, 0, 1\}$. If we denote $2\sigma_j^2$ their quadratic moment, we have $2\sigma_j^2 = 2/3$ and $\sigma_j = \sqrt{1/3}$. We note $s_n^2 = \sum_{j=1}^n \sigma_j^2 = n/3$.

The Local Central Limit Theorem. Our analysis relies on a classical result on random walks, called the *Local Central Limit Theorem*. It basically provides an approximation of $\Pr[\sum A_j = a]$ for independent random variables A_j. This is a version of the Central Limit Theorem in which the conclusion is strengthened from convergence of the law to locally uniform pointwise convergence of the densities. It is worded as follows in [DM95], where ϕ and Φ are the standard normal density and distribution functions:

Theorem 2.6.4 *Let A_j be independent, integer valued random variables where A_j has probability mass function f_j. For each j, let $q(f_j) = \sum_k \min(f_j(k), f_j(k+1))$ and $Q_n = \sum_{i=1}^n q(f_j)$. Denote $S_n = A_1 + \cdots + A_n$. Suppose that there are numbers α_n, β_n such that*

1. $\lim_{n\to\infty} \Pr[(S_n - \alpha_n)/\beta_n) < t] = \Phi(t), -\infty < t < \infty$,
2. $\beta_n \to \infty$,
3. *and* $\limsup \beta_n^2/Q_n < \infty$,

then $\sup_k |\beta_n \Pr[S_n = k] - \phi((k - \alpha_n)/\beta_n)| \to 0$ *as* $n \to \infty$.

While those notations may seem a little overwhelming, this can be easily explained in our case. With $A_j \in \{-1, 0, 1\}$ with probability 1/3 for each value.

1. It requires the variables to verify the Lindeberg-Feller theorem. However as long as the variables verify the Lindeberg's condition[2], this is true for $\beta_n = s_n$ and $\alpha_n = 0$.
2. In our application, $\beta_n = s_n = \sqrt{n/3}$, so once again we comply with the condition.
3. Since $f_j(k)$ is simply the probability that A_j equals k, then $q(f_j) = 2/3$. This leads to $Q_n = 2n/3$. As a consequence, $\beta_n^2/Q_n = 1/2$.

So we have: $\sup_k |\beta_n \Pr[S_n = k] - \phi((k - \alpha_n)/\beta_n)| \to 0$, that is, in our case

$$\sup_k |\sqrt{n/3} \Pr[S_n = k] - \phi(k/\sqrt{n/3})| \to 0.$$

We will solely focus on the case $k = 0$: since $\phi(0) = 1/\sqrt{2\pi}$, $\Pr[S_n = 0] = \Theta(1/\sqrt{n})$. In addition, it is clear that $\Pr[S_n = k] \leq \Pr[S_n = 0]$ for any $k \neq 0$. (cf [HK08])

Lemma 2.6.5 *Let $(A_{ij})_{[\![1,n]\!]\times[\![1,J]\!]}$ be independent, integer valued random variables in $\{-1, 0, 1\}$, then $\forall X \in [\![1, \tau]\!]^n$, $\Pr[\sum_{i=1}^n \sum_{j=1}^J X_i A_{ij} = 0] = \Omega(1/\tau\sqrt{nJ})$, where the probability distribution is over the A_{ij}.*

This lemma will be useful to prove the lower bound in the following, we only consider word with no-null coefficient X_i, if a X_i is null, we simply work with a shorter random walk of length $J \cdot (n-1)$ instead of Jn.

Proof: Let us denote d_{ij}, the random variable defined as $X_i A_{ij}$: they are independent, integer valued random variables. As above, $s_n^2 = \sum_{i=1}^n \sum_{j=1}^J \sigma_j^2 = \sum_{i=1}^n JX_i^2/3$. So $nJ/3 \leq s_n^2 \leq n\tau^2 J/3$.

[2] Lindeberg's condition is a sufficient criteria of the Lindeberg-Feller theorem, for variables with a null expected value it requires that $\forall \epsilon > 0, \lim_{n\to\infty} 1/s_n^2 \sum_{j=1}^n E[A_j^2 \cdot 1_{\{|A_j| > \epsilon s_n\}}] \to 0$. In our case, as soon as $n > 3/\epsilon^2$, we have $|A_j| \leq 1 \leq \epsilon\sqrt{n/3} \leq \epsilon s_n$, so the sum is null. ($1_{\{|A_j|>\epsilon s_n\}}$ is the indicator function of variables greater that ϵs_n

1. The Lindeberg's condition is verified. As soon as $n > 3\tau/J\epsilon^2$ we have $\epsilon s_n > \tau$ and so $|d_{ij}| < s_n$, and so once again the sum is null.

2. $s_n \to \infty$.

3. Each $d_{ij} \in \{-X_i, 0, X_i\}$ with probability 1/3 for each value, so $q(f_{ij}) = 2/3$ and so $Q_n = \sum_{i,j} q(f_{ij}) = 2nJ/3$. So $\beta_n^2/Q_n \leq (n\tau J/3)/(2nJ/3) \leq \tau/2 < \infty$.

Then we can apply the Local Central Limit Theorem to d_{ij}, and conclude: $\Pr[\sum_{i=1}^{n}\sum_{j=1}^{J} X_i A_{ij} = 0] = \Theta(1/s_n) = \Theta(1/\tau\sqrt{(nJ)}$. □

In the following, we will denote $a(X) = \sum_{i=1}^{n} a_i X_i$, where $X \in \{0, \ldots, \tau\}^n$. The probabilities will be over the variable a_{ij} while X and Y are assumed to be chosen by the adversary. Our goal is to show that even for bad choices of X and Y, a random draw of a_{ij} provides enough freedom.

Let $J = J(\lambda)$ be a positive function. We define the following two probabilistic polynomial-time algorithms ($\mathsf{PHF.TrapGen}, \mathsf{PHF.TrapEval}$):

- $\mathsf{PHF.TrapGen}(1^\lambda, g, h)$**:** which chooses some independent and uniform elements $(a_{ij})_{(0,\ldots,\ell),(1,\ldots,J)}$ in $\{-1, 0, 1\}$, and random group exponents $(b_i)_{(0,\ldots,\ell)}$. It sets $h_0 = g^{a_0}h^{b_0}$, and $a_i = \sum_{j=1}^{J} a_{ij}$, $h_i = g^{a_i}h^{b_i}$ for $i \in \{1, \ldots, \ell\}$. It then outputs the hash key $\kappa = (h_0, \ldots, h_\ell)$ and the trapdoor $t = (b_0, \ldots, a_\ell, b_\ell)$.

- $\mathsf{PHF.TrapEval}(t, X)$**:** which parses $X = (X_1, \ldots, X_\ell) \in \Sigma^\ell = \{0, \ldots, \tau\}^\ell$ and outputs $a_X = a_0 + \sum a_i X_i$ and $b_X = b_0 + \sum b_i X_i$.

As this definition verifies readily the syntactic and correctness requirements, we only have to prove the two other ones. We stress the importance of the hardwired 1 in front of a_0 this allows us to consider multisets $X' = 1 :: X$ and $Y' = 1 :: Y$, and so there is no k such that $X' = kY'$. And we also stress that $a_i = \sum_{j=1}^{J} a_{ij}$ is already a random walk of length J (described by the a_{ij}), on which we can apply the Local Central Limit Theorem and so $\Pr[a_i = 0] = \Theta(1/\sqrt{J})$. By noticing that summing independent random walks is equivalent to a longer one and applying the Local Central Limit Theorem, we have:

$$\Theta(1/\tau\sqrt{(\ell+1)J}) \leq \Pr[a(X') = 0] \leq \Theta(1/\sqrt{J}).$$

To explain further the two bounds:

- For the upper bound: we consider X fixed, and note $t = \sum_{i \leq \ell} a_i X_i$, by construction a_i are independent, so a_0 is independent from t then $\Pr[a(X') = 0] = \Pr[a_0 = -t] \leq \Pr[a_0 = 0] \leq \Theta(1/\sqrt{J})$ (because the random walk is more likely to reach 0 than any other value, and a_0 is a random walk of length J).

- For the lower bound, we proceed by recurrence on ℓ, to show $H_\ell : \Theta(1/\tau\sqrt{(\ell+1)J}) \leq \Pr[a(X') = 0]$ (where $X' \in 1 :: [\![0, \tau]\!]^\ell$): For $\ell = 0$, we consider $X' = 1$, we have a random walk of length J, so $\Theta(1/\tau\sqrt{J}) \leq \Theta(1/\sqrt{J}) \leq \Pr[a(X') = 0]$. We note $X_0 = 1$ for the hardwired 1 in X'. Let us suppose the property true at rank k, let us prove it at rank $k+1$:

  - If $\exists i_0, X_{i_0} = 0$ then we can consider a random walk of length k and apply the previous step, and conclude because $\Theta(1/\tau\sqrt{(k+1)J}) \leq \Theta(1/\tau\sqrt{kJ})$
  - Else, one can apply lemma 2.6.5 to conclude.

  Therefore, $\forall \ell, \forall X' \in 1 :: [\![0, \tau]\!]^\ell, \Theta(1/\tau\sqrt{(\ell+1)J}) \leq \Pr[a(X') = 0]$

We can now deduce that $\forall X, Y \in [\![0, \tau]\!]^\ell$ with $X \neq Y$: $\Pr[a(Y') = 0 | a(X') = 0] \leq \Theta(1/\sqrt{J})$. This can easily be seen by noting i_0 the first index where $Y_i \neq X_i$. We will note $\bar{X}' = X' - X_{i_0}$, in the following we will use the fact that $a(X') = 0 \Leftrightarrow a(\bar{X}') = -a_{i_0}X_{i_0}$.($X \neq Y$ so i_0 exists, and thanks to the hardwired

1 we do not have to worry about Y' being a multiple of X'.)

$$\begin{aligned}
\Pr[a(Y')=0|a(X')=0] &\leq \Pr[a(Y')=a(X')|a(X')=0] \\
&\leq \Pr[Y_{i_0}a_{i_0}+a(\bar{Y}')=X_{i_0}a_{i_0}+a(\bar{X}')|a(X')=0] \\
&\leq \max_t \Pr[(Y_{i_0}-X_{i_0})a_{i_0}=t|a(\bar{X}')=-X_{i_0}a_{i_0}] \qquad (2.4)\\
&\leq \max_{s,t'} \Pr[a_{i_0}=t'|a(\bar{X}')=s] \qquad (2.5)\\
&\leq \max_{t'} \Pr[a_{i_0}=t'] \qquad (2.6)\\
&\leq \Pr[a_{i_0}=0] \\
&\leq \Theta(1/\sqrt{J})
\end{aligned}$$

(2.4) we start with $(Y_{i_0}-X_{i_0})a_{i_0}=a(\bar{X}')-a(\bar{Y}')$, and then consider the max probability for all values $a(\bar{X}')-a(\bar{Y}')$.

(2.5) We consider the maximum probability for all values of $-X_{i_0}a_{i_0}$.

(2.6) a_{i_0} and $a(\bar{X}')$ are independent.

Hence, for all $X_1, Y_1, \ldots, Y_q$, we have

$$\begin{aligned}
\Pr[a_{X_1}=0 \wedge a_{Y_1}, \ldots, a_{Y_q}\neq 0] &= \Pr[a_{X_1}=0]\Pr[a_{Y_1}, \ldots, a_{Y_q}\neq 0|a_{X_1}=0] \\
&\geq \Theta(1/\tau\sqrt{\ell J})\left(1-\sum_{i=1}^{q}\Pr[a_{Y_i}=0|a_{X_1}=0]\right) \\
&\geq \Theta(1/\tau\sqrt{\ell+1J})(1-q\Theta(1/\sqrt{J})).
\end{aligned}$$

Now we set $J=q^2$, to obtain the result. In that case the experiment success is minored by something linear in $1/(q\tau\sqrt{\ell+1})$. □

□

2.6.5 Multi Cramer-Shoup Encryption

The Cramer-Shoup (and Linear Cramer-Shoup) schemes can be extended to encrypt vectors, with labels:

Double Linear Cramer-Shoup Encryption (DLCS)

Informally the Linear Cramer-Shoup encryption scheme can be extended as follows:

- The global parameters consist of a group $\mathbb{G}$ of order p, three independent generators $g_1, g_2, g_3 \xleftarrow{\$} \mathbb{G}$, and a collision-resistant hash function family $\mathcal{H}$.
- The key generation algorithm chooses $\mathsf{dk}=(x_1,x_2,x_3,y_1,y_2,y_3,z_1,z_2,z_3) \xleftarrow{\$} \mathbb{Z}_p^9$, and sets, for $i=1,2$, $c_i=g_i^{x_i}g_3^{x_3}$, $d_i=g_i^{y_i}g_3^{y_3}$, and $h_i=g_i^{z_i}g_3^{z_3}$. It also chooses a collision-resistant hash function $\mathfrak{H}_K \xleftarrow{\$} \mathcal{H}$. The encryption key is $\mathsf{pk}=(c_1,c_2,d_1,d_2,h_1,h_2,\mathfrak{H}_K)$.
- To encrypt $(M,N)\in\mathbb{G}^2$ under the key pk with a label ℓ, one chooses $r,s,a,b \xleftarrow{\$} \mathbb{Z}_p$, computes

$$\begin{array}{rcll}
\mathcal{C} &=& (\mathbf{u}=(g_1^r,g_2^s,g_3^{r+s}), e=Mh_1^rh_2^s, v=v_1^rv_2^s) & \stackrel{\text{def}}{=} \mathsf{LCS}(\ell,\mathsf{pk},M;r,s)\\
\mathcal{C}' &=& (\alpha=(g_1^a,g_2^b,g_3^{a+b}), \beta=Nh_1^ah_2^b, \gamma=v_1^av_2^b) & \stackrel{\text{def}}{=} \mathsf{LCS}^*(\ell,\mathsf{pk},N,\xi;a,b)
\end{array}$$

  where the v and γ's are computed afterward with $v_1=c_1d_1^{\xi}$ and $v_2=c_2d_2^{\xi}$, and $\xi=\mathfrak{H}_K(\ell,\mathbf{u},e)$, hence the notation of LCS without input ξ (when it is generated during the encryption) and LCS^* with input ξ when it is provided from outside. Thus $(\mathcal{C},\mathcal{C}') \leftarrow \mathsf{Encrypt}(\ell,\mathsf{pk},M,N;r,s,a,b)$.
- The decryption algorithm first computes $\xi=\mathfrak{H}_K(\ell,\mathbf{u},e)$ and checks whether $u_1^{x_1+\xi y_1}\cdot u_2^{x_2+\xi y_2}\cdot u_3^{x_3+\xi y_3} \stackrel{?}{=} v$. If the equality holds, one computes $M=e/(u_1^{z_1}u_2^{z_2}u_3^{z_3})$ and $N=\beta/(\alpha_1^{z_1}\alpha_2^{z_2}\alpha_3^{z_3})$, and outputs (M,N). Otherwise, one outputs $\perp$.

We are now going to formalize precisely this encryption and show the IND-PD-CCA (*indistinguishability against partial-decryption chosen-ciphertext attacks*) security level of multi-message double encryption ($n-\mathsf{DLCS}$), where a pair of message-vectors is encrypted, with a common ξ computed globally on all the sub-ciphertexts of the first vector only. Hence, the two vectors $\vec{M}$ and $\vec{N}$ do not have to be sent together to be encrypted: see the Section 2.6.5, page 48 for the description and the security model for IND-PD-CCA , which provides a decryption oracle on the first vector only (the only one to be CCA-protected, since ξ is computed on this part only). The proof uses a classical hybrid argument present in the original proof of the linear Cramer-Shoup.

Theorem 2.6.6 *The Multiple $n-\mathsf{DLCS}$ encryption scheme is IND-PD-CCA if $\mathcal{H}$ is a collision-resistant hash function family, under the* DLin *assumption in $\mathbb{G}$:*

$$\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(n,q_d,t) \leq 4n \times \left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) + \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) + \frac{q_d}{p}\right).$$

In the following, we consider a CRS containing a group $\mathbb{G}$ of order p, independent generators ($\mathbf{g} = (g_1,g_2,g_3), \mathbf{h} = (h_1,h_2), \mathbf{c} = (c_1,c_2), \mathbf{d} = (d_1,d_2)$), and a collision-resistant hash function $\mathfrak{H}_K$, we will omit the encryption key and denote $\mathsf{DLCSCom}(\ell,M,N;r,s,a,b) \stackrel{\text{def}}{=} \mathsf{Encrypt}(\ell,\mathsf{pk},M,N;r,s,a,b)$. In a similar way, we will omit the key in $\mathsf{LCS}(\ell,M;r,s)$ and $\mathsf{LCS}^*(\ell,N,\xi;a,b)$, which can then all be seen as extractable commitment schemes.

We can now generalize (and prove the security) this protocol to:

Multi Double Linear Cramer-Shoup Encryption ($n-\mathsf{DLCS}$)

We can encrypt pairs of message vectors $(M_i,N_i)_{i\in[\![1,n]\!]}$, partially $\mathsf{IND}-\mathsf{CCA2}$ protected, with a common ξ:

- $\mathsf{ESetup}(1^{\mathfrak{K}})$: generates a group $\mathbb{G}$ of order p, with three independent generators $(g_1,g_2,g_3) \stackrel{\$}{\leftarrow} \mathbb{G}^3$;
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$: generates $\mathsf{dk} = (x_1,x_2,x_3,y_1,y_2,y_3,z_1,z_2,z_3) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^9$, and sets, for $i=1,2$, $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. It also chooses a collision-resistant hash function $\mathfrak{H}_K$. The encryption key is $\mathsf{ek} = (c_1,c_2,d_1,d_2,h_1,h_2,\mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell,\mathsf{ek},\vec{M};\vec{r},\vec{s})$: for a vector $\vec{M} \in \mathbb{G}^n$ and two vectors $\vec{r},\vec{s} \in \mathbb{Z}_p^n$, computes

  $$\mathcal{C} = (\mathcal{C}_1,\ldots,\mathcal{C}_n), \text{ where } \mathcal{C}_i = (\mathbf{u}_i = (g_1^{r_i}, g_2^{s_i}, g_3^{r_i+s_i}), e_i = M_i \cdot h_1^{r_i} h_2^{s_i}, v_i = (c_1 d_1^{\xi})^{r_i} (c_2 d_2^{\xi})^{s_i})$$

  with the v_i computed afterwards with $\xi = \mathfrak{H}_K(\ell,\mathbf{u}_1,\ldots,\mathbf{u}_n,e_1,\ldots,e_n)$.
- $\mathsf{Encrypt}'(\ell,\mathsf{ek},\vec{N},\xi;\vec{a},\vec{b})$: for a vector $\vec{N} \in \mathbb{G}^n$ and two vectors $\vec{a},\vec{b} \in \mathbb{Z}_p^n$, computes

  $$\mathcal{C}' = (\mathcal{C}'_1,\ldots,\mathcal{C}'_n), \text{ where } \mathcal{C}'_i = (\vec{\alpha}_i = (g_1^{a_i}, g_2^{b_i}, g_3^{a_i+b_i}), \beta_i = N_i \cdot h_1^{a_i} h_2^{b_i}, \gamma_i = (c_1 d_1^{\xi})^{a_i} (c_2 d_2^{\xi})^{b_i})$$

  where the γ_i's are computed with the above $\xi = \mathfrak{H}_K(\ell,\mathbf{u}_1,\ldots,\mathbf{u}_n,e_1,\ldots,e_n)$, hence the additional input.
  One can use both simultaneously: on input $(\ell,\mathsf{ek},\vec{M},\vec{N};\vec{r},\vec{s},\vec{a},\vec{b})$, the global encryption algorithm first calls $\mathsf{Encrypt}(\ell,\mathsf{ek},\vec{M};\vec{r},\vec{s})$ and to get $\mathcal{C}$ and ξ, and then calls $\mathsf{Encrypt}'(\ell,\mathsf{ek},\vec{N},\xi;\vec{a},\vec{b})$ to get $\mathcal{C}'$.
- $\mathsf{Decrypt}(\ell,\mathsf{dk},\mathcal{C},\mathcal{C}')$: one first parses $\mathcal{C} = (C_1,\ldots,C_n)$ and $\mathcal{C}' = (\mathcal{C}'_1,\ldots,\mathcal{C}'_n)$, where $C_i = (\mathbf{u}_i,e_i,v_i)$ and $\mathcal{C}'_i = (\vec{\alpha}_i,\beta_i,\gamma_i)$, for $i=1,\ldots,n$, computes $\xi = \mathfrak{H}_K(\ell,\mathbf{u}_1,\ldots,\mathbf{u}_n,e_1,\ldots,e_n)$ and checks whether, for $i=1,\ldots,n$, $u_{i,1}^{x_1+\xi y_1} \cdot u_{i,2}^{x_2+\xi y_2} \cdot u_{i,3}^{x_3+\xi y_3} \stackrel{?}{=} v_i$ (but not for the γ_i's). If the equality holds, one computes $M_i = e_i/(u_{i,1}^{z_1} u_{i,2}^{z_2} u_{i,3}^{z_3})$ and $N_i = \beta_i/(\alpha_{i,1}^{z_1} \alpha_{i,2}^{z_2} \alpha_{i,3}^{z_3})$, and outputs $(\vec{M} = (M_1,\ldots,M_n), \vec{N} = (N_1,\ldots,N_n))$. Otherwise, one outputs $\perp$.
- $\mathsf{PDecrypt}(\ell,\mathsf{dk},\mathcal{C})$: is a partial decryption algorithm that does as above but working on the $\mathcal{C}$ part only to get $\vec{M} = (M_1,\ldots,M_n)$ or $\perp$.

DLCS denotes the particular case where $n=1$: $\mathsf{DLCS}(\ell,\mathsf{ek},M,N;r,s,a,b) = (\mathcal{C},\mathcal{C}')$, with

$$\begin{array}{rcl} \mathcal{C} & = & (\mathbf{u} = (g_1^r, g_2^s, g_3^{r+s}), e = M \cdot h_1^r h_2^s, v = (c_1 d_1^{\xi})^r (c_2 d_2^{\xi})^s) = \mathsf{LCS}(\ell,\mathsf{ek},M;r,s), \\ \mathcal{C}' & = & (\vec{\alpha}_i = (g_1^a, g_2^b, g_3^{a+b}), \beta = N \cdot h_1^a h_2^b, \gamma = (c_1 d_1^{\xi})^a (c_2 d_2^{\xi})^b) = \mathsf{LCS}^*(\ell,\mathsf{ek},N,\xi;a,b) \end{array}$$

where $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.

This scheme is indistinguishable against *partial-decryption chosen-ciphertext* attacks, where a partial-decryption oracle only is available, but even when we allow the adversary to choose $\vec{M}$ and $\vec{N}$ in two different steps (see the security game below), under the DLin assumption and if one uses a collision-resistant hash function (see section Theorem 2.6.5, page 48).

Indistinguishability against partial-decryption chosen-ciphertext attacks for vectors, in two steps: this security notion can be formalized by the following security game, where the adversary $\mathcal{A}$ keeps some internal state between the various calls $\texttt{FIND}_M$, $\texttt{FIND}_N$ and GUESS. In the first stage $\texttt{FIND}_M$, it receives the encryption key ek; in the second stage $\texttt{FIND}_N$, it receives the encryption of $\vec{M}_b$: $\mathcal{C}^* = \mathsf{Encrypt}(\ell, \mathsf{ek}, \vec{M}_b)$; in the last stage GUESS it receives the encryption of $\vec{N}_b$: $\mathcal{C}'^* = \mathsf{Encrypt}'(\ell, \mathsf{ek}, \xi^*, \vec{N}_b)$, where ξ^* is the value involved in $\mathcal{C}$. During all these stages, it can make use of the oracle $\mathsf{ODecrypt}(\ell, \mathcal{C})$, that outputs the decryption of $\mathcal{C}$ under the label ℓ and the challenge decryption key dk, using $\mathsf{PDecrypt}(\ell, \mathsf{dk}, \mathcal{C})$. The input queries $(\ell, \mathcal{C})$ are added to the list $\mathcal{CT}$.

$\mathsf{Exp}^{\mathsf{ind-pd-cca}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}, n)$
1. $\mathsf{param} \leftarrow \mathsf{ESetup}(1^{\mathfrak{K}})$; $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param})$
2. $(\ell^*, \vec{M}_0, \vec{M}_1) \leftarrow \mathcal{A}(\texttt{FIND}_M : \mathsf{ek}, \mathsf{ODecrypt}(\cdot,\cdot))$
3. $\mathcal{C}^* \leftarrow \mathsf{Encrypt}(\ell^*, \mathsf{ek}, \vec{M}_b)$
4. $(\vec{N}_0, \vec{N}_1) \leftarrow \mathcal{A}(\texttt{FIND}_N : \mathcal{C}^*, \mathsf{ODecrypt}(\cdot,\cdot))$
5. $\mathcal{C}'^* \leftarrow \mathsf{Encrypt}'(\ell^*, \mathsf{ek}, \xi^*, \vec{N}_b)$
6. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathcal{C}'^*, \mathsf{ODecrypt}(\cdot,\cdot))$
7. IF $(\ell^*, \mathcal{C}^*) \in \mathcal{CT}$ RETURN 0
8. ELSE RETURN b'

The advantages are, where q_d is the number of decryption queries:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{\mathcal{E}}(\mathcal{A}) &= |\Pr[\mathsf{Exp}^{\mathsf{ind-pd-cca}-1}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}, n) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind-pd-cca}-0}_{\mathcal{E},\mathcal{A}}(\mathfrak{K}, n) = 1]| \\ \mathsf{Adv}^{\mathsf{ind-pd-cca}}_{\mathcal{E}}(n, q_d, t) &= \max_{\mathcal{A} \le t} \mathsf{Adv}^{\mathsf{ind-pd-cca}}_{\mathcal{E}}(\mathcal{A}).\end{aligned}$$

Theorem 2.6.7 *The Multiple $n-$DLCS encryption scheme is IND-PD-CCA if $\mathcal{H}$ is a collision-resistant hash function family, under the* DLin *assumption in $\mathbb{G}$:*

$$\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(n, q_d, t) \le 4n \times \left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) + \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) + \frac{q_d}{p}\right).$$

Corollary 2.6.8 *The Multiple $n-$LCS encryption scheme is IND-CCA if $\mathcal{H}$ is a collision-resistant hash function family, under the* DLin *assumption in $\mathbb{G}$.*

Proof: Let us be given a DLin challenge $(g_1, g_2, g_3, u_1 = g_1^r, u_2 = g_2^s, u_3 = g_3^t)$, for which we have to decide whether (u_1, u_2, u_3) is a linear tuple in basis (g_1, g_2, g_3), and thus $t = r + s \bmod p$, or a random one. From an IND-PD-CCA adversary $\mathcal{A}$ against the encryption scheme, we built a DLin distinguisher $\mathcal{B}$. The latter first uses (g_1, g_2, g_3) as the global parameters. It also picks $x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3 \xleftarrow{\$} \mathbb{Z}_p^9$ and sets $c_i = g_i^{x_i} g_3^{x_3}, d_i = g_i^{y_i} g_3^{y_3}, h_i = g_i^{z_i} g_3^{z_3}$, for $i = 1, 2$. It chooses a collision-resistant hash function $\mathfrak{H}_K$ and provides $\mathcal{A}$ with the encryption key $\mathsf{ek} = (c_1, c_2, d_1, d_2, h_1, h_2, \mathfrak{H}_K)$.

- In the initial game $\mathcal{G}_0$,
  - A's decryption queries are answered by $\mathcal{B}$, simply using the decryption key dk.
  - When $\mathcal{A}$ submits the first challenge vectors $\vec{M}_0 = (M_{0,1}, \ldots, M_{0,n})$ and $\vec{M}_1 = (M_{1,1}, \ldots, M_{1,n})$, with a label ℓ^*, $\mathcal{B}$ chooses a random bit $b \xleftarrow{\$} \{0,1\}$ and encrypts $\vec{M}_b$:
    - it chooses two random vectors $\vec{r}^*, \vec{s}^* \xleftarrow{\$} \mathbb{Z}_p^n$
    - it defines $\mathcal{C}_i^* = (\mathbf{u}_i^* = (g_1^{r_i^*}, g_2^{s_i^*}, g_3^{r_i^*+s_i^*}), e_i^* = M_{b,i} \cdot h_1^{r_i^*} h_2^{s_i^*}, v_i^* = (c_1 d_1^{\xi^*})^{r_i^*} (c_2 d_2^{\xi^*})^{s_i^*})$, for $i = 1, \ldots, n$, where the v_i^*'s are computed with $\xi^* = \mathfrak{H}_K(\ell^*, \mathbf{u}_1^*, \ldots, \mathbf{u}_n^*, e_1^*, \ldots, e_n^*)$, and $\mathcal{C}^* = (\mathcal{C}_1^*, \ldots, \mathcal{C}_n^*)$.
  - When $\mathcal{A}$ submits the second challenge vectors $\vec{N}_0 = (N_{0,1}, \ldots, N_{0,n})$ and $\vec{N}_1 = (N_{1,1}, \ldots, N_{1,n})$,

* $\mathcal{B}$ chooses two random vectors $\vec{a}^*, \vec{b}^* \xleftarrow{\$} \mathbb{Z}_p^n$
* it defines $\mathcal{C}_i'^* = (\vec{\alpha}_i^* = (g_1^{a_i^*}, g_2^{b_i^*}, g_3^{a_i^*+b_i^*}), \beta_i^* = N_{b,i} \cdot h_1^{a_i^*} h_2^{b_i^*}, \gamma_i^* = (c_1 d_1^{\xi^*})^{a_i^*} (c_2 d_2^{\xi^*})^{b_i^*})$, for $i = 1, \ldots, n$, where the γ_i^*'s are computed with the above $\xi^* = \mathfrak{H}_K(\ell^*, \mathbf{u}_1^*, \ldots, \mathbf{u}_n^*, e_1^*, \ldots, e_n^*)$, and $\mathcal{C}'^* = (\mathcal{C}_1'^*, \ldots, \mathcal{C}_n'^*)$.

– When $\mathcal{A}$ returns b', $\mathcal{B}$ outputs $b' \stackrel{?}{=} b$.

$$\Pr_0[1 \leftarrow \mathcal{B}] = \Pr_0[b' = b] = (\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1)/2.$$

$\mathcal{G}_1$ We assume $t = r + s \bmod p$, to encrypt the challenge vectors $\vec{M}_b$ and $\vec{N}_b$, $\mathcal{B}$ does as above, excepted for $\mathcal{C}_1^*$: $\mathcal{C}_1^* = (\mathbf{u}_1^* = (u_1, u_2, u_3), e_i^* = M_{b,1} \cdot u_1^{z_1} u_2^{z_2} u_3^{z_3}, v_1^* = u_1^{x_1+\xi^* y_1} u_2^{x_2+\xi^* y_2} u_3^{x_3+\xi^* y_3})$, which actually defines $r_1^* = r$ and $s_1^* = s$.

$$\mathbf{u}_1^* = (g_1^{r_1^*}, g_2^{s_1^*}, g_3^{r_1^*+s_1^*}) \qquad e_1^* = M_{b,1} \cdot (g_1^{r_1^*})^{z_1} (g_2^{s_1^*})^{z_2} (g_3^{r_1^*+s_1^*})^{z_3} = M_{b,1} \cdot h_1^{r_1^*} h_2^{s_1^*}$$
$$v_1^* = (g_1^{r_1^*})^{x_1+\xi^* y_1} (g_2^{s_1^*})^{x_2+\xi^* y_2} (g_3^{r_1^*+s_1^*})^{x_3+\xi^* y_3} = (c_1 d_1^{\xi^*})^{r_1^*} (c_2 d_2^{\xi^*})^{s_1^*}$$

The challenge ciphertexts are identical to the encryptions of $\vec{M}_b$ and $\vec{N}_b$ in $\mathcal{G}_0$. Decryption queries are still answered the same way. Hence the gap between this game and the previous game is 0.

$$\Pr_1[1 \leftarrow \mathcal{B}] = \Pr_0[1 \leftarrow \mathcal{B}] = (\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1)/2.$$

$\mathcal{G}_2$ We now assume that $t \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). First, we have to check that the *incorrect* computation of v_1^* does not impact the probability to reject invalid ciphertexts, then we prove that e_1^* is totally independent of $M_{b,1}$.

1. About the validity checks, $u_{i,1}^{x_1+\xi y_1} \cdot u_{i,2}^{x_2+\xi y_2} \cdot u_{i,3}^{x_3+\xi y_3} \stackrel{?}{=} v_i$, where $\xi = \mathcal{H}(k\ell, \mathbf{u}_1, \ldots, \mathbf{u}_n, e_1, \ldots, e_n)$, three cases can appear with respect to the challenge ciphertext $\mathcal{C}^* = ((\mathbf{u}_1^*, e_1^*, v_1^*), \ldots, (\mathbf{u}_n^*, e_n^*, v_n^*))$:
   (a) $(\ell, \mathbf{u}_1, e_1, \ldots, \mathbf{u}_n, e_n) = (\ell^*, \mathbf{u}_1^*, e_1^*, \ldots, \mathbf{u}_n^*, e_n^*)$, then necessarily, for some i, $v_i \neq v_i^*$, then the check on index i will fail since one value only is acceptable;
   (b) $(\ell, \mathbf{u}_1, e_1, \ldots, \mathbf{u}_n, e_n) \neq (\ell^*, \mathbf{u}_1^*, e_1^*, \ldots, \mathbf{u}_n^*, e_n^*)$, but $\xi = \xi^*$, then the adversary has generated a collision for the hash function $\mathfrak{H}_K$.
   (c) $(\ell, \mathbf{u}_1, e_1, \ldots, \mathbf{u}_n, e_n) \neq (\ell^*, \mathbf{u}_1^*, e_1^*, \ldots, \mathbf{u}_n^*, e_n^*)$, and $\xi \neq \xi^*$: the ciphertext should be accepted iff $v_i = u_{i,1}^{x_1+\xi y_1} \cdot u_{i,2}^{x_2+\xi y_2} \cdot u_{i,3}^{x_3+\xi y_3}$, for $i = 1, \ldots, n$. To make it acceptable, if we denote $g_2 = g_1^{\beta_2}$ and $g_3 = g_1^{\beta_3}$, we indeed have

$$\begin{array}{lcccccccc}
\log_{g_1} c_1 & = & x_1 & & +\beta_3 x_3 & & & & \\
\log_{g_1} d_1 & = & & & & y_1 & & +\beta_3 y_3 & \\
\log_{g_1} c_2 & = & & \beta_2 x_2 & +\beta_3 x_3 & & & & \\
\log_{g_1} d_2 & = & & & & & \beta_3 y_2 & +\beta_3 y_3 &
\end{array}$$

with in addition,

$$\begin{array}{lcl}
\log_{g_1} v_1^* & = & r x_1 + s\beta_2 x_2 + t\beta_3 x_3 + r\xi^* y_1 + s\xi^*\beta_2 y_2 + t\xi^*\beta_3 y_3 \\
\log_{g_1} v_i^* & = & r_i^* x_1 + s_i^*\beta_2 x_2 + (r_i^* + s_i^*)\beta_3 x_3 + r_i^*\xi^* y_1 + s_i^*\xi^*\beta_2 y_2 + (r_i^* + s_i^*)\xi^*\beta_3 y_3 \\
 & = & r_i^* \log_{g_1} c_1 + s_i^* \log_{g_1} c_2 + \xi^* r_i^* \log_{g_1} d_1 + \xi^* s_i^* \log_{g_1} c_2 \qquad \text{for } i = 2, \ldots, n \\
\log_{g_1} \gamma_i^* & = & a_i^* x_1 + b_i^*\beta_2 x_2 + (a_i^* + b_i^*)\beta_3 x_3 + a_i^*\xi^* y_1 + b_i^*\xi^*\beta_2 y_2 + (a_i^* + b_i^*)\xi^*\beta_3 y_3 \\
 & = & a_i^* \log_{g_1} c_1 + b_i^* \log_{g_1} c_2 + \xi^* a_i^* \log_{g_1} d_1 + \xi^* b_i^* \log_{g_1} c_2 \qquad \text{for } i = 1, \ldots, n
\end{array}$$

The $2n - 1$ last relations are thus linearly dependent with the 4 above relations, hence remains the useful relations

$$\begin{array}{lcccccccc}
\log_{g_1} c_1 & = & x_1 & & +\beta_3 x_3 & & & & (1) \\
\log_{g_1} d_1 & = & & & & y_1 & & +\beta_3 y_3 & (2) \\
\log_{g_1} c_2 & = & & \beta_2 x_2 & +\beta_3 x_3 & & & & (3) \\
\log_{g_1} d_2 & = & & & & & \beta_2 y_2 & +\beta_3 y_3 & (4) \\
\log_{g_1} v_1^* & = & r x_1 & +s\beta_2 x_2 & +t\beta_3 x_3 & +r\xi^* y_1 & +s\xi^*\beta_2 y_2 & +t\xi^*\beta_3 y_3 & (5)
\end{array}$$

One can note that for v_1^* to be predictable, because of the x_1, x_2 and y_1, y_2 components, we need to have (5) $= r$ (1) $+ s$ (3) $+ r\xi^*$ (2) $+ s\xi^*$ (4), and then $t = r + s$, which is not the case, hence v_1^* looks random: in this game, v_1^* is perfectly uniformly distributed in $\mathbb{G}$. Furthermore, for any v_i in the decryption query, if $\mathbf{u}_i = (g_1^{r'}, g_2^{s'}, g_3^{t'})$ is not a linear triple, then it should be such that

$$\log_{g_1} v_i = r'x_1 + s'\beta_2 x_2 + t'\beta_3 x_3 + r'\xi y_1 + s'\xi\beta_2 y_2 + t'\xi\beta_3 y_3.$$

Since the matrix

$$\begin{pmatrix} 1 & 0 & \beta_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \beta_3 \\ 0 & \beta_2 & \beta_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_2 & \beta_3 \\ a & b\beta_2 & c\beta_3 & a\xi^* & b\xi^*\beta_2 & c\xi^*\beta_3 \\ r' & s'\beta_2 & t'\beta_3 & r'\xi & s'\xi\beta_2 & t'\xi\beta_3 \end{pmatrix} \quad \text{has determinant } \beta_2^2\beta_3^2(\xi^*-\xi)(t-r-s)(t'-r'-s') \neq 0,$$

then the correct value for v_i is unpredictable: an invalid ciphertext will be accepted with probability $1/p$.

2. Let us now consider the mask $u_1^{z_1}u_2^{z_2}u_3^{z_3}$: its discrete logarithm in basis g_1 is $rz_1+s\beta_2 z_2+t\beta_3 z_3$, whereas the informations about (z_1, z_2, z_3) are $h_1 = g_1^{z_1}g_3^{z_3}$ and $h_2 = g_2^{z_2}g_3^{z_3}$. The matrix

$$\begin{pmatrix} 1 & 0 & \beta_3 \\ 0 & \beta_2 & \beta_3 \\ r & s\beta_2 & t\beta_3 \end{pmatrix} \quad \text{has determinant } \beta_2\beta_3(t-r-s)(t'-r'-s') \neq 0,$$

then the value of the mask is unpredictable: in this game, e_1^* is perfectly uniformly distributed in $\mathbb{G}$.

Since the unique difference between the two games is the linear/random tuple, unless a collision is found for $\mathfrak{H}_K$ (probability bounded by $\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t)$) and or an invalid ciphertext is accepted (probability bounded by q_d/p), then

$$\Pr_2[1 \leftarrow \mathcal{B}] \geq \Pr_1[1 \leftarrow \mathcal{B}] - \mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}.$$

$\mathcal{G}_3$ To encrypt the challenge vectors $\vec{M}_b$ and $\vec{N}_b$, $\mathcal{B}$ does as above, excepted for $\mathcal{C}_1^*$: for a random $t_1^* \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{u}_1^* = (g_1^{r_1^*}, g_2^{s_1^*}, g_3^{t_1^*})$, $e_1^* \xleftarrow{\$} \mathbb{G}$, and $v_1^* \xleftarrow{\$} \mathbb{G}$. As just explained, this is perfectly indistinguishable with the previous game:

$$\Pr_3[1 \leftarrow \mathcal{B}] = \Pr_2[1 \leftarrow \mathcal{B}] \geq (\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1)/2 - \mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}.$$

$\mathcal{G}_4$ To encrypt the challenge vectors $\vec{M}_b$ and $\vec{N}_b$, $\mathcal{B}$ does as above, excepted for $\mathcal{C}^*$: for a random vector $\vec{t}^* \xleftarrow{\$} \mathbb{Z}_p^n$, for $i = 2, \ldots, n$: $\mathbf{u}_i^* = (g_1^{r_i^*}, g_2^{s_i^*}, g_3^{t_i^*})$, $e_i^* \xleftarrow{\$} \mathbb{G}$, and $v_i^* \xleftarrow{\$} \mathbb{G}$. Thus replacing sequentially the $\mathcal{C}_i^*$'s by random ones, as we've just done, we obtain

$$\Pr_4[1 \leftarrow \mathcal{B}] \leq \Pr_3[1 \leftarrow \mathcal{B}] - (n-1)\left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}\right).$$

$\mathcal{G}_5$ To encrypt the challenge vectors $\vec{M}_b$ and $\vec{N}_b$, $\mathcal{B}$ does as above, excepted for $\mathcal{C}'^*$: for a random vector $\vec{c}^* \xleftarrow{\$} \mathbb{Z}_p^n$, for $i = 1, \ldots, n$: $\vec{\alpha}_1^* = (g_1^{a_i^*}, g_2^{b_i^*}, g_3^{c_i^*})$, $\beta_i^* \xleftarrow{\$} \mathbb{G}$, and $\gamma_i^* \xleftarrow{\$} \mathbb{G}$. Thus replacing sequentially the $\mathcal{C}_i'^*$'s by random ones, as we've just done, we obtain

$$\Pr_5[1 \leftarrow \mathcal{B}] \leq \Pr_4[1 \leftarrow \mathcal{B}] - n\left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}\right).$$

In this last game, it is clear that $\Pr_5[1 \leftarrow \mathcal{B}] = 1/2$, since $(\vec{M}_b, \vec{N}_b)$ is not used anymore:

$$\frac{\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1}{2} - 2n \times \left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}\right) \leq \frac{1}{2},$$

which concludes the proof. □

Multi Linear Cramer-Shoup Encryption ($n-\mathsf{LCS}$)

The Linear Cramer-Shoup can be adapted the same way, to encrypt message vectors $(M_i)_{i\in[\![1,n]\!]}$, $\mathsf{IND}-\mathsf{CCA2}$ protected, with a common ξ:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: generates a group $\mathbb{G}$ of order p, with three independent generators $(g_1,g_2,g_3) \xleftarrow{\$} \mathbb{G}^3$;
- $\mathsf{KeyGen}(\mathsf{param})$: generates $\mathsf{dk}=(x_1,x_2,x_3,y_1,y_2,y_3,z_1,z_2,z_3) \xleftarrow{\$} \mathbb{Z}_p^9$, and sets, for $i=1,2$, $c_i = g_i^{x_i}g_3^{x_3}$, $d_i = g_i^{y_i}g_3^{y_3}$, and $h_i = g_i^{z_i}g_3^{z_3}$. It also chooses a collision-resistant hash function $\mathfrak{H}_K$. The encryption key is $\mathsf{pk}=(c_1,c_2,d_1,d_1,h_1,h_2,\mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell,\mathsf{pk},\vec{M};\vec{r},\vec{s})$: for two vectors $\vec{M}\in\mathbb{G}^n$ and vectors $\vec{r},\vec{s}\in\mathbb{Z}_p^n$, computes

$$\mathcal{C} \;=\; (C_1,\ldots,C_n),\ \text{where } C_i=(\mathbf{u}_i=(g_1^{r_i},g_2^{s_i},g_3^{r_i+s_i}),e_i=M_i\cdot h_1^{r_i}h_2^{s_i},v_i=(c_1d_1^{\xi})^{r_i}(c_2d_2^{\xi})^{s_i})$$

  where the v_i's are computed afterwards with $\xi=\mathfrak{H}_K(\ell,\mathbf{u}_1,\ldots,\mathbf{u}_n,e_1,\ldots,e_n)$. It outputs $\mathcal{C}$.
- $\mathsf{Decrypt}(\ell,\mathsf{dk},\mathcal{C})$: one first parses $\mathcal{C}=(C_1,\ldots,C_n)$, where $C_i=(\mathbf{u}_i,e_i,v_i)$ for $i\in[\![1,n]\!]$, computes $\xi=\mathfrak{H}_K(\ell,\mathbf{u}_1,\ldots,\mathbf{u}_n,e_1,\ldots,e_n)$ and checks whether, for $i\in[\![1,n]\!]$, $u_{i,1}^{x_1+\xi y_1}\cdot u_{i,2}^{x_2+\xi y_2}\cdot u_{i,3}^{x_3+\xi y_3} \stackrel{?}{=} v_i$. If the equality holds, one computes $M_i=e_i/(u_{i,1}^{z_1}u_{i,2}^{z_2}u_{i,3}^{z_3})$ and outputs $\vec{M}=(M_1,\ldots,M_n$. Otherwise, one outputs $\perp$.

LCS is the particular case where $n=1$.

With the same techniques as above, we can show that this scheme is indistinguishable against *chosen-ciphertext* attacks, under the DLin assumption and if one uses a collision-resistant hash function (see Theorem 2.6.5, page 48).

Multi Double Cramer-Shoup Encryption ($n-\mathsf{DCS}$)

Similarly we can encrypt pairs of message vectors $(M_i,N_i)_{i\in[\![1,n]\!]}$, partially $\mathsf{IND}-\mathsf{CCA2}$ protected, with a common ξ:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: generates a group $\mathbb{G}$ of order p, with two independent generators $(g_1,g_2) \xleftarrow{\$} \mathbb{G}^3$;
- $\mathsf{KeyGen}(\mathsf{param})$: generates $\mathsf{dk}=(x_1,x_2,y_1,y_2,z) \xleftarrow{\$} \mathbb{Z}_p^5$, and sets $c=g_1^{x_1}g_2^{x_2}$, $d=g_1^{y_1}g_2^{y_2}$, and $h=g_1^z$. It also chooses a collision-resistant hash function $\mathfrak{H}_K$. The encryption key is $\mathsf{pk}=(c,d,g_1,g_2,\mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell,\mathsf{pk},\vec{M},\vec{N};\vec{r},\vec{a})$: for two vectors $\vec{M},\vec{N}\in\mathbb{G}^n$ and two vectors $\vec{r},\vec{a}\in\mathbb{Z}_p^n$, computes

$$\begin{aligned}\mathcal{C} &= (C_1,\ldots,C_n),\ \text{where } C_i=(\mathbf{u}_i=(g_1^{r_i},g_2^{r_i}),e_i=M_i\cdot h^{r_i},v_i=(cd^{\xi})^{r_i}),\\ \mathcal{C}' &= (S_1,\ldots,S_n),\ \text{where } S_i=(\boldsymbol{\alpha}_i=(g_1^{a_i},g_2^{a_i}),\beta_i=N_i\cdot h^{a_i},\gamma_i=(cd^{\xi})^{a_i}),\end{aligned}$$

  where the v_i's and γ_i's are computed afterwards with $\xi=\mathfrak{H}_K(\ell,\mathbf{u}_1,\ldots,\mathbf{u}_n,e_1,\ldots,e_n)$. Note that once again ξ depends on the $\mathcal{C}$ parts only, not on $\mathcal{C}'$. It outputs $(\mathcal{C},\mathcal{C}')$.
- $\mathsf{Decrypt}(\ell,\mathsf{dk},\mathcal{C},\mathcal{C}')$: one first parses $\mathcal{C}=(C_1,\ldots,C_n)$ and $\mathcal{C}'=(S_1,\ldots,S_n)$, where $C_i=(\mathbf{u}_i,e_i,v_i)$ and $S_i=(\boldsymbol{\alpha}_i,\beta_i,\gamma_i)$, for $i\in[\![1,n]\!]$, computes $\xi=\mathfrak{H}_K(\ell,\mathbf{u}_1,\ldots,\mathbf{u}_n,e_1,\ldots,e_n)$ and checks whether, for $i\in[\![1,n]\!]$, $u_{i,1}^{x_1+\xi y_1}\cdot u_{i,2}^{x_2+\xi y_2} \stackrel{?}{=} v_i$ (but not for the γ_i's). If the equality holds, one computes $M_i=e_i/(u_{i,1}^z)$ and $N_i=\beta_i/(\alpha_{i,1}^z)$, and outputs $(\vec{M}=(M_1,\ldots,M_n),\vec{N}=(N_1,\ldots,N_n))$. Otherwise, one outputs $\perp$.
- $\mathsf{PDecrypt}(\ell,\mathsf{dk},\mathcal{C})$: is a partial decryption algorithm that does as above but working on the $\mathcal{C}$ part only to get $\vec{M}=(M_1,\ldots,M_n)$ or $\perp$.

DCS denotes the particular case where $n=1$:

$$\mathsf{DCS}(\ell,M,N;r,a)=\left(\begin{array}{rcl}\mathcal{C} &=& (\mathbf{u}=(g_1^r,g_2^R),e=M\cdot h^r,v=(cd^{\xi})^r\\ \mathcal{C}' &=& (\boldsymbol{\alpha}=(g_1^a,g_2^a),\beta=N\cdot h^a,\gamma=(cd^{\xi})^a)\end{array}\right),\ \begin{array}{c}\text{where}\\ \xi=\mathfrak{H}_K(\ell,\mathbf{u},e)\end{array}$$

This scheme is indistinguishable against *partial-decryption chosen-ciphertext* attacks, where a partial-decryption oracle only is available, but even when we allow the adversary to choose $\vec{M}$ and $\vec{N}$ in two different steps (see the security game below), under the DDH assumption and if one uses a collision-resistant hash function (and this can be proven in a similar fashion to Theorem 2.6.5, page 48).

2.6.6 Commitment *à la* Lindell

Recently, Lindell [Lin11] proposed a highly efficient UC commitment. Our commitment strongly relies on it, but does not need to be UC secure. We will then show that the decommitment check can be done in an implicit way with an appropriate smooth projective hash function. Basically, the technique consists in encrypting M in $\mathcal{C} = (\mathbf{u}, e, v) = \mathsf{LCS}(\ell, M; r, s)$, also getting $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$, and then encrypting $1_{\mathbb{G}}$ in $\mathcal{C}' = \mathsf{LCS}^*(\ell, 1_{\mathbb{G}}, \xi; a, b)$, with the same ξ. For a given challenge ε, we can see $\mathcal{C} \times \mathcal{C}'^{\varepsilon} = \mathsf{LCS}^*(\ell, M, \xi; r + \varepsilon a, s + \varepsilon b)$, where the computations are done component-wise, as an encryption of M, still using ξ. Note that Lindell used $\mathcal{C}^{\varepsilon} \times \mathcal{C}'$, but our choice seems more natural, since we essentially re-randomize the initial encryption $\mathcal{C}$, but we have to take care of choosing $\varepsilon \neq 0$. It makes use of an equivocable commitment: the Pedersen commitment [Ped92], already defined in Section 2.3.2, page 26.

Description

Our n-message vector commitment, which includes labels, is depicted on Figure 2.3, where the computation between vectors are component-wise. Note that for this commitment scheme, we can use $\vec{\varepsilon} = (\varepsilon, \ldots, \varepsilon)$. For the version with SPHF implicit verification Section 5.3.2, page 100, according to the language, one can have to use independent components $\vec{\varepsilon} \xleftarrow{\$} (\mathbb{Z}_p^*)^n$.

- $\mathsf{Setup}(1^{\mathfrak{K}})$: A group $\mathbb{G}$ of prime order p, with ten independent generators $(g_1, g_2, g_3, h_1, h_2, c_1, c_2, d_1, d_2, \zeta) \xleftarrow{\$} \mathbb{G}^{10}$, a collision-resistant hash function $\mathfrak{H}_K$, and a reversible mapping $\mathcal{G}$ from $\{0,1\}^{\mathfrak{K}}$ to $\mathbb{G}$. One can denote $\mathsf{ek} = (c_1, c_2, d_1, d_1, h_1, h_2, \mathfrak{H}_K)$;
- $\mathsf{Commit}(\ell, \vec{m}; \vec{r}, \vec{s}, \vec{a}, \vec{b}, t)$: for $(\vec{r}, \vec{s}, \vec{a}, \vec{b}, t) \xleftarrow{\$} \mathbb{Z}_p^{4n+1}$, with $\vec{M} = \mathcal{G}(\vec{m})$

  $(\mathcal{C}, \mathcal{C}') \leftarrow n - \mathsf{DLCS}(\ell, \mathsf{ek}, \vec{M}, (1_{\mathbb{G}})^n; \vec{r}, \vec{s}, \vec{a}, \vec{b})$
  $\chi = \mathfrak{H}_K(\vec{m}, \mathcal{C}')$,
  $\mathcal{C}'' = g_1^t \zeta^{\chi}$
  $\mathcal{C}, \mathcal{C}''$ $\longrightarrow$
  $\prod_i \varepsilon_i \stackrel{?}{\neq} 0$
  $\vec{z} = (\vec{r} + \vec{\varepsilon} \times \vec{a}, \vec{s} + \vec{\varepsilon} \times \vec{b})$
  $\vec{\varepsilon}$ $\longleftarrow$ $\varepsilon \xleftarrow{\$} \mathbb{Z}_p^*, \vec{\varepsilon} \leftarrow (\varepsilon, \ldots, \varepsilon)$
- $\mathsf{Decommit}(\ell, \mathcal{C}, \mathcal{C}', \vec{\varepsilon})$:
  $\mathcal{C}', t, \vec{m}, \mathbf{z}$ $\longrightarrow$
  $\vec{M} = \mathcal{G}(\vec{m}), \chi = \mathfrak{H}_K(\vec{m}, \mathcal{C}')$
  compute ξ from $\mathcal{C}, \mathcal{C}'' \stackrel{?}{=} g_1^t \zeta^{\chi}$
  $\mathcal{C} \times \mathcal{C}'^{\vec{\varepsilon}} \stackrel{?}{=} n - \mathsf{LCS}^*(\ell, \vec{M}, \xi; \mathbf{z}_r, \mathbf{z}_s)$

Figure 2.3: $n - \mathsf{DLCS}$ Commitment Scheme

Analysis

Let us briefly show the properties of this commitment:

- Hiding property: $\vec{m}$ is committed in the Pedersen commitment $\mathcal{C}''$, that does not leak any information, and in the $n - \mathsf{LCS}$ encryption $\mathcal{C}$, that is indistinguishable, even with access to the decryption oracle (extractability). This also implies non-malleability.
- Binding property: $\vec{m}$ is committed in the Pedersen commitment $\mathcal{C}''$, that is computationally binding.
- Extractability: using the decryption key of the LCS encryption scheme, one can extract $\vec{M}$ from $\mathcal{C}$, and thus $\vec{m}$. Latter, one has to open the ciphertext $\mathcal{C}\mathcal{C}'^{\vec{\varepsilon}}$ with $\vec{M}' = \mathcal{G}(\vec{m}')$, which can be different from $\vec{M}$ in the case that $\mathcal{C}'$ contains $\vec{N} \neq (1_{\mathbb{G}})^n$. But then $\vec{M}' = \vec{M} \times \vec{N}^{\vec{\varepsilon}}$, that is unpredictable at the commit time of $\mathcal{C}''$. With probability at most $1/p$, one can open the commitment with a value $\vec{m}'$ different from $\vec{m}$, if this value $\vec{m}'$ has been correctly anticipated in $\mathcal{C}''$.
- Equivocability: if one wants to open with $\vec{m}'$, one computes $\vec{M}' = \mathcal{G}(\vec{m}')$, $\vec{N} = (\vec{M}'/\vec{M})^{1/\vec{\varepsilon}}$, encrypts $\vec{N}$ in $\mathcal{C}' = n - \mathsf{LCS}^*(\ell, \vec{N}, \xi; \vec{a}, \vec{b})$, and updates χ and t, using the Pedersen trapdoor for equivocability.

$\star$

Part I

Groth-Sahai Based Protocols

In this part we build several signature protocols while relying heavily on the Groth-Sahai methodology. We can divide this work into two main themes. In the first part, we work around *Group Signatures* [Cv91], while in the second we will focus on a new primitive closely related to *Blind Signatures* [Cha83]. The interesting part is the shift of focus between the first schemes where our concern is to protect the anonymity of a signer and the last ones where we are more concerned about the message confidentiality.

First in order to allow a member of a group to sign anonymously on behalf of a group, we are going to hide his identity with commitments, and then use the Gorth-Sahai methodology to show that the committed user is registered. However we are then interested in adding some additional capabilities to the user and tracing authorities. We first follow the idea of *Traceable Signatures* [KTY04], and allow some delegation of the tracing capacities. This requires us to create an additional trap in the signature, and prove that this trap is correctly generated. Doing so, we also allow user to *step-in* and *step-out* of a signature, this simply means that a user is able to recognize/disavow a signature as his own without having to require an authority to trace it. We further upgrade this scheme to present the first instantiation of *List Signatures* [CSST06] where anyone is able to detect if a signer has signed twice in the same time period, without of course opening the signatures.

Then we build protocols to sign a hidden message and prove its knowledge, once again the Groth-Sahai methodology will be the cornerstone of our construction. To do so we present a neat primitive *Signature on Randomizable Ciphertexts* which allows us to encrypt/commit a message and then sign this object, and then later we show this can be handled like a encryption/commitment to a signature on the plaintext. One of the most important point is that if the user is able to decrypt/decommit, he can now recover a valid signature on the plaintext. This signature is a regular signature with the same randomization property as the starting signature scheme and so we manage to build a round-optimal blind signature scheme which outputs a regular signature. We then further work around this construction to allow partial blindness, and even some homomorphic properties thanks to one of our previous result on the Waters function programmability (cf Section 2.6.4, page 42).

Chapter 3

Group Based Signatures

Contents

In this chapter we use the Groth-Sahai methodology, presented earlier in details in Section 2.4.1, page 29, to solve various open problems while staying compatible with our batching techniques.

In this chapter, we will present the first instantiation of List Signatures in the standard model. We will be using Groth-Sahai methodology to protect the anonymity of the signer while preserving unforgeability.

A first step toward this goal is to achieve traceable group signatures way more efficient than the previous ones, while providing a new functionality. Users are now able to step both in and out, in other words we allow them to acknowledge they signed a signature or prove they did not. We keep the classical requirements of such signatures, we thus have an opener able to know who did a signature and prove it, we have tracing authorities specialized on one identity, so they can only say (and prove) if a signature was done by a specific user $\mathcal{U}_i$, we have a group manager (possibly distinct from the previous authorities) who can manage users in the group.

Contrarily to the rest of this thesis, we will do this set of instantiations in the asymmetric setting. In the symmetric approach, we would have one less trapdoor, we would either have an easy pairing or no kind of pairing at all, while here we want some kind of intermediate result, as we will need some pairing capacities to allow tracing operations, while restraining others to protect the anonymity. Thanks to the missing efficient homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$, the asymmetric setting provides this extra capacities. To be compatible with the methodology we will need to rely on SXDH, even if our non-anonymous scheme would only require XDH.

3.1 Group Signatures

A group signature scheme [Cv91] is a protocol which lets a member of a group individually issue signatures on behalf of the group, in an anonymous but revocable way: an opener is able to revoke anonymity of the actual signer in case of abuse. The members of the group are not trusted. Several steps have been made in the study of those protocols: Bellare, Micciancio and Warinschi [BMW03] gave formal definitions of the security properties of group signatures (the BMW model), and proposed a (unpractical) scheme under general assumptions. Later, Bellare, Shi and Zhang [BSZ05] extended this model to dynamic groups (the BSZ model), emphasizing the importance of unforgeability and anonymity. Boneh, Boyen and Shacham [BBS04] proposed a very efficient group signature scheme using bilinear maps, in the random oracle model, but that does not fit within these models.

Group signatures guarantee *anonymity*, which means that nobody (except the opener) can link the signature to the signer, but also *unlinkability*, which means that one cannot tell whether two signatures

have been produced by the same user.

We use similar notations as [BSZ05] for the BSZ model to define, in a game-based way, the security notions. In a group signature scheme, there are several users, which are all registered in a PKI. We thus assume that each user $\mathcal{U}_i$ owns a pair $(\mathsf{usk}[i], \mathsf{upk}[i])$ certified by the PKI. There is a group manager, also known as *Issuer*, since he will issue certificates to grant access to the group, and an *Opener* that will be able to revoke anonymity, and thus trace back the actual signers. Those two authorities are not necessarily the same. To be precise, a group signature scheme is a sequence of (interactive) protocols:

Group Signature
$\ulcorner$ $\mathsf{GS} = (\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verif}, \mathsf{Open}, \mathsf{Judge})$:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: this algorithm generates the global parameters of the system, the public key pk and the private keys: the master secret key msk given to the group manager, and the opening key skO sent to the opener;
- $\mathsf{Join}(\mathcal{U}_i)$: this is an interactive protocol between a user $\mathcal{U}_i$ (using his secret key $\mathsf{usk}[i]$) and the group manager (using his private key msk). At the end of the protocol, the user obtains a signing key $\mathsf{sk}[i]$ (or group membership certificate), and the group manager adds the user to the registration list, storing some information in $Reg[i]$.
- $\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}[i], m; \mu)$: To sign a message m, the user uses his secret key $\mathsf{sk}[i]$ and some randomness μ, to output a signature σ valid under the group public key pk
- $\mathsf{Verif}(\mathsf{pk}, m, \sigma)$: anybody should be able to verify the validity of the signature σ on the message m, w.r.t. the public key pk. This algorithm thus outputs 1 if the signature is valid, and 0 otherwise.
- $\mathsf{Open}(\mathsf{skO}, \mathsf{pk}, m, \sigma)$: granted the opening key skO, for a valid signature σ w.r.t. the public key pk, the *Opener* can provide the identity signer. It thus outputs the user i, together with a proof Π.
- $\mathsf{Judge}(\mathsf{pk}, m, \sigma, i, \Pi)$: this algorithm publicly checks the claim of the opener.

$\lrcorner$

3.1.1 Security Notions

The *correctness* notion guarantees that honest users should be able to generate valid signatures, and the opener should then be able to get the identity of the signers, and provide a convincing proof for the judge. In the following experiments that formalize the security notions, the adversary can run the Join protocol:

- either through the joinP-oracle (passive join), which means that it creates an honest user for whom it does not know the secret keys: the index i is added to the HU (Honest Users) list. The adversary gets back the public part of the certificate $\mathsf{pk}[i]$;
- or through the joinA-oracle (active join), which means that it interacts with the group manager to create a user it will control: the index i is added to the CU (Corrupted Users) list. The adversary gets back the whole certificate $\mathsf{pk}[i]$, and $\mathsf{sk}[i]$.

For users whose secret keys are known to the adversary, we let the adversary play on their behalf. For honest users, the adversary can interact with them, granted some oracles:

- $\mathsf{corrupt}(i)$, if $i \in \mathsf{HU}$, provides the secret key $\mathsf{sk}[i]$ of this user. The adversary can now control it. The index i is then moved from HU to CU;
- $\mathsf{sign}(i, m)$, if $i \in \mathsf{HU}$, plays as the honest user i would do in the signature process. Then i is appended to the list $\mathcal{S}[m]$.

Traceability and Non-Frameability

Traceability (see Figure 3.1, page 57 (a)) says that nobody should be able to produce a valid signature that cannot be opened in a convincing way. Furthermore, non-frameability (see Figure 3.1, page 57 (b)) guarantees that no dishonest player (even the authorities, i.e. the Group Manager and the Opener, hence the keys msk and skO provided to the adversary) will be able to frame an honest user: an honest user that does not sign a message M should not be convincingly declared as a possible signer, non-frameability also shows that the group manager can not cheat. We thus say that:

(a) Experiment $\mathsf{Exp}^{\mathsf{tr}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}, \mathsf{skO}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(m, \sigma) \leftarrow \mathcal{A}(\mathsf{pk} : \mathsf{joinA}, \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign}, \mathsf{open})$
3. `IF` $\mathsf{Verif}(\mathsf{pk}, m, \sigma) = 0$, `RETURN` 0
4. `IF` $\exists j \notin \mathsf{CU} \cup \mathcal{S}[m]$,
   $\mathsf{Open}(\mathsf{pk}, m, \sigma, \mathsf{skO}) = (j, \Pi)$
   `RETURN` 1
5. `ELSE RETURN` 0

$$\mathsf{Adv}^{\mathsf{tr}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{tr}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K}) = 1]$$

(b) Experiment $\mathsf{Exp}^{\mathsf{nf}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}, \mathsf{skO}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(m, \sigma) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{msk}, \mathsf{skO} : \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign})$
3. `IF` $\mathsf{Verif}(\mathsf{pk}, m, \sigma) = 0$ `RETURN` 0
4. `IF` $\exists i \in \mathsf{HU} \setminus \mathcal{S}[m]$,
   $\mathsf{Open}(\mathsf{pk}, m, \sigma, \mathsf{skO}) = (i, \Pi)$
   `RETURN` 1
5. `ELSE RETURN` 0

$$\mathsf{Adv}^{\mathsf{nf}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{nf}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K}) = 1]$$

Figure 3.1: Unforgeability Notions

- GS is *traceable* if, for any polynomial adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{tr}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K})$ is negligible;
- GS is *non-frameable* if, for any polynomial adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{nf}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K})$ is negligible.

In both games, the adversary generates a signature σ on a message m of its choice. In the latter game, the adversary itself can play the role of the opener, trying to frame an honest user i with a proof Π, hence (i, Π) in its output.

Anonymity: Given two of honest users i_0 and i_1, the adversary should not have any significant advantage in guessing which one of them have issued a valid signature.

Experiment $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}, \mathsf{skO}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(m, i_0, i_1) \leftarrow \mathcal{A}(\mathtt{FIND}, \mathsf{pk}, \mathsf{msk} : \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign})$
3. $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, i_b, m, \mathsf{sk}[i])$
4. $b' \leftarrow \mathcal{A}(\mathtt{GUESS}, \sigma : \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign})$
5. `IF` $i_0 \notin \mathsf{HU}$ `OR` $i_1 \notin \mathsf{HU}$ `RETURN` 0
6. `RETURN` b'

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K}) = 1]$$

Figure 3.2: Anonymity Notions

The adversary can interact with honest users as before (with sign and corrupt), but the challenge signature is generated using the interactive signature protocol Sign, where the adversary plays the role of the corrupted users, but honest users are activated to play their roles.

GS is *anonymous* if, for any polynomial adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{GS},\mathcal{A}}(\mathfrak{K})$ is negligible. The *full-anonymity* notion means that anonymity is guaranteed even if the adversary is granted access to the open-oracle (excepted on the challenge signature), however we won't achieve it in this chapter.

3.2 Traceable Signatures

3.2.1 Security Notions for Traceable Signatures

This model supersedes regular group signatures, so we will continue to use notations from the BSZ model and extend them when required. We follow the original model from traceable signatures [KTY04] with some improvements, but with similar notations and terminology. We will mainly stress the differences.

In a traceable signature scheme, there are several users, which are all registered in a PKI. We thus assume that any user $\mathcal{U}_i$ owns a pair $(\mathsf{usk}[i], \mathsf{upk}[i])$ of secret and public keys, certified by the PKI. There are several authorities:

- the *Group Manager*: it issues certificates for users to grant access to the group.
- the *Opener*: it is able to *open* or *trace* any signature. The former means that it can learn who is the actual signer of a given signature while the latter decides, on a given signature and an alleged signer, whether the signature has really been generated by this signer or not. It is also able to delegate the latter tracing capability but for specific users only. To this aim, it *reveals* a trapdoor to a *Sub-Opener*. The latter gets the ability to trace a specific user only (decide whether the signer

associated to the trapdoor is the actual signer of a signature) without learning anything about the other users.

Those two authorities can be the same party, as in [LY09]. However, since this is a stronger model, but we prefer to separate the roles as in [BSZ05].

In the initial model, users also have the capability to *Claim* a signature, i.e. they are able to publicly confirm they are the author of a given signature. We enhance the functionalities with a *Deny* algorithm, that allows a user to prove that he is not the actual author of a given signature. Both are combined in a Step algorithm, with $\mathsf{Step}-\mathsf{in}$ and $\mathsf{Step}-\mathsf{out}$ procedures to confirm and deny a signature respectively, using the signing key only.

A Traceable signature scheme (with stepping capacities) is defined by a sequence of (interactive) protocols, $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verif}, \mathsf{Open}, \mathsf{Reveal}, \mathsf{Trace}, \mathsf{Step})$:

While in many security models, there is an additional party called "Judge" that verifies all the claims, we are going to omit it in the following, as anyone will be able to directly verify the various claims.

While $\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verif}, \mathsf{Open}$ are similar to those in Group Signatures, $\mathsf{Reveal}, \mathsf{Trace}, \mathsf{Step}$ are new:

- $\mathsf{Reveal}(\mathsf{pk}, i, \mathsf{skO})$: This algorithm, with input skO and a target user i, outputs a tracing key $\mathsf{tk}[i]$ specific to the user i, together with a proof $\Pi_{\mathcal{E}}$ confirming this $\mathsf{tk}[i]$ is indeed a tracing key of the user i.
- $\mathsf{Trace}(\mathsf{pk}, m, \sigma, \mathsf{tk}[i])$: Using the sup-opener key $\mathsf{tk}[i]$ for user i, this algorithm outputs 1 iff σ is a valid signature produced by i, together with a proof Π_{sO} confirming the decision.
- $\mathsf{Step}(\mathsf{pk}, m, \sigma, \mathsf{sk}[i])$: Using the user's secret key $\mathsf{sk}[i]$, this algorithm outputs 1 iff σ if a valid signature produced by i, together with a proof Π_c confirming the claim.

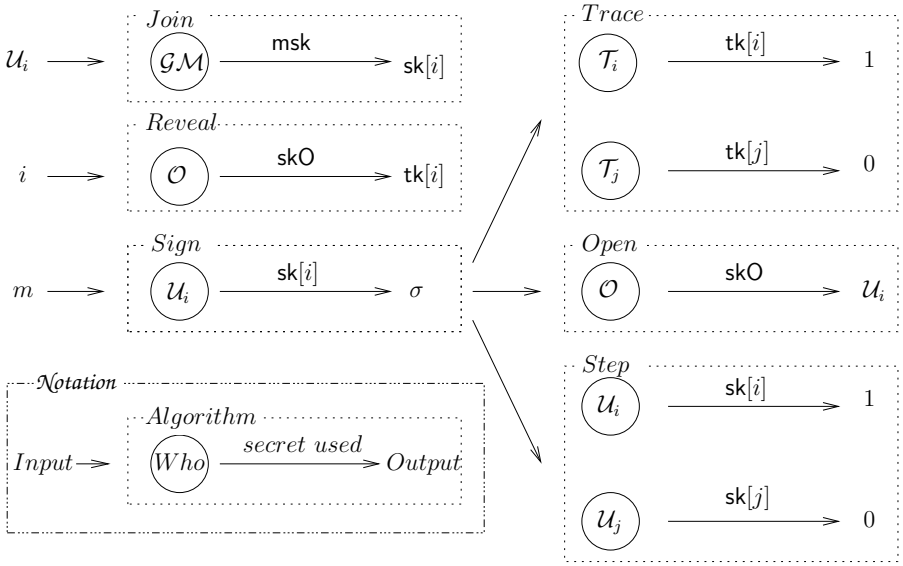


Figure 3.3: An Extended Traceable Signature Scheme

Traceable Signature

Traceable signatures were introduced by Kiayias, Tsiounis and Yung in [KTY04] as an improvement of group signatures (defined in [Cv91]). In addition to the classical properties of a group signature scheme, that allows users to sign in the name of the group, while the opener only is able to trace back the actual signer, traceable signatures allow the opener to delegate the tracing decision for a specific user without revoking the anonymity of the other users: the opener can delegate its tracing capability to sub-openers, but against specific signers without letting them trace other users. This gives two crucial advantages: on the one hand tracing agents (sub-openers) can run in parallel; on the other hand, honest users do not have to fear for their anonymity if authorities are looking for signatures produced by misbehaving

users only. This is in the same vein as searchable encryption [ABC+05], where a trapdoor, specific to a keyword, allows to decide whether a ciphertext contains this keyword or not, and provides no information about ciphertexts related to other keywords.

The first efficient traceable signatures, provably secure in the standard model, were introduced by Libert and Yung in [LY09].

Correctness: The *correctness* notion guarantees that honest users should be able to generate valid signatures, those notions are direct extensions of the classic ones with an additional authority and also consistency between open, trace and step algorithms. More precisely the correctness guarantees that

- a message signed by an honest user i should
  - successfully pass the verification process;
  - open to i;
  - lead to a positive answer for the trace and step procedures under user i's related keys;
- as the traceability property of group signatures, for any valid signature σ, the opening algorithm should designate some user. And the latter should be accepted by the trace and step procedures.

In the following experiments that formalize the security notions, the adversary can run the Join protocol, either passively (receives only public values, as seen by an eavesdropper) or actively (receives all the values, as the legitimate user):

- either through the joinP-oracle (passive join), which means that it creates an honest user for whom it does not know the secret keys: the index i is added to the HU (Honest Users) list;
- or through the joinA-oracle (active join), which means that it interacts with the group manager to create a user it will control: the index i is added to the CU (Corrupted Users) list.

The adversary is given the master key (the group manager is corrupted), and so does not need access to the joinA oracle since it can simulate it by itself, to create corrupted users (that are not necessarily in CU). After a user is created, the adversary plays the role of corrupted users, and can interact with honest users, granted the previous oracles, and also new ones:

- $\mathsf{open}(m, \sigma)$, if (m, σ) is valid, returns the identity i of the signer. Then (i, m, σ) is appended to the list $\mathcal{O}$ of opened signatures;
- $\mathsf{reveal}(i)$, if $i \in \mathsf{HU}$, returns the tracing key $\mathsf{tk}[i]$ for the user i. Then i is appended to the list $\mathcal{R}$ of the revealed users;
- $\mathsf{tr}(i, m, \sigma)$, if $i \in \mathsf{HU}$ and (m, σ) is valid, returns 1 iff i is the signer who made σ on m is i. Then (i, m, σ) is appended to the list $\mathcal{T}$ of traced signatures;
- $\mathsf{step}(i, m, \sigma)$, if $i \in \mathsf{HU}$, plays as the honest user i would do to step in/out of the signature σ on message m.

For a corrupted user i, with the secret key $\mathsf{sk}[i]$, the adversary can run itself the Step and Trace procedures, and or course sign too. We thus have the following sets:

- I, the set of registered users, which is the disjunction of HU and CU the honest and dishonest users respectively;
- $\mathcal{S}$, the list of generated signatures (i, m, σ), and $\mathcal{S}[m] = \{i | (i, m, \sigma) \in \mathcal{S}\}$;
- $\mathcal{O}$, the list of opened signatures (i, m, σ), and $\mathcal{O}[m] = \{i | (i, m, \sigma) \in \mathcal{O}\}$;
- $\mathcal{R}$, the list of revealed users i;
- $\mathcal{T}$, the list of traced signatures (i, m, σ), and $\mathcal{T}[m] = \{i | (i, m, \sigma) \in \mathcal{T}\}$.

A signature is identified by a user-message pair and not σ itself in those sets because we do not expect strong unforgeability. In our instantiations, signatures will be re-randomizable: it is easy for anyone to produce a new valid signature σ' on a message m from a previous one σ, but on the same message. The subsequent relaxation on the security has already been used in [LY09] for traceable signatures.

Soundness: This is the main security notion that defines two unforgeability properties. The security games are shortened thanks to the correctness which implies that the opening, the tracing and the stepping processes are consistent:

- Misidentification, which means that the adversary should not be able to produce a non-trivial valid signature that could not be opened to a user under its control. The adversary wins if Open either accuses an unknown user or has an invalid proof, so returning $\perp$. (see Figure 3.4, page 60 (a));

- Non-Frameability, which means that the adversary should not be able to produce a non-trivial valid signature corresponding (that opens) to an honest user even if the authorities are corrupted (see Figure 3.4, page 60 (b));

TS is *Sound* if, for any polynomial adversary $\mathcal{A}$, both advantages $\mathsf{Adv}^{\mathsf{Misl}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K})$ and $\mathsf{Adv}^{\mathsf{nf}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K})$ are negligible. The first notion (Misidentification) guarantees traceability (the Open algorithm always succeeds

(a) Experiment $\mathsf{Exp}^{\mathsf{Misl}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}, \mathsf{skO}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(m, \sigma) \leftarrow \mathcal{A}(\mathsf{pk} : \mathsf{joinP}, \mathsf{joinA}, \mathsf{corrupt}, \mathsf{sign}, \mathsf{reveal})$
3. IF $\mathsf{Verif}(\mathsf{pk}, m, \sigma) = 0$, RETURN 0
4. IF $\mathsf{Open}(\mathsf{pk}, m, \sigma, \mathsf{skO}) = \perp$, RETURN 1
5. IF $\exists j \notin \mathsf{CU} \cup \mathcal{S}[m]$,
$\mathsf{Open}(\mathsf{pk}, m, \sigma, \mathsf{skO}) = (j, \Pi)$
RETURN 1
6. ELSE RETURN 0

$$\mathsf{Adv}^{\mathsf{Misl}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{Misl}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K}) = 1]$$

(b) Experiment $\mathsf{Exp}^{\mathsf{nf}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}, \mathsf{skO}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(m, \sigma) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{msk}, \mathsf{skO} : \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign})$
3. IF $\mathsf{Verif}(\mathsf{pk}, m, \sigma) = 0$ RETURN 0
4. IF $\exists i \in \mathsf{HU} \setminus \mathcal{S}[m]$,
$\mathsf{Open}(\mathsf{pk}, m, \sigma, \mathsf{skO}) = (i, \Pi)$
RETURN 1
5. ELSE RETURN 0

$$\mathsf{Adv}^{\mathsf{nf}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{nf}}_{\mathsf{nf},\mathcal{A}}(\mathfrak{K}) = 1]$$

Figure 3.4: Security Notions: Soundness

on valid signatures) but also honest users cannot be framed when the group manager is honest. The second one (non-frameability) is somewhat stronger since it allows the group manager to be corrupted, but would not guarantee by itself traceability.

Anonymity: We now address the privacy concerns. For two distinct signers i_0, i_1, chosen by the adversary, the latter should not have any significant advantage in guessing if the issued signature comes from i_0 or i_1. We can consider either a quite strong anonymity notion (usually named full-anonymity) where the adversary is allowed to query the opening oracle (resp. tracing, stepping) on any signatures, excepted signatures that are equivalent to the challenge signature with respect to the signers i_0 or i_1; or the classical anonymity notion, where open, tr and of course reveal are not available to the adversary. TS is *anonymous* if, for any polynomial adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K})$ is negligible (see Figure 3.5, page 60).

Experiment $\mathsf{Exp}^{\mathsf{anon}_b}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}, \mathsf{skO}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(m, i_0, i_1) \leftarrow \mathcal{A}(\mathtt{FIND}, \mathsf{pk}, \mathsf{msk} : \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign}, \mathsf{open}^*, \mathsf{tr}^*, \mathsf{reveal}^*, \mathsf{step}^*)$
3. $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, i, m, \mathsf{sk}[i_b])$
4. $b' \leftarrow \mathcal{A}(\mathtt{GUESS}, \sigma : \mathsf{joinA}, \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign}, \mathsf{open}^*, \mathsf{tr}^*, \mathsf{reveal}^*, \mathsf{step}^*)$
5. IF $i_0 \notin \mathsf{HU} \setminus (\mathcal{R} \cup \mathcal{T}[m] \cup \mathcal{O}[m] \cup \mathcal{S}[m])$ OR $i_1 \notin \mathsf{HU} \setminus (\mathcal{R} \cup \mathcal{T}[m] \cup \mathcal{O}[m] \cup \mathcal{S}[m])$ RETURN 0
6. ELSE RETURN b'

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathsf{TS},\mathcal{A}}(\mathfrak{K}) = 1]$$

Figure 3.5: Security Notions: Anonymity ($\mathsf{open}^* = \mathsf{tr}^* = \mathsf{reveal}^* = \mathsf{step}^* = \emptyset$) and Full-Anonymity ($\mathsf{open}^* = \mathsf{open}, \mathsf{tr}^* = \mathsf{tr}, \mathsf{reveal}^* = \mathsf{reveal}, \mathsf{step}^* = \mathsf{step}$)

In the following however, our scheme will only fulfil the anonymity requirement, as Groth-Sahai commitments don't give us enough freedom in the simulation, precisely they won't be extractable and equivocable simultaneously.

3.2.2 Traceable Group Signatures with Stepping Capabilities

Tools

Here we recall, two classical instantiations of tools we will use specifically in the two following schemes. First, the Dodis-Yampolskiy Verifiable Random Function, given ℓ results of the functions, one should not be able to predict the next one, however a honest user should be able to produce a proof showing he has indeed computed the correct value, and then a certificate, basically something proving a user has been registered by a specific authority.

Those new tools will rely on those security hypotheses:

q-Decisional Diffie-Hellman Inverse in $\mathbb{G}_1$ (q-DDHI [BB04, DY05])
⌜ Let $\mathbb{G}_1$ be a cyclic group of order p generated by g_1. The q-DDHI problem consists, given a tuple $(g_1, g_1^{\gamma}, \ldots, g_1^{\gamma^q}) \in \mathbb{G}_1^{q+1}$ and $D \in \mathbb{G}_1$, in deciding whether $D = g_1^{1/\gamma}$ or not. ⌟

q-Hybrid Hidden Strong Diffie-Hellman in $\mathbb{G}_1, \mathbb{G}_2$ (q-HHSDH)
⌜ Let $\mathbb{G}_1, \mathbb{G}_2$ be multiplicative cyclic groups of order p generated by g_1, g_2 respectively. The q-HHSDH problem consists, given $(g_1, k, g_2, g_2^{\gamma})$ and several partly hidden tuples $(g_1^{x_i}, g_2^{x_i}, y_i, (kg_1^{y_i})^{1/(\gamma+x_i)})_{i\in[1,q]}$, in computing $(g_1^x, g_2^x, g_1^y, g_2^y, (kg_1^y)^{1/\gamma+x})$ for a new pair (x, y). ⌟

About that last assumption: intuitively, under KEA[1], it can be reduced to a standard q-SDH (Under KEA, the reduction to q-SDH is similar to the one in [DP06]). It follows the idea of the BB-SDH introduced in [BCC+08]. However, in our construction neither the scalar given is the one involved directly in the SDH part, nor we give a second group element raised to the power γ. Therefore this new assumption seems to remain reasonable.

Pseudo Random Function: We will use a variation of the Dodis-Yampolskiy VRF [DY05], introduced in [CHL05a]. It basically states that for a polynomial number of scalars z_i, and a pair $(g_1, g_1^x) \in \mathbb{G}_1^2$, the values $g_1^{1/(x+z_i)}$ look random and independent. We will use this property to build our identifiers. In the proof of anonymity, the simulator will be able to choose the z_i prior to any interaction with the adversary so we rely in the framework where the VRF is secure under the $q-$DDHI assumption.

A verification of a correct computation can be made, if a user can produce a valid g_2^x with respect to g_1^x, and $g_2^{z_i}$ such that:

$$e(g_1^{1/(x+z_i)}, g_2^x g_2^{z_i}) = e(g_1, g_2) \quad \text{and} \quad e(g_1^x, g_2) = e(g_1, g_2^x).$$

Certification: Since our new primitive is quite related to group signatures, we also introduce the BBS-like certification [BBS04] proposed by Delerablée and Pointcheval [DP06], in order to achieve non-frameability.

During the Setup, the group manager chooses an additional generator k_1 of $\mathbb{G}_1$, and a master secret key $\mathsf{msk} = \gamma \in \mathbb{Z}_p$. It sets the group public key as$(g_1, g_2, k_1, \Omega = g_2^{\gamma})$. During the Join procedure, the authority chooses an x_i for the user, and they choose together y_i (so that it is unknown to the group manager, but known to the user: his secret key). After an interactive process, the user gets his certificate, $(A_i = (k_1 g_1^{y_i})^{1/(\gamma+x_i)}, g_1^{x_i}, g_2^{x_i}, y_i)$, where the verification consists in:

$$e(A_i, \Omega g_2^{x_i}) \stackrel{?}{=} e(k_1, g_2) \cdot e(g_1, g_2)^{y_i} \quad \text{and} \quad e(g_1^{x_i}, g_2) \stackrel{?}{=} e(g_1, g_2^{x_i}).$$

We use this certification, because contrarily to the simple BBS certificate this one provides non-frameability, in the sense that the authority does not learn the whole final certificate, namely y_i is known solely by the user.

Traceable Signature Instantiation

We are first going to describe our construction, without anonymity. The latter security property will be achieved granted commitments and proofs of validity, that will be easy and efficient since everything fits within the Groth-Sahai methodology.

[1] The Knowledge-of-Exponent Assumption was first proposed in [Dam92]. Informally, KEA says that if an adversary takes g, h generated by Gen and outputs a DDH tuple (g^a, h^a), then it must know a. This hypothesis is KEA-I

$\mathsf{Setup}(1^{\mathfrak{K}})$**:** The system generates a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$. One also chooses a generator k_1 of $\mathbb{G}_1$, and also independent generators $(u_i) \xleftarrow{\$} \mathbb{G}_1^{k+1}$, where k is a polynomial in $\mathfrak{K}$ to define the Waters function $\mathcal{F}$.

The group manager chooses a scalar $\gamma \xleftarrow{\$} \mathbb{Z}_p$ for the master key $\mathsf{msk} = \gamma$, and computes $\Omega = g_2^{\gamma}$. The opener produces a computationally binding Groth-Sahai environment with $\mathsf{ek} \in \mathbb{Z}_p^2$ as an extraction key, we will only use the extraction key in $\mathbb{G}_2$ in the real world, we will note it α. We will note the commitment key ck, and commitments under it: $\mathcal{C}$. (Technically, it leads to double ElGamal encryptions). The public key is then $\mathsf{pk} = (g_1, g_2, k_1, \mathcal{F}, \Omega, \mathsf{ck})$.

$\mathsf{Join}(\mathcal{U}_i)$**:** In order to join the system, a user $\mathcal{U}_i$, with a pair of keys $(\mathsf{usk}[i], \mathsf{upk}[i])$ in the PKI, interacts with the group manager (similarly to [DP06]):

- $\mathcal{U}_i$ chooses a random $y_i' \in \mathbb{Z}_p$, computes and sends $Y_i' = g_1^{y_i'}$, an extractable commitment of y_i' with a proof of consistency. Actually the trapdoor of the commitment will not be known to anybody, except to our simulator in the security proof to be able to extract y_i'.
- The group manager chooses a new $x_i \in \mathbb{Z}_p$ and a random $y_i'' \xleftarrow{\$} \mathbb{Z}_p$, computes and sends y_i'', $A_i = (k_1 Y_i' Y_i'')^{1/(\gamma + x_i)}$ and $X_{i,2} = g_2^{x_i}$ where $Y_i'' = g_1^{y_i''}$;
- $\mathcal{U}_i$ checks whether $e(A_i, \Omega g_2^{x_i}) \stackrel{?}{=} e(k_1, g_2) \cdot e(g_1, g_2)^{y_i' + y_i''}$. He can then compute $y_i = y_i' + y_i''$. And so we have $e(A_i, \Omega X_{i,1}) = e(k_1, g_2) \cdot e(g_1, g_2)^{y_i}$. He produces a commitment of $\mathsf{tk}[i] = g_2^{y_i} : \mathbf{e}_i = \mathcal{C}(g_2^{y_i})$ and a proof of consistency $\pi_J[i]$, and signs $(A_i, X_{i,2}, g_1^{y_i}, \mathbf{e}_i, \pi_J[i])$ under $\mathsf{usk}[i]$ into $\mathbf{s}_i$.
- The group manager verifies s_i under $\mathsf{upk}[i]$ and the given proof, and appends the tuple $(i, \mathsf{upk}[i], A_i, \mathbf{X}_i, g_1^{y_i}, \mathbf{e}_i, \pi_J[i], \mathbf{s}_i)$ to *Reg*. He can then send the last part of the certificate $X_{i,1} = g_1^{x_i}$.
- $\mathcal{U}_i$ thus checks, if this new value is consistent with $g_2^{x_i}$ (i.e. $e(X_{i,1}, g_2) \stackrel{?}{=} e(g_1, X_{i,2})$), and then owns a valid certificate $(A_i, \mathbf{X}_i, y_i)$, where $\mathsf{sk}[i] = y_i$ is known to him only. The secrecy of y_i will be enough for the overall security. Note that if $X_{i,1}$ is invalid, one can ask for it again. In any case, the group manager cannot frame the user, but just do a denial of service attack, which is unavoidable. We expect the *Reg* array to be constantly certified, i.e. , we expect the Group Manager to sign every rows. (This will only be required in our step in/out process)

At this stage, $Reg[i] = \{(i, \mathsf{upk}[i], A_i, \mathbf{X}_i, g_1^{y_i}, \mathbf{e}_i, \pi_J[i], \mathbf{s}_i)\}$, and $\mathsf{sk}[i] = y_i$.

$\mathsf{Sign}(\mathsf{sk}[i], m; s, z)$**:** When a user i wants to sign a message m, he computes the signature of m under his private key $\mathsf{sk}[i]$. The underlined element are the core of his certificate, any of those can be used to determine the user identity. First, he picks two scalars $s, z \xleftarrow{\$} \mathbb{Z}_p$, he creates an ephemeral $\mathsf{ID}(y_i, z) = g_1^{1/(z+y_i)}$, and publishes σ:

$$(\sigma_0 = \mathsf{ID}(y_i, z), \sigma_1 = \underline{\mathbf{X}_i}, \sigma_2 = \underline{y_i}, \sigma_3 = \underline{A_i}, \sigma_4 = (g_1^z, g_2^z), \sigma_5 = k_1^z \mathcal{F}(m)^s, \sigma_6 = (g_1^s, g_2^s))$$

that satisfy the relations:

$$\begin{aligned} e(\sigma_0, \sigma_{4,2} g_2^{\sigma_2}) &= e(g_1, g_2) & e(\underline{\sigma_{1,1}}, g_2) &= e(g_1, \underline{\sigma_{1,2}}) \\ e(\underline{\sigma_3}, \Omega \underline{\sigma_{1,2}}) &= e(k_1, g_2) \cdot e(g_1, g_2^{\sigma_2}) & e(\underline{g_1^{\sigma_2}}, g_2) &= e(g_1, \underline{g_2^{\sigma_2}}) \\ e(\sigma_5, g_2) &= e(k_1, \sigma_{4,2}) \cdot e(\mathcal{F}(m), \sigma_6) & e(\sigma_{4,1}, g_2) &= e(g_1, \sigma_{4,2}) \\ e(\sigma_{6,1}, g_2) &= e(g_1, \sigma_{6,2}). \end{aligned}$$

Basically, σ_0 is a certificate of the public key σ_4, and (σ_5, σ_6) is a Waters' signature of m under this key, while the three other elements are just the complete certificate. As explained above, for the sake of clarity, we started with a non-anonymous scheme. To achieve anonymity, some of these tuples are thereafter committed, but only those that can be linked to a user, basically those we underlined. As shown in the equations σ_2 is a scalar, but needs to be committed in both groups, which will be perfect for the following proofs as it will be enough to extract both g_1^y and g_2^y (as required by the previous computational assumption):

$$\sigma = (\sigma_0, \mathcal{C}(\sigma_1), \mathcal{C}(\sigma_2), \mathcal{C}(\sigma_3), \sigma_4, \sigma_5, \sigma_6)$$

We add the corresponding Groth-Sahai proofs to prove the validity of the previous pairing equations. The second and third equations, checking if $\mathbf{X}_i$ is well-formed and if A_i is well-formed are regular pairing

product so need each 4 elements in each group, the first one (ID is well formed) is a Linear Pairing Product, so needs only 2 extra elements in $\mathbb{G}_1$, the fourth one (the same y_i is committed in two bases) is a quadratic equation and so can be proven with 2 elements in each group. The last ones do not use any committed data and so can be directly checked. Overall we will need 22 group elements in $\mathbb{G}_1$ and 16 in $\mathbb{G}_2$, which is far under the 83 required in the Libert-Yung construction. Especially if we consider elements in $\mathbb{G}_1$ to be half the size of those in $\mathbb{G}_2$. (In a standard instantiation an element in $\mathbb{G}_2$ is at most as big as an element in a DLin-based instantiation, so our scheme has only 33 % of the original communication cost.)

Verif(pk, m, σ): One simply has to verify if all the pairing equations hold.

Open$(\mathsf{pk}, m, \sigma, \alpha)$: The Opener just opens the commitment of A_i in σ_3, and then outputs a Groth-Sahai proof of knowledge of an extraction key α such that $e(\sigma_{3,1}, g_2) = e(A_i, g_2) \cdot e(\sigma_{3,2}, g_2^{\alpha})$. He checks $\mathbf{s}_i$, and depending on its consistency blames the user U_i or the Group Manager or $\perp$. This is a linear multi-scalar multiplication in $\mathbb{G}_1$ and so the proof Π is composed of only 1 group element in $\mathbb{G}_1$ and is publicly verifiable.

Reveal(pk, i, α): The Opener verifies $\pi_J[i]$, and $\mathbf{s}_i$ in *Reg* and uses α to decrypt e_i and extracts the tracing key: $\mathsf{tk}[i] = g_2^{y_i}$. He then send it to the sub-opener together with a publicly verifiable proof showing that $\mathsf{tk}[i]$ is a valid decryption of $\mathbf{e}_i$. (Again a linear Multi-Scalar Multiplication but in $\mathbb{G}_2$ this time).

Trace$(\mathsf{pk}, m, \sigma, \mathsf{tk}[i])$: The Sub-Opener picks $\delta \xleftarrow{\$} \mathbb{Z}_p$ and outputs a blinded tuple $(c_1 = \mathsf{tk}[i]^{\delta}, c_2 = \sigma_{4,2}^{\delta}, c_3 = g_2^{\delta})$ and the target user i. Anyone can then check the validity of the tuple that should satisfy:

$$e(g_1^{y_i}, c_3) = e(g_1, c_1) \quad \text{and} \quad e(\sigma_{4,1}, c_3) = e(g_1, c_2)$$

and then know the result of the trace process from the test $e(\sigma_0, c_2 c_1) = e(g_1, c_3)$. We recall that $g_1^{y_i}$ is included in *Reg*$[i]$ and is thus considered public, and that each row in *Reg* is signed, so the group manager authority acknowledged that.

Step$(\mathsf{pk}, m, \sigma, \mathsf{sk}[i])$: To step in or out, a user picks a random δ, and publishes a similar blinded tuple $(c_1 = g_2^{\delta y_i}, c_2 = \sigma_{4,2}^{\delta}, c_3 = g_2^{\delta})$ and i. Anyone can then check the validity of this tuple as above:

$$e(g_1^{y_i}, c_3) = e(g_1, c_1) \quad \text{and} \quad e(\sigma_{4,1}, c_3) = e(g_1, c_2)$$

and then if the step is in or out with: $e(\sigma_0, c_2 c_1) \stackrel{?}{=} e(g_1, c_3)$.

Another way to step-in or out of a given signature, less efficient but which induces the knowledge of y_i: a user just does the same thing as a sub-opener, together with either an extra signature involving his private key or a bit-per-bit proof of knowledge of y_i. This adds an extra-property outside the scope of our model, which proves that the step in/out has really been initiated by the user itself (and not a tracing authority).

Security

We will rely on four assumptions, SXDH we already detailed for the indistinguishably of our commitments, CDH^+ we detailed when we presented our asymmetric variant of Waters for the unforgeability of the signature, the asymmetrical q-DDHI assumption for the pseudo-randomness of the the Dodis-Yampolskiy VRF, and the q-HHSDH for the certificate and so the non-frameability.

Correctness: Correctness of our scheme is guaranteed by the perfect soundness of the Groth-Sahai proofs in the signature and in the output of procedures Open, Trace and Step.

Anonymity: We now study the anonymity property.

Theorem 3.2.1 *If there exists an adversary $\mathcal{A}$ that can break the anonymity property of the scheme, then there exists an adversary $\mathcal{B}$ that can break the $\ell -$ DDHI problem in $\mathbb{G}_1$ or the* SXDH *assumption, where ℓ is the maximal number of signing queries for a user.*

Intuitively, we will show first that a perfectly binding instantiation can be supplanted by a perfectly hiding one without consequence under SXDH, in this case only the ID can possibly link information, but under DDHI it does not.

Proof: Let us assume that an adversary is able to break the anonymity property of our scheme. It means that in the anonymity game, he has a non-negligible advantage $\epsilon > 0$ to distinguish $G^{(0)}$ where $b = 0$ from $G^{(1)}$ where $b = 1$. We start our sequence of games from $G^{(0)}$, denoted $\mathcal{G}_0$.

$\mathcal{G}_1$ The simulator $\mathcal{B}$ is given a challenge $A = (g, g^y, \ldots, g^{y^\ell}) \in \mathbb{G}_1^{\ell+1}$ and $D = g^{1/y}$, for an unknown y. $\mathcal{B}$ first chooses different random values $z^*, z_1, \ldots, z_{\ell-1} \in \mathbb{Z}_p^\ell$, which completely define the polynomial $P = \prod_{i=1}^{\ell-1}(X + z_i)$, of degree $\ell - 1$. Granted the above challenge, $\mathcal{B}$ can compute $g_1 = g^{P(y)}$, and the rest of the public key is computed as in a normal scheme. In particular, one knows the master secret key γ.

The future challenge user i_0 will virtually have $y_{i_0} = \mathsf{sk}[i_0] = y - z^*$. $\mathcal{B}$ can compute $g^{y_{i_0}} = g_1^y / g_1^{z^t}$ (from the input A without using y, even if we know it here). The public certificate (in the *Reg* list) for the challenge user is $(g_1^{x_{i_0}}, g_2^{x_{i_0}}, g_1^y / g_1^{z^t}, (k_1 g_1^y / g_1^{z^t})^{1/(\gamma+x)})$, plus some proofs and signatures. $\mathcal{B}$ is also given Groth-Sahai commitment keys (extraction key is unknown, hence the classical notion of anonymity and not full-anonymity). This Setup is indistinguishable from the real one since all the keys are generated as in the real game. Because of the knowledge of the master secret key, $\mathcal{B}$ easily answers any join queries, both active and passive. However he still has to guess i_0, which is correct with probability $1/n$, where n is the total number of passive join queries. It can also answer any corruption, that should not happen for the challenge user, even if we know y in this game.

As our simulator is able to know all x_i and all y_i for registered users (except the challenge one), he will be able to answer signing queries as users would do. For the challenge user, on the j-th signing query, he computes $\sigma_0 = g_1^{1/(\mathsf{sk}[i]+z_j)} = g^{\prod_{i \neq j}(y+z_i)}$, which can be done from the challenge input A, the rest is done as in the real game using y and z_j as ephemeral random. For the challenge signing query, he does the same has above with the ephemeral value z^*, and the expected ID is $g_1^{1/(\mathsf{sk}[i]+z^*)} = g^{P(y)/y} = g^{Q(y)} g^{\prod(z_i)/y}$, where $Q = (\prod_{i=1}^{\ell}(X + z_i) - \prod(z_i))/X$ is a polynomial of degree $\ell - 1$ and thus $g^{Q(y)}$ can be computed from the instance. He thus outputs $\sigma_0 = g^{Q(y)} \cdot D^{\prod(z_i)}$. Since $D = g^{1/y}$, the signature is similar to the above one, and so is indistinguishable from a real signature.

$\mathcal{G}_2$ We then modify the game, and we initialize Groth-Sahai commitment keys in a perfectly hiding setting leading to perfectly WI commitments and proofs. This game is indistinguishable from the previous one under the SXDH.

$\mathcal{G}_3$ For the challenge user signing queries, we use random commitments for $\sigma_1, \sigma_2, \sigma_3$. As we are in the perfectly hiding setting, this is indistinguishable anyway.

$\mathcal{G}_4$ We do not know anymore y, that we did not use anyway, and thus this game is perfectly indistinguishable from the previous one.

$\mathcal{G}_5$ D is a random value, which is indistinguishable from the real one under the $\ell-$DDHI assumption as we only have a polynomial number of z_i in input like in the Dodis-Yampolskiy PRF: the challenge signature does not depend anymore on the challenge user.

To complete the proof, we should make the same sequence again, starting from $\mathcal{G}_0$' that is $G^{(1)}$, up to $\mathcal{G}_5$', that is perfectly indistinguishable from $\mathcal{G}_5$, hence the computational indistinguishability between $\mathcal{G}_0' = G^{(1)}$ and $\mathcal{G}_0 = G^{(0)}$. □

Soundness: Within the soundness analysis, we prove traceability and non-frameability.

Theorem 3.2.2 *If there exists an adversary $\mathcal{A}$ against the soundness of the scheme, then we can build an adversary $\mathcal{B}$ that can either break a computational problem ($Q-$HHSDH, $Q'-$HSDH, or CDH$^+$), or a decisional one (the 1-DDHI, or the SXDH), where Q is maximal number of users, and Q' is the maximal number of signing queries for a user.*

Note that the 1-DDHI is equivalent to the Decisional Square Diffie-Hellman, since we have a sequence $(g_1, g_1^\gamma, g_1^\delta)$ where one has to decide whether $\delta = 1/\gamma$, which can be written $(G = g_1^\gamma, G^\alpha = g_1, G^\beta = g_1^\delta)$, where $\alpha = 1/\gamma$, and $\beta = \delta/\gamma$, and one has to decide whether $\delta = 1/\gamma$, and thus whether $\beta = \alpha^2$.

Proof: Non-frameability and misidentification are very closely related, we will treat both simultaneously, there are three ways to cheat the soundness of our scheme: either by creating a new certificate ($\mathcal{G}_1$) which induces a misidentification attack, or by using an existing certificate but on a new message ($\mathcal{G}_{2a,2b}$) which breaks the non-frameability.

We study the security of the unencrypted version of our scheme (because of the perfect soundness of the Groth-Sahai proofs in the perfectly binding setting, and extractability of the commitments). We will construct three different games, in the first one ($\mathcal{G}_1$), we assume the adversary is able to forge a signature by generating a new certificate (a new tuple $(\sigma_1, \sigma_2, \sigma_3)$), in the second one ($\mathcal{G}_2a$) the adversary is able to forge a new σ_0 and so break the tracing or step procedure, in the last game ($\mathcal{G}_2b$) the adversary forges a new Waters signature (a new tuple $(\sigma_4, \sigma_5, \sigma_6)$).

$\mathcal{G}_1$ Let us be given a $Q-\mathsf{HHSDH}$ challenge (g_1, k, g_2, Ω), $(g_1^{x_i}, g_2^{x_i}, y_i, A_i)_{i\in[\![1,Q]\!]}$. We build an adversary $\mathcal{B}$ able to solve this challenge, from $\mathcal{A}$ that breaks the soundness of our scheme by generating a new tuple $(\sigma_1, \sigma_2, \sigma_3)$. $\mathcal{B}$ generates the commitment keys, so that he knows the trapdoor, and publishes the group public key $(g_1, g_2, k_1, \mathcal{F}, \Omega, \mathcal{C})$. To answer the i-th join queries, if this is an active join, $\mathcal{B}$ extracts y_i' and adapts his y_i'' so that $y_i' + y_i'' = y_i$, if it is a passive join, $\mathcal{B}$ directly chooses y_i. As he knows the extraction key, he can give the opening key skO to the adversary.

After at most Q join queries, $\mathcal{A}$ is able to output a new signature with a new certificate tuple with non-negligible probability. As $\mathcal{B}$ knows the trapdoor of the commitment scheme, he can obtain $(g_1^x, g_2^x, g_1^y, g_2^y, A = (kg_1^y)^{1/(\gamma+x)})$ and so he is able to answer the challenge $Q-\mathsf{HHSDH}$ instance.

$\mathcal{G}_{2a}$ Let us be given a $Q-\mathsf{HSDH}$ challenge (g_1, g_2, g_2^y) and $(g_1^{t_i}, g_2^{t_i}, \mathsf{ID}(y, t_i) = g_1^{1/(y+t_i)})_{i\in[\![1,Q]\!]}$. We build an adversary $\mathcal{B}$ answering this challenge, from an adversary $\mathcal{A}$ breaking the soundness of our scheme by forging a new ID.

$\mathcal{B}$ generates a new γ, skO, he then gives $\mathsf{msk} = \gamma, \mathsf{skO}$ to $\mathcal{A}$, together with the public parameters $(g_1, g_2, k_1, \mathcal{F}, \Omega = g_2^\gamma, \mathcal{C})$. $\mathcal{B}$ can answer any joinP queries as he knows msk, the user on which we expect the attack (the challenge user) will have a certificate corresponding to one with y as a secret key. (Specifically $\mathsf{tk}[i] = g_2^y$). $\mathcal{A}$ can corrupt any user, if he tries to corrupt the challenge user, the simulation fails. As all uncorrupted user looks the same, with non-negligible probably the simulation continues. Thanks to the challenge tuple, $\mathcal{B}$ can answer to at most Q signing queries for challenge user (each time using a new ID).

After at most Q signing queries, $\mathcal{A}$ succeeds in breaking the non-frameability with non-negligible probability by generating a new ID, on an uncorrupted user. As uncorrupted users are indistinguishable, with non negligible probability this user is the challenge one, and so $\mathcal{B}$ is able to produce a new tuple $(g_1^t, g_2^t, g_1^{1/(t+y)})$, which breaks the $Q-\mathsf{HSDH}$ assumption.

$\mathcal{G}_{2b}$ Let us be given an CDH^+ challenge (in other words an asymmetric Waters public key) $(\mathsf{pk} = (g_1^t, g_2^t)$ for the global parameters $(g_1, g_2, k_1, \mathcal{F})$. We build an algorithm $\mathcal{B}$ that breaks the unforgeability of this signature, and thus the CDH^+ problem, from an adversary $\mathcal{A}$ breaking the non-frameability property of our scheme by reusing an existing ID with the corresponding certificate, but on a new message.

$\mathcal{G}_{2b,1}$ In the first game, $\mathcal{B}$ knows the discrete logarithm value t, generates a new γ, skO, he then gives $\mathsf{msk} = \gamma, \mathsf{skO}$ to $\mathcal{A}$, together with the public key $(g_1, g_2, k_1, \Omega = g_2^\gamma, \mathcal{C})$. $\mathcal{B}$ can answer any joinP queries as he knows msk, and extract the secret keys from the extraction keys of the commitment scheme, one of those uncorrupted user is expected to be our challenge user, with secret key y, the one $\mathcal{A}$ has to frame.

$\mathcal{B}$ can answer any signing queries. On one of them for our challenge user, say on m, he will use the above t as ephemeral Waters public key (for the z), and thus computes a $\sigma_0 = \mathsf{ID}(y, t)$ with the corresponding Groth-Sahai proof. This way $\mathcal{A}$ now possesses a valid signature on m, with $\sigma_4 = (g_1^t, g_2^t), \sigma_5 = k_1^t \mathcal{F}(m)^s, \sigma_6 = (g_1^s, g_2^s)$. With non-negligible probably $\mathcal{A}$ breaks the non-frameability of our scheme, by hypothesis $\mathcal{A}$ does it by reusing an existing $\sigma_0, \ldots, \sigma_4$, as uncorrupted users are indistinguishable, $\mathcal{A}$ frames our challenge user with non-negligible probability, and as he makes a finite number of signing queries, he will use with non-negligible probability $\sigma_4 = (g_1^t, g_2^t)$.

Therefore, with non-negligible probability $\mathcal{A}$ outputs a new valid signature on m' with $\sigma_4 = (g_1^t, g_2^t)$, this means we have $(\sigma_4, \sigma_5, \sigma_6)$ such that $e(\sigma_{4,1}, g_2) = e(g_1, \sigma_{4,2}), e(\sigma_{6,1}, g_2) = e(g_1, \sigma_{6,2})$, and $e(\sigma_5, g_2) = e(k_1, \sigma_{4,2}).e(\mathcal{F}(m'), \sigma_{6,2})$, and so $\mathcal{B}$ can outputs a valid forgery on the Waters challenge for the public key (g_1^t, g_2^t). But in this game, we know t.

$\mathcal{G}_{2b,2}$ In a second game, we switch the Groth-Sahai setup into the perfectly hiding one, so that the proofs can be simulated, more precisely without using t when proving the validity of σ_0. This is indistinguishable from the previous game under the SXDH assumption.

$\mathcal{G}_{2b,3}$ In a third game, we replace σ_0 by a random value, still simulating the proofs. As explained in the anonymity proof, a random ID is indistinguishable from the real one under the DDHI problem. Furthermore, here there is only one element, hence the 1-DDHI assumption. In this last game, one does not need to know t anymore, and thus the signature forgery reduces to breaking the CDH^+.

Now let $\mathcal{A}$ be an adversary against the soundness of our scheme with an advantage ϵ. If with probability greater than $\epsilon/3$, $\mathcal{A}$ breaks the misidentification property of the scheme, then we can run the game $\mathcal{G}_1$, else if with probability greater than $\epsilon/3$, $\mathcal{A}$ breaks the non-frameability property with a new ID, then we can run the game $\mathcal{G}_{2a}$, else $\mathcal{A}$ breaks the non-frameability property with a new Waters component and so we run the game $\mathcal{G}_{2b}$. So if there exists an adversary against the soundness of our scheme, we can break with non-negligible probability one of the previous problems. □

3.3 List Signatures

List signatures were introduced by Canard *et al.* in [CSST06]. They let users sign anonymously, in an irrevocable way, but grant linkability in a specific time-frame: no one can trace back the actual signer, but if a user signs two messages within a specific time-frame, the signatures will be linkable. Since then, it has been an open problem to know if there was any way to construct such a list signature scheme in the standard model.

Our previous construction of traceable signature is a first milestone to the realization of such signature. We simply need to adapt the previous *identifier* value, in such a way that on hand it will be the same for two signatures from the same user in a given time period, while on the other hand it is unpredictable between two different time periods.

3.3.1 Security Notions for List Signatures

In this section we briefly present the security notions involved in a list signature, we will see that most of them are directly linked to security notions present in traceable signatures. We will once again use similar notations as [BSZ05]. In a List Signature scheme, there are several users, which are all registered in a PKI. We thus continue to assume that each user $\mathcal{U}_i$ owns a pair $(\mathsf{usk}[i], \mathsf{upk}[i])$ certified by the PKI. In a standard implementation there is only one authority: The Group Manager: it issues certificates for users to grant access to the group. (Technically, we can still add an *Opener*, it will work exactly as before, however for the sake of clarity we will skip this part to lighten our construction.)

A List Signature scheme is thus defined by a sequence of (interactive) publicly verifiable protocols:

List Signature
⌜ LS = (Setup, Join, Sign, Verif, Match):

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter. This algorithm generates the global parameters of the system, the public key pk and the master secret key msk given to the group manager.
- $\mathsf{Join}(\mathcal{U}_i)$: this is an interactive protocol between a user $\mathcal{U}_i$ (using his secret key $\mathsf{usk}[i]$) and the group manager (using his private key msk). At the end of the protocol, the user obtains a signing key $\mathsf{sk}[i]$, and the group manager adds the user to the registration list, storing some information in $Reg[i]$.
- $\mathsf{Sign}(\mathsf{pk}, i, m, t, sk[i])$: this is a (possibly interactive) protocol expected to be made by a registered user i, using his own key $\mathsf{sk}[i]$. It produces a signature σ on the message m at the time-frame t .
- $\mathsf{Verif}(\mathsf{pk}, m, t, \sigma)$: anybody should be able to verify the validity of the signature, with respect to the public key pk. This algorithm thus outputs 1 if the signature is valid, and 0 otherwise.
- $\mathsf{Match}(\mathsf{pk}, m_1, t_1, m_2, t_2, \sigma_1, \sigma_2)$: This outputs 1 iff $t_1 = t_2$ and σ_1 and σ_2 were produced by the same user.

⌟

(a) Experiment $\mathsf{Exp}^{\mathsf{uf}}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(t, (m_i, \sigma_i)_{i \in [\![1,n]\!]}) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{msk} : \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign})$
3. IF $\exists i \mathsf{Verif}(\mathsf{pk}, m_t, t, \sigma_t) = 0$, RETURN 0
4. IF $\exists i \neq j, \mathsf{Match}(\mathsf{pk}, m_i, t, m_j, t, \sigma_i, \sigma_j) = 1$ RETURN 0
5. IF $n > \#\mathsf{CU} + \mathcal{S}(t)$, RETURN 1
6. ELSE RETURN 0

$$\mathsf{Adv}^{\mathsf{uf}}_{\mathsf{LS},\mathcal{A}}(k) = \Pr[\mathsf{Exp}^{\mathsf{uf}}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K}) = 1]$$

(b) Experiment $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(m, t, i_0, i_1) \leftarrow \mathcal{A}(\mathtt{FIND}, \mathsf{pk} : \mathsf{joinA}, \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign})$
4. $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, i_b, m, t, \{sk[i_b]\})$
5. $b' \leftarrow \mathcal{A}(\mathtt{GUESS}, \sigma : \mathsf{joinA}, \mathsf{joinP}, \mathsf{corrupt}, \mathsf{sign})$
6. IF $i_0 \in \mathsf{CU}$ OR $i_1 \in \mathsf{CU}$ RETURN $\perp$
7. IF $(i_0, *) \in \mathcal{S}(t)$ OR $(i_1, *) \in \mathcal{S}(t)$ RETURN $\perp$
8. ELSE RETURN b'

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K}) = 1]$$

Figure 3.6: Security Notions for List Signatures

Security Notions

The *correctness* notion guarantees that honest users should be able to generate valid signatures.

In the following experiments that formalize the security notions, the adversary can run the Join protocol,

- either through the joinP-oracle (passive join), which means that it creates an honest user for whom it does not know the secret keys: the index i is added to the HU (Honest Users) list;
- or through the joinA-oracle (active join), which means that it interacts with the group manager to create a user it will control: the index i is added to the CU (Corrupted Users) list.

After a user is created, the adversary can interact with honest users, granted some oracles, quite similar to the previous ones:

- $\mathsf{corrupt}(i)$, if $i \in \mathsf{HU}$, provides the specific secret key of this user. The adversary can now control it during the whole simulation. Therefore i is added to CU;
- $\mathsf{sign}(\mathsf{pk}, i, m, t)$, if $i \in \mathsf{HU}$, plays as the honest user i would do in the signature process to generate a signature on message m during the time-frame t. Then (i, m, t) is appended to the list $\mathcal{S}$ (generated signatures). It should be noted that the time-frame is included, and $\mathcal{S}(t)$ returns the messages m and user i on which the signing oracle was queried during a specific time-frame..

Soundness: This is the main security notion, see Figure 3.6, page 67 (a): An adversary can produce at most one valid signature per time-frame per corrupted player. LS is *Sound* if for any polynomial adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{uf}}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K})$ is negligible. This combine two properties, first the adversary should not be able to make any user sign twice in the same time-frame without being detected, and then he should not be able to generate a signature on a message m for an uncorrupted player i, except by having queried the signing oracle on the very same user i and message m during the same time-frame.

Anonymity: We now address the privacy concerns, see Figure 3.6, page 67 (b). Given two honest users i_0 and i_1, the adversary should not have any significant advantage in guessing which one of them has issued a valid signature. LS is *anonymous* if, for any polynomial adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathsf{LS},\mathcal{A}}(\mathfrak{K}) = 1]$ is negligible.

3.3.2 List Signatures in the Standard Model

The protocol is quite similar to the previous one except for two things, z is no longer chosen at random, but is simply a scalar corresponding to the time-frame t, for the sake of clarity we will t instead of $z(t)$, and we can no longer use k_1^z as a private Waters key, z being public, so we will use a private h_1^y:

$\mathsf{Setup}(1^{\mathfrak{K}})$: The system generates a pairing-friendly environment $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, generators for a Waters function $\mathcal{F}$ in $\mathbb{G}_1$, a Groth-Sahai CRS denoted by $\mathcal{C}$. One also chooses independent generators (h_1, k_1) of $\mathbb{G}_1^2$. The group manager chooses a scalar $\gamma \xleftarrow{\$} \mathbb{Z}_p$ for the master key $\mathsf{msk} = \gamma$, and computes $\Omega = g_2^\gamma$. The group public key is then $\mathsf{pk} = (g_1, g_2, h_1, k_1, \mathcal{F}, \Omega, \mathcal{C})$.

$\mathsf{Join}(\mathcal{U}_i)$**:** In order to join the system, a user $\mathcal{U}_i$, with a pair of keys $(\mathsf{usk}[i], \mathsf{upk}[i])$ in the PKI, interacts with the group manager, similarly to the previous scheme, so that at the end, the user owns a certificate $\{A_i, \mathbf{X}_i = (g_1^{x_i}, g_2^{x_i}), g_1^{y_i}\}$, where x_i is chosen by the group manager but y_i is chosen in common, but private to the user, while still extractable for our simulator in the proof. Then, $Reg[i] = \{i, \mathsf{upk}[i], A_i, \mathbf{X}_i, g_1^{y_i}, \mathbf{e}_i, \mathbf{s}_i\}$, whereas $\mathsf{sk}[i] = y_i$.

$\mathsf{Sign}(\mathsf{pk}, m, t, \mathsf{sk}[i])$**:** When a user i wants to sign a message m during the time-frame t, he computes the signature of m under his private key $\mathsf{sk}[i]$: First, he will create his ephemeral $\mathsf{ID}(i,t) = g_1^{1/(t+y_i)}$, and computes

$$\sigma = (\sigma_0 = \mathsf{ID}(i,t), \sigma_1 = \underline{\mathbf{X}_i}, \sigma_2 = \underline{y_i}, \sigma_3 = \underline{A_i}, \sigma_4 = g_1^s, g_2^s, \sigma_5 = \underline{h_1^{y_i}\mathcal{F}(m)^s}).$$

The relations could be verified by:

$$\begin{aligned}
e(\sigma_0, g_2^t g_2^{\sigma_{2,2}}) &= e(g_1, g_2) & e(\underline{\sigma_{1,1}}, g_2) &= e(g_1, \underline{\sigma_{1,2}}) \\
e(\underline{\sigma_3}, \Omega\underline{\sigma_{1,2}}) &= e(k_1, g_2) \cdot e(g_1, g_2^{\sigma_{2,2}}) & e(g_1^{\sigma_{2,1}}, g_2) &= e(g_1, g_2^{\sigma_{2,2}}) \\
(\underline{\sigma_5}, g_2) &= e(h_1, g_2^{\sigma_{2,2}}) \cdot e(\mathcal{F}(m), \sigma_{4,2}) & e(g_1^{\sigma_{4,1}}, g_2) &= e(g_1, g_2^{\sigma_{4,2}})
\end{aligned}$$

In order to get anonymity, before publication, some of these tuples are thereafter committed, together with the corresponding Groth-Sahai proofs, to prove the validity of the previous pairing equations

$$\sigma_i = (\sigma_0, \mathcal{C}(\sigma_1), \mathcal{C}(\sigma_2), \mathcal{C}(\sigma_3), \sigma_4, \mathcal{C}(\sigma_5))$$

y_i has to be committed both in $\mathbb{G}_1$ and $\mathbb{G}_2$, hence the existence of both $\sigma_{2,1}$ and $\sigma_{2,2}$ in the previous relations.

$\mathsf{Match}(\mathsf{pk}, m, t, m', t', \sigma, \sigma')$**:** This algorithm return 1 iff $t = t'$ and $\sigma_0 = \sigma'_0$.

It should be noted, that such signatures also allow an *Opener* (if he knows the commitments extraction key, and possibly tracing authorities, if the opener gives them the tracing key $\mathsf{tk}[i]$: the plaintext associated with $\mathbf{e}_i$)

Security

The security of this scheme can be proven in a similar way to the previous one. The main difference between the two schemes comes from σ_5 where we cannot use t but a the private value y_i that appears in some other equations, and so we need to commit it, together with a proof of validity under the committed verification key $\sigma_{2,2}$. This alters a little the security proof. However in both cases it is easier. In the anonymity, σ_5 becomes a random when we switch in the perfectly hiding setting, so we only need to worry about it in G_1, however we have g_1^y, so we only need to define $h_1 = g_1^\mu$ for a random μ and we can compute $h_1^{y-t^*} = (g_1^y / g_1^{t*})^\mu$. Afterwards it is exactly the same as before. And since the adversary can only make a limited number of signature queries, he will only be able to work on a polynomial number of time-frames t and so we can still use the Dodis-Yampolskiy VRF.

Theorem 3.3.1 *If there exists an adversary $\mathcal{A}$ that can break the anonymity property of the scheme, then there exists an adversary $\mathcal{B}$ that can break the $\ell -$ DDHI problem in $\mathbb{G}_1$ or the* SXDH *assumption, where ℓ is the maximal number of signing queries for a user.*

Intuitively, we will show first that a perfectly binding instantiation can be supplanted by a perfectly hiding one without consequence under SXDH, in this case only the ID can possibly link information, but under DDHI it does not.

Proof: Let us assume that an adversary is able to break the anonymity property of our scheme. It means that in the anonymity game, he has a non-negligible advantage $\epsilon > 0$ to distinguish $G^{(0)}$ where $b = 0$ from $G^{(1)}$ where $b = 1$. We start our sequence of games from $G^{(0)}$, denoted $\mathcal{G}_0$.

$\mathcal{G}_1$ The simulator $\mathcal{B}$ is given a challenge $A = (g, g^y, \ldots, g^{y^\ell}) \in \mathbb{G}_1^{\ell+1}$ and $D = g^{1/y}$, for an unknown y. $\mathcal{B}$ first chooses different random values $t^*, t_1, \ldots, t_{\ell-1} \in \mathbb{Z}_p^\ell$, which completely define the polynomial $P = \prod_{i=1}^{\ell-1}(X + t_i)$, of degree $\ell - 1$. Granted the above challenge, $\mathcal{B}$ can compute $g_1 = g^{P(y)}$, and the rest of the public key is computed like in the regular scheme. In particular, one knows the master secret key γ.

The future challenge user i_0 will virtually have $y_{i_0} = \mathsf{sk}[i_0] = y - t^*$. $\mathcal{B}$ can compute $g^{y_{i_0}} = g_1^y / g_1^{t^*}$ (from the input A without using y, even if we know it here). The public certificate (in the *Reg* list) for the challenge user is $(g_1^{x_{i_0}}, g_2^{x_{i_0}}, g_1^y / g_1^{z^*}, (k_1 g_1^y / g_1^{z^*})^{1/(\gamma+x)})$, plus some proofs and signatures. $\mathcal{B}$ is also given Groth-Sahai commitment keys (extraction key is unknown, hence the classical notion of anonymity and not full-anonymity). This Setup is indistinguishable from the real one since all the keys are generated as in the real game. He also defines $h_1 = g_1^\mu$, so the private signing key for the challenge user is $\mathsf{sk} = h_1^{y_{i_0}} = (g_1^y / g_1^{t^*})^\mu$. Because of the knowledge of the master secret key, $\mathcal{B}$ easily answers any join queries, both active and passive. However $\mathcal{B}$ still has to guess i_0, which is correct with probability $1/n$, where n is the total number of passive join queries. It can also answer any corruption, that should not happen for the challenge user, even if we know y in this game.

The simulator $\mathcal{B}$ is able to know all x_i and all y_i for registered users (except the challenge one), he will be able to answer signing queries as users would do. For the challenge user, on the j-th signing query, he computes $\sigma_0 = g_1^{1/(\mathsf{sk}[i]+z_j)} = g^{\prod_{i \neq j}(y+t_i)}$, which can be done from the challenge input A, the rest is done as in the real game using y and a s_j as ephemeral random and the signing key sk. For the challenge signing query, he does the same has above with the ephemeral value t^*, and the expected ID is $g_1^{1/(\mathsf{sk}[i]+t^*)} = g^{P(y)/y} = g^{Q(y)} g^{\prod(t_i)/y}$, where $Q = (\prod_{i=1}^{\ell}(X + t_i) - \prod(t_i))/X$ is a polynomial of degree $\ell - 1$ and thus $g^{Q(y)}$ can be computed from the instance. He thus outputs $\sigma_0 = g^{Q(y)} \cdot D^{\prod(t_i)}$. Since $D = g^{1/y}$, the signature is similar to the above one, and so is indistinguishable from a real signature.

$\mathcal{G}_2$ We then modify the game, and we initialize Groth-Sahai commitment keys which leads to perfectly WI commitments and proofs. This game is indistinguishable from the previous one under the SXDH.

$\mathcal{G}_3$ For the challenge user signing queries, we use random commitments for $\sigma_1, \sigma_2, \sigma_3, \sigma_5$. As we are in the perfectly hiding setting, this is indistinguishable anyway.

$\mathcal{G}_4$ We do not know anymore y, that we did not use anyway, and thus this game is perfectly indistinguishable from the previous one.

$\mathcal{G}_5$ D is a random value, which is indistinguishable from the real one under the $\ell-\mathsf{DDHI}$ assumption as we only have a polynomial number of t_i in input like in the Dodis-Yampolskiy PRF: the challenge signature does not depend anymore on the challenge user.

To complete the proof, we should make the same sequence again, starting from $\mathcal{G}_0$' that is $G^{(1)}$, up to $\mathcal{G}_5$', that is perfectly indistinguishable from $\mathcal{G}_5$, hence the computational indistinguishability between $\mathcal{G}_0' = G^{(1)}$ and $\mathcal{G}_0 = G^{(0)}$. □

Soundness: Within the soundness analysis, we prove traceability and non-frameability.

Theorem 3.3.2 *If there exists an adversary $\mathcal{A}$ against the soundness of the scheme, then we can build an adversary $\mathcal{B}$ that can either break a computational problem ($Q-\mathsf{HHSDH}$, $Q'-\mathsf{HSDH}$, or CDH^+), or a decisional one (the 1-DDHI, or the SXDH), where Q is maximal number of users, and Q' is the maximal number of signing queries for a user.*

The soundness property is easier to achieve than in the previous protocol, being able to sign twice in the same time-frame implies to be able either to generate two different ID for the same t, and so to work with two different y_i, and so to have two certificates, either to sign another message with the same user and to break the Waters unforgeability. And so the simulator will use the extraction key of Groth and Sahai commitments in the perfectly binding setting to work to obtain the answer to the associated challenges.

We just presented the first construction of List Signature in the standard model. However starting from the same idea than our Traceable Signature, one is also able to create delegatable signatures. One can see that our previous $(\mathsf{vk} = (g_1^z, g_2^z), \mathsf{sk} = h_1^z)$ can be seen as a Waters signature key, certified by the user $\mathcal{U}_i$ who generated $\sigma_0, \sigma_1, \sigma_2, \sigma_3$. Anyone who possesses such pair of keys can sign in the name of $\mathcal{U}_i$. In fact they sign in the same of someone in the group who remain *weakly*-anonymous as long as any opener / tracer is involved. The *weakly* is there, because two signatures generated by the same delegated-signers are linkable. However while the adversary can link signatures generated by the same delegated-signers this does not leak extra information about $\mathcal{U}_i$.

Chapter 4

Signatures on Randomizable Ciphertexts

Contents

In this chapter we propose a new primitive we call *Signature on Randomizable Ciphertexts*: given a signature on a ciphertext, anyone, knowing neither the signing key nor the encrypted message, can randomize the ciphertext and *adapt* the signature to the fresh encryption. An *Extractable Signature on Randomizable Ciphertexts* is the same, where knowing some decryption key dk, one can recover the plain signature on the original plaintext, such that $\mathsf{SigExt}_{\mathcal{SC}}(\mathsf{dk}, \mathsf{vk}, \mathsf{Sign}_{\mathcal{SC}}(\mathsf{sk}, \mathsf{pk}, (\mathsf{Encrypt}_{\mathcal{SC}}(\mathsf{pk}, \mathsf{vk}, m; r)); s)) = \mathsf{Sign}_{\mathcal{S}}(\mathsf{sk}, \mathsf{pk}, m; s)$. Leading us to a somewhat commutative property between the signature and the encryption.

We originally introduced this notion in [BFPV11], and here we propose an extended version where we are able to handle more sophisticated ciphertexts (basically ciphertexts with two public parts, each chosen by a participant). Those results let us achieve a round optimal blind signature under classical assumptions in the standard model, and even a perfectly blind round optimal signature with partial blindness under the same assumptions. We further define the notion of Multi-Blind Signatures, where under our new result on Waters function programmability (cf 2.6.4, page 42) we are able to combine several blind signatures into one. From now on, we drop any non standard hypotheses and solely prove our security under DLin and CDH.

4.1 Definition, and Security Notions

This section presents the global framework and the security model for our new concept of signatures on ciphertexts (or commitments).

We first define a scheme of signatures on ciphertexts. Note that, like stated earlier, this definition can be adapted for commitments, when one uses a perfectly binding commitment scheme, which uniquely defines the committed input.

Signatures on Ciphertexts
$\ulcorner \mathcal{SC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathcal{S}}, \mathsf{KeyGen}_{\mathcal{E}}, \mathsf{Encrypt}, \mathsf{Sign}, \mathsf{Decrypt}, \mathsf{Verif})$ is defined as follows:

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param_e and param_s for the associated encryption and signature schemes;
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param}_e)$ generates a pair of keys, the encryption key pk and the associated decryption key dk;
- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param}_s)$ generates a pair of keys, the verification key vk and the signing key sk;
- $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{vk}, m; r)$ produces a ciphertext c on input the message $m \in \mathcal{M}$ and the encryption key pk, using the random coins $r \in \mathcal{R}_e$. This ciphertext is intended to be later signed under the signing key associated to the verification key vk (the field for vk can be empty if the signing algorithm is universal and does not require a ciphertext specific to the signer);
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{pk}, c; s)$, on input a ciphertext c and a signing key sk, using the random coins $s \in \mathcal{R}_s$, produces a signature σ, or $\perp$ if the ciphertext c is not valid (w.r.t. pk, and possibly vk associated to sk);
- $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{vk}, c)$ decrypts the ciphertext c under the private key dk. It outputs the plaintext, or $\perp$ if c is invalid (w.r.t. pk, and possibly vk);
- $\mathsf{Verif}(\mathsf{vk}, \mathsf{pk}, c, \sigma)$ checks whether σ is a valid signature on c, w.r.t. the public key vk. It outputs 1 if the signature is valid, and 0 otherwise (possibly because of an invalid ciphertext c, with respect to pk, and possibly vk).

⌟

Classical security notions could still be applied to this signature scheme, but we want ciphertexts and signatures to be efficiently malleable, as long as the plaintext is not affected. This will be useful for probabilistic schemes, and even more for the randomizable scheme we will present below. In the classical definition of *existential unforgeability* (EUF) [GMR88], a new signature on an already signed message is not considered a valid forgery—as opposed to strong unforgeability (SUF). When signing ciphertexts, EUF would consider a signature on a randomized ciphertext as a valid forgery. But if the ciphertext is equivalent to an already signed ciphertext (i.e. it encrypts the same plaintext), this may not be critical in some applications; in particular if we decrypt later anyway and a decrypted message-signature pair is unforgeable. We thus define the most appropriate unforgeability (UF) notion for signatures on ciphertexts:
$\mathcal{SC}$ is *unforgeable* if, for any polynomial-time adversary $\mathcal{A}$, the advantage $\mathsf{Succ}^{\mathsf{uf}}_{\mathcal{SC},\mathcal{A}}(\mathfrak{K}) := \Pr[\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{SC},\mathcal{A}}(\mathfrak{K}) = 1]$ is negligible, with $\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{SC},\mathcal{A}}$ defined in Figure 4.1, page 71.

$\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{SC},\mathcal{A}}(\mathfrak{K})$
1. $(\mathsf{param}_e, \mathsf{param}_s) \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$; $\mathsf{SM} := \emptyset$
2. $\{(\mathsf{pk}_i, \mathsf{dk}_i)\} \leftarrow \mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param}_e)$; $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param}_s)$
3. $(\mathsf{pk}_j, c, \sigma) \leftarrow \mathcal{A}^{\mathsf{sign}(\mathsf{sk},\cdot,\cdot)}(\mathsf{param}_s, \mathsf{param}_e, \mathsf{vk}, \{(\mathsf{pk}_i, \mathsf{dk}_i)\})$;
4. $m \leftarrow \mathsf{Decrypt}(\mathsf{dk}_j, \mathsf{vk}, c)$
5. `IF` $m = \perp$ `OR` $m \in \mathsf{SM}$ `OR` $\mathsf{Verif}(\mathsf{vk}, \mathsf{pk}_j, c, \sigma) = 0$ `RETURN` 0
6. `RETURN` 1

Figure 4.1: Unforgeability of signatures on ciphertexts

Where $\mathsf{sign}(\mathsf{sk}, \cdot, \cdot)$ is an oracle that takes as input a previously generated encryption key pk_i and a ciphertext c, and generates a signature σ on it (if the ciphertext is valid). It also updates the set SM of signed plaintexts with $m = \mathsf{Decrypt}(\mathsf{dk}_i, \mathsf{vk}, c)$, if the latter exists.

Unforgeability in the above sense thus states that no adversary is able to generate a new valid ciphertext-signature pair for a ciphertext that encrypts a new message, i.e. different to those encrypted in ciphertexts that were queried to the signing oracle.

Signatures on Randomizable Ciphertexts

Our primitive is based on an encryption scheme and a signature scheme. Since we want randomizability, we use the previous definitions where both are enhanced by a randomization algorithm.

Randomizable Encryption Scheme

⌜ Let $(\mathsf{Setup}, \mathsf{KeyGen}_{\mathcal{E}}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an encryption scheme with the following additional algorithm:

- $\mathsf{Random}(\mathsf{pk}, c; r')$ produces a new ciphertext c', equivalent to the input ciphertext c, under the public key pk, using the additional random coins $r' \in \mathcal{R}_e$.

An encryption scheme is called *randomizable* if for any $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}_\mathcal{E}(\mathsf{param})$, message $m \in \mathcal{M}$, coins $r \in \mathcal{R}_e$, and ciphertext $c = \mathsf{Encrypt}(\mathsf{pk}, m; r)$, the following distributions are statistically indistinguishable: $\mathcal{D}_0 = \{r' \xleftarrow{\$} \mathcal{R}_e : \mathsf{Encrypt}(\mathsf{pk}, m; r')\}$ and $\mathcal{D}_1 = \{r' \xleftarrow{\$} \mathcal{R}_e : \mathsf{Random}(\mathsf{pk}, c; r')\}$.

⌟

Randomizable Signature Scheme

⌜ Let $(\mathsf{Setup}, \mathsf{KeyGen}_\mathcal{S}, \mathsf{Sign}, \mathsf{Verif})$ be a signature scheme, with the following additional algorithm:

- $\mathsf{Random}(\mathsf{vk}, M, \sigma; s')$ produces a new signature σ' valid under vk from σ on a message M, using the additional random coins $s' \in \mathcal{R}_s$.

A signature scheme is called *randomizable* if for any $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_\mathcal{S}(\mathsf{param})$, message $M \in \mathcal{M}$, random $s \in \mathcal{R}_s$, signature $\sigma = \mathsf{Sign}(\mathsf{sk}, M; s)$, the following distributions are statistically indistinguishable: $\mathcal{D}_0 = \{s' \xleftarrow{\$} \mathcal{R}_s : \mathsf{Sign}(\mathsf{sk}, M; s')\}$ and $\mathcal{D}_1 = \{s' \xleftarrow{\$} \mathcal{R}_s : \mathsf{Random}(\mathsf{vk}, M, \sigma; s')\}$. ⌟

The usual unforgeability notions apply (except strong unforgeability, since the signature is malleable, by definition). We now extend the randomization to signatures on randomizable ciphertexts:

Randomizable Signature on Randomizable Ciphertexts

⌜ Let $(\mathsf{Setup}, \mathsf{KeyGen}_\mathcal{S}, \mathsf{KeyGen}_\mathcal{E}, \mathsf{Encrypt}, \mathsf{Sign}, \mathsf{Decrypt}, \mathsf{Verif})$ be a scheme of signatures on ciphertexts, with the following additional algorithm:

- $\mathsf{Random}(\mathsf{vk}, \mathsf{pk}, c, \sigma; r', s')$ outputs a ciphertext c' that encrypts the same message as c under the public key pk, and a signature σ' on c'. Further inputs are a signature σ on c under vk, and random coins $r' \in \mathcal{R}_e$ and $s' \in \mathcal{R}_s$.

A signature on ciphertexts is called *randomizable* if for any global parameters $(\mathsf{param}_e, \mathsf{param}_s) \leftarrow \mathsf{Setup}(1^\kappa)$, keys $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}_\mathcal{E}(\mathsf{param}_e)$ and $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_\mathcal{S}(\mathsf{param}_s)$, $m \in \mathcal{M}$, and random coins $r \in \mathcal{R}_e$ and $s \in \mathcal{R}_s$, for $c = \mathsf{Encrypt}(\mathsf{pk}, \mathsf{vk}, m; r)$ and $\sigma = \mathsf{Sign}(\mathsf{sk}, \mathsf{pk}, c; s)$ the following distributions $\mathcal{D}_0$ are statistically indistinguishable:

$$\mathcal{D}_0 = \{r' \xleftarrow{\$} \mathcal{R}_e; s' \xleftarrow{\$} \mathcal{R}_s : (c' = \mathsf{Encrypt}(\mathsf{pk}, \mathsf{vk}, m; r'), \sigma' = \mathsf{Sign}(\mathsf{sk}, \mathsf{pk}, c'; s'))\}$$
$$\mathcal{D}_1 = \{r' \xleftarrow{\$} \mathcal{R}_e; s' \xleftarrow{\$} \mathcal{R}_s : (c', \sigma') = \mathsf{Random}(\mathsf{vk}, \mathsf{pk}, c, \sigma; r', s')\}$$

⌟

We will denote by 1_e and 1_s the neutral elements in $\mathcal{R}_e$ and $\mathcal{R}_s$ that keep the ciphertexts and/or signatures unchanged after randomization. If $\mathcal{R}_e$ and $\mathcal{R}_s$ are groups (which will be the case for all our schemes, with addition being the group operation) and if we show that it is possible to additively update the randomness then this proves that the schemes are randomizable. The same unforgeability notion as above applies. If an additional extraction algorithm exists for the signature, we get extractable signatures on ciphertexts (defined below). Then, our above unforgeability notion for signatures on ciphertexts follows from the standard unforgeability notion on signatures.

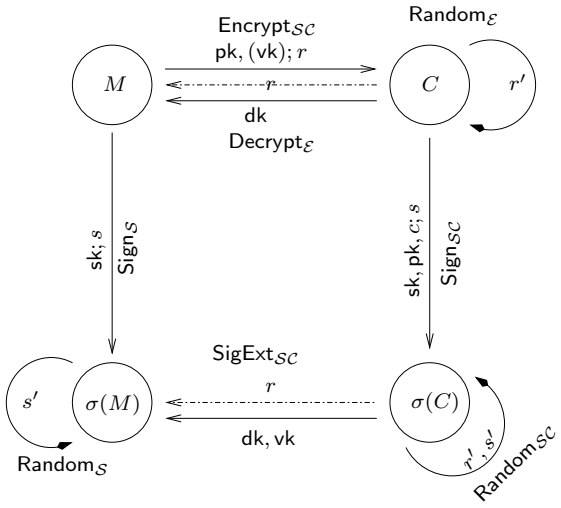
Extractable Signatures on Randomizable Ciphertexts

For a scheme of signatures on randomizable ciphertexts (SRC) $\mathcal{SC}$, we define the following additional algorithm:

- $\mathsf{SigExt}_\mathcal{SC}(\mathsf{dk}, \mathsf{vk}, \sigma)$, which is given a decryption key, a verification key and a signature, outputs a signature σ'.

Let us assume that there is a signature scheme $\mathcal{S}$ where $\mathsf{Setup}_\mathcal{S}$ is the projection of $\mathsf{Setup}_\mathcal{SC}$ on the signature component and $\mathsf{KeyGen}_\mathcal{S} = \mathsf{SKeyGen}$. The scheme $\mathcal{SC}$ is *extractable* if the following holds: for any $(\mathsf{param}_e, \mathsf{param}_s) \leftarrow \mathsf{Setup}_\mathcal{SC}(1^\kappa)$, $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{EKeyGen}(\mathsf{param}_e)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SKeyGen}(\mathsf{param}_s) = \mathsf{KeyGen}_\mathcal{S}(\mathsf{param}_s), m \in \mathcal{M}$, random coins $r \in \mathcal{R}_e, s \in \mathcal{R}_s$, for $c = \mathsf{Encrypt}_\mathcal{SC}(\mathsf{pk}, \mathsf{vk}, m; r)$ and $\sigma = \mathsf{Sign}_\mathcal{SC}(\mathsf{sk}, \mathsf{pk}, c; s)$, the output $\sigma' = \mathsf{SigExt}_\mathcal{SC}(\mathsf{dk}, \mathsf{vk}, \sigma)$ is a valid signature on m under vk, that is, $\mathsf{Verif}_\mathcal{S}(\mathsf{vk}, m, \sigma')$ is true.

An extractable SRC scheme $\mathcal{SC}$ allows the following: a user can encrypt a message m and obtain a signature σ on the ciphertext c. From (c, σ) the owner of the decryption key can now not only recover the encrypted message m, but also a signature σ' on the *message* m, using the functionality $\mathsf{SigExt}_{\mathcal{SC}}$. The signature σ on the ciphertext c could thus be seen as an *encryption of a signature* on the message m: for extractable signatures on ciphertexts, encryption and signing can thus be seen as commutative (see Figure 4.2, page 73).



A message M can be encrypted using random coins r ($\mathsf{Encrypt}_{\mathcal{SC}}$).
The signer can sign this ciphertext ($\mathsf{Sign}_{\mathcal{SC}}$) and anyone can randomize the pair ($\mathsf{Random}_{\mathcal{SC}}$).
A signature on the plaintext can be obtained using either dk (for $\mathsf{SigExt}_{\mathcal{SC}}$) or the coins r (if $\sigma(C)$ has not been randomized); the result is the same as a signature of M by the signer ($\mathsf{Sign}_{\mathcal{S}}$).

Figure 4.2: (Strong) extractable signatures on randomizable ciphertexts

On this figure, one can easily see that $\mathsf{SigExt}_{\mathcal{SC}} \circ \mathsf{Sign}_{\mathcal{SC}} \circ \mathsf{Encrypt} = \mathsf{Sign}_{\mathcal{S}}$, granting therefore some kind of commutativity between the signature and the encryption.

Strong Extractability

We can immediately apply the notion of extractable signatures on randomizable ciphertexts to build a one-round classical blind signature scheme, but we can even consider more complex scenarios, such as *three-player blind signature schemes* (see Section 4.3, page 79) with applications to e-cash systems.

As already sketched above, we may have an additional property: as for encryption, knowing the random coins used for encryption may suffice to decrypt. After encrypting a message m as c, one knows the random coins r used for the encryption. In all our instantiations we have that σ is the encryption of σ' with the same coins r used to encrypt the message. The user who encrypted m is thus able to extract σ', and not only the owner of the decryption key. A system $(\mathcal{SC}, \mathcal{S})$ with such a property will be called a *Strong Extractable (Randomizable) Signature on Ciphertexts* (augmented by the dotted lines in Figure 4.2, page 73). This property will be crucial in our scheme which achieves perfect blindness in Section 4.4.1, page 82

4.2 Instantiations

4.2.1 A First Instantiation

Our first construction combines linear encryption [BBS04] and Waters signatures [Wat05] as follows: given an encryption of the "Waters hash" $\mathcal{F}(M)$ of a message M (and some additional values), the signer can make an encryption of a signature on M. Decrypting the latter leads thus to a classical Waters signature on M, which will provide extractability.

Before presenting the final scheme in Section 4.2.2, page 76, we first combine naively linear encryption and standard Waters signatures. We then modify the Waters signature to significantly improve efficiency of the scheme. The constructions we give here all make use of a symmetric pairing and so result in a prettier protocol ; as for our other schemes we also did an asymmetric construction more efficient which is just a straightforward transposition.

Waters Signature on Linear Ciphertexts

Using Waters signatures, we will sign a linear encryption of $F = \mathcal{F}(M)$. We note that from a "ciphertext" using the decryption key, one can only extract $\mathcal{F}(M)$ (from which M can be obtained for small message spaces). Signatures only remain unforgeable on F if in addition a proof Π_M of knowledge of M such that $F = \mathcal{F}(M)$ is given. The keys are independent Waters signature keys ($\mathsf{vk} = Y = g^y$ and $\mathsf{sk} = Z = h^y$), and linear encryption keys ($\mathsf{dk} = (x_1, x_2)$, $\mathsf{pk} = (X_1 = g^{x_1}, X_2 = g^{x_2})$). A first idea would be to define a signature on an encrypted message $c = \big(c_1 = X_1^{r_1}, c_2 = X_2^{r_2}, c_3 = g^{r_1+r_2} \cdot \mathcal{F}(M)\big)$ as $\sigma = (c_1^s, c_2^s, Z \cdot c_3^s)$. However, there are two problems:

- While the randomization of the signing coins s into $s + s'$ is easy from c, the randomization of the encryption coins r into $r + r'$ requires the knowledge of the values X_1^s, X_2^s and g^s (see Section 4.2.2, page 77 for how to randomize). We therefore include them in the signature.

- For the reduction of our notion of unforgeability to the security of Waters' scheme, we need to simulate the oracle returning signatures on ciphertexts having a Waters signature oracle. We can first extract M from the proof of knowledge Π_M and submit M to our oracle. From a reply $(Z \cdot \mathcal{F}(M)^s, g^{-s})$, we then have to generate $\sigma = (c_1^s, c_2^s, Z \cdot c_3^s; X_1^s, X_2^s, g^s)$ for an unknown s. We could do so if we knew the randomness (r_1, r_2) for c_1, c_2 and c_3; hence we add another proof to the extended ciphertext: Π_r proves knowledge of r_1 and r_2, used to encrypt $\mathcal{F}(M)$, which consists of bit-by-bit commitments $C_1 = (\mathcal{C}'(r_{1,1}), \ldots, \mathcal{C}'(r_{1,\ell}))$ and $C_2 = (\mathcal{C}'(r_{2,1}), \ldots, \mathcal{C}'(r_{2,\ell}))$, where ℓ is the bit-length of the order p, and proofs that each sub-commitment is indeed a bit commitment.

The global proof on the message and the randomness, which we denote by $\Pi = (\Pi_M, \Pi_r)$, can be done with randomizable commitments and proofs, using the Groth-Sahai methodology [GS08, FP09], and consists of $9\,k + 18\,\ell + 6$ group elements (where k and ℓ are the respective bit lengths of messages and of the order of $\mathbb{G}$). Such an extended ciphertext (c, Π) can then be signed, after a test of validity of the proof Π. Decryption and verification follow straight from the corresponding algorithms for Waters signatures and linear encryption. More interestingly, the above signature on randomizable ciphertexts is *extractable*: on a valid signature, if one knows the decryption key $\mathsf{dk} = (x_1, x_2)$, one can compute $\Sigma = (\Sigma_1 = \sigma_3/(\sigma_1^{1/x_1}\sigma_2^{1/x_2}), \Sigma_2 = \sigma_6^{-1})$, which is a valid signature on M:

$$\Sigma_1 = \sigma_3/(\sigma_1^{1/x_1}\sigma_2^{1/x_2}) = Z \cdot g^{s(r_1+r_2)} \cdot \mathcal{F}(M)^s/(g^{sr_1}g^{sr_2}) = Z \cdot \mathcal{F}(M)^s$$
$$\Sigma_2 = \sigma_6^{-1} = g^{-s}$$

Note that without knowing the decryption key, the same can be obtained from the coins (r_1, r_2) used for encryption: $\Sigma = (\Sigma_1 = \sigma_3/\sigma_6^{r_1+r_2}, \Sigma_2 = \sigma_6^{-1})$.

From the randomization formula of the basic schemes, we easily get the randomization property of the above Waters signature on linear ciphertexts. One shows unforgeability in the UF sense under the CDH assumption in $\mathbb{G}$: extractability provides a forgery on a new message, but only known as $F = \mathcal{F}(M)$. Since one also has to provide a valid proof Π_M that contains commitments to the bits of message M, the knowledge of the trapdoor (λ, μ) for the commitments allows to recover M too, which leads to an existential attack of the basic Waters signature scheme.

Theorem 4.2.1 *The Waters signature on linear ciphertexts is extractable when one defines*

- $\mathsf{SigExt}(\mathsf{dk} = (x_1, x_2), \mathsf{vk} = X, \sigma)$*: On a valid signature, if one knows the decryption key* $\mathsf{dk} = (x_1, x_2)$*, one can get back a signature on* M *(or* $F = \mathcal{F}(M)$*):* $\Sigma = (\Sigma_1 = \sigma_3/(\sigma_1^{1/x_1}\sigma_2^{1/x_2}), \Sigma_2 = \sigma_6^{-1})$*. Note that one can also get the same value from the coins for encryption* r_1, r_2*:* $\Sigma = (\Sigma_1 = \sigma_3/\sigma_6^{r_1+r_2}, \Sigma_2 = \sigma_6^{-1})$*.*

Proof: The proof follows from the fact that

$$\Sigma_1 = \sigma_3/(\sigma_1^{1/x_1}\sigma_2^{1/x_2}) = Y \cdot c_3^s/(c_1^{s/x_1}c_2^{s/x_2}) = Y \cdot g^{s(r_1+r_2)} \cdot \mathcal{F}(M)^s/g^{sr_1}g^{sr_2} = Y \cdot \mathcal{F}(M)^s,$$
$$\Sigma_2 = \sigma_6^{-1} = g^{-s}.$$

□

Theorem 4.2.2 *The Waters signature on linear ciphertexts is randomizable when one defines*

- $\mathsf{Random}(\mathsf{vk}=X, \mathsf{pk}=(X_1,X_2), (c=(c_1,c_2,c_3),\Pi), \sigma=(\sigma_1,\sigma_2,\sigma_3;\sigma_4,\sigma_5,\sigma_6); r'_1, r'_2, s')$*: In order to randomize the signature and the ciphertext, the algorithm outputs:*

$$c' = (c_1 \cdot X_1^{r'_1}, c_2 \cdot X_2^{r'_2}, c_3 \cdot g^{r'_1+r'_2})$$
$$\sigma' = (\sigma_1 \cdot c_1^{s'} \times \sigma_4^{r'_1} \cdot X_1^{r'_1 s'}, \sigma_2 \cdot c_2^{s'} \times \sigma_5^{r'_2} \cdot X_2^{r'_2 s'}, \sigma_3 \cdot c_3^{s'} \times \sigma_6^{r'_1+r'_2} \cdot g^{(r'_1+r'_2)s'}; \sigma_4 \cdot X_1^{s'}, \sigma_5 \cdot X_2^{s'}, \sigma_6 \cdot g^{s'})$$

together with a randomization Π' *of* Π.

Proof: On the input $(\mathsf{vk}=X, \mathsf{pk}=(X_1,X_2), (c=(c_1,c_2,c_3),\Pi), \sigma=(\sigma_1,\sigma_2,\sigma_3;\sigma_4,\sigma_5,\sigma_6); r'_1, r'_2, s')$, where for some random scalars $s, r_1, r_2 \in \mathbb{Z}_p$,

$$c = (c_1 = X_1^{r_1}, c_2 = X_2^{r_2}, c_3 = g^{r_1+r_2} \cdot \mathcal{F}(M))$$
$$\sigma = (c_1^s = X_1^{r_1 s}, c_2^s = X_2^{r_2 s}, Y \cdot c_3^s = Y \cdot g^{(r_1+r_2)s} \cdot \mathcal{F}(M)^s; X_1^s, X_2^s, g^s)$$

the algorithm outputs (where $R_1 = r_1 + r'_1$, $R_2 = r_2 + r'_2$ and $S = s + s'$):

$$\begin{aligned}
c' &= (c_1 \cdot X_1^{r'_1} = X_1^{r_1+r'_1}, c_2 \cdot X_2^{r'_2} = X_2^{r_2+r'_2}, c_3 \cdot g^{r'_1+r'_2} = g^{r_1+r'_1+r_2+r'_2} \cdot \mathcal{F}(M)) \\
&= (X_1^{R_1}, X_2^{R_2}, g^{R_1+R_2} \cdot \mathcal{F}(M)) \\
\sigma' &= (\sigma_1 \cdot c_1^{s'} \times \sigma_4^{r'_1} \cdot X_1^{r'_1 s'}, \sigma_2 \cdot c_2^{s'} \times \sigma_5^{r'_2} \cdot X_2^{r'_2 s'}, \sigma_3 \cdot c_3^{s'} \times \sigma_6^{r'_1+r'_2} \cdot g^{(r'_1+r'_2)s'}; \sigma_4 \cdot X_1^{s'}, \sigma_5 \cdot X_2^{s'}, \sigma_6 \cdot g^{s'}) \\
&= (X_1^{r_1 s} \cdot X_1^{r_1 s'} \times X_1^{r'_1 s} \cdot X_1^{r'_1 s'}, X_2^{r_2 s} \cdot X_2^{r_2 s'} \times X_2^{r'_2 s} \cdot X_2^{r'_2 s'}, \\
&\qquad Y \cdot \mathcal{F}(M)^s \cdot g^{(r_1+r_2)s} \cdot \mathcal{F}(M)^{s'} \cdot g^{(r_1+r_2)s'} \times g^{(r'_1+r'_2)s} \cdot g^{(r'_1+r'_2)s'}; X_1^{s+s'}, X_2^{s+s'}, g^{s+s'}) \\
&= (X_1^{(r_1+r'_1)(s+s')}, X_2^{(r_2+r'_2)(s+s')}, Y \cdot \mathcal{F}(M)^{s+s'} \cdot g^{(r_1+r_2+r'_1+r'_2)(s+s')}; X_1^{s+s'}, X_2^{s+s'}, g^{s+s'}) \\
&= (X_1^{R_1 S}, X_2^{R_2 S}, Y \cdot \mathcal{F}(M)^S \cdot g^{(R_1+R_2)S}; X_1^S, X_2^S, g^S) = ({c'_1}^S, {c'_2}^S, Y \cdot {c'_3}^S; X_1^S, X_2^S, g^S).
\end{aligned}$$

We thus have:

$$\mathsf{Random}(\mathsf{vk}, \mathsf{pk}, c, \sigma; r'_1, r'_2, s') = \mathsf{Sign}(\mathsf{sk}, \mathsf{pk}, c'; s + s'), \text{ where } c' = \mathsf{Encrypt}(\mathsf{pk}, \mathsf{vk}, M; r_1 + r'_1, r_2 + r'_2).$$

□

Theorem 4.2.3 *The Waters signature on linear ciphertexts is unforgeable (in the UF sense) under the* CDH *assumption in* $\mathbb{G}$.

Proof: Let us denote $\mathcal{SC}$ our above signature on ciphertexts (but omit it in the subscripts for clarity), and $\mathcal{S}$ the Waters signature scheme. We know that the latter is existentially unforgeable under the CDH assumption. Let us assume that $\mathcal{A}$ is able to break the unforgeability of $\mathcal{SC}$. We will build an adversary $\mathcal{B}$ against that Waters signature scheme. We note that $\mathcal{B}$ generated the parameters for the commitments for the proof Π of knowledge of M, r_1 and r_2, so that it can extract the values.

- $\mathsf{Setup}(1^\mathfrak{K})$: we first run the $\mathsf{Setup}_\mathcal{S}(1^\mathfrak{K})$ algorithm, from which we get $\mathsf{param}_\mathcal{S} = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \vec{u})$. We set $\mathsf{param}_s = \mathsf{param}_\mathcal{S} = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \vec{u})$, and $\mathsf{param}_e = (p, \mathbb{G}, \mathbb{G}_T, e, g)$. $\mathcal{B}$ sets the commitment parameters so that it can extract committed values.

- $\mathsf{KeyGen}_\mathcal{E}(\mathsf{param}_e)$: for each new key request, $\mathcal{B}$ chooses two random scalars $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_p$, which define the secret key $\mathsf{dk} = (x_1, x_2)$, and the public key as $\mathsf{pk} = (X_1 = g^{x_1}, X_2 = g^{x_2})$.

- $\mathsf{KeyGen}_\mathcal{S}(\mathsf{param}_s)$: for the unique signing key request, one gets the verification key $\mathsf{vk}_\mathcal{S}$ from the Waters EUF-CMA security game. $\mathcal{B}$ sets $\mathsf{vk} = \mathsf{vk}_\mathcal{S}$.

- $\mathcal{A}$ can now access a signing oracle, with queries of the form $\mathsf{Sign}(\mathsf{vk}, \mathsf{pk}, \cdot)$, for any pk and ciphertext of its choice. But the ciphertext looks like $c = (c_1, c_2, c_3)$ together with $\Pi = (\Pi_M, \Pi_r)$.
  - If the tuple (c, Π) is not valid, then $\mathcal{B}$ returns $\perp$;
  - Otherwise, $\mathcal{B}$ can extract M from the proof of knowledge Π_M, that contains a bit-by-bit extractable commitment C_M of M. It then queries $\mathsf{Sign}_\mathcal{S}(\mathsf{sk}_\mathcal{S}, M)$ to the signing oracle, and adds (vk, M) to the SM set. It receives back $\sigma' = (\sigma'_1 = \mathsf{sk} \cdot \mathcal{F}(M)^s, \sigma'_2 = g^{-s})$. $\mathcal{B}$ can also

extract r_1 and r_2 from the bit-by-bit commitments C_1 and C_2 included in the proof Π_r of knowledge of r_1 and r_2. It then returns the signature σ defined as

$$\left(\begin{array}{ll} \sigma_1 = {\sigma'_2}^{-r_1 x_1} = g^{s r_1 x_1} = X_1^{s r_1}, & \sigma_2 = {\sigma'_2}^{-r_2 x_2} = X_2^{s r_2}, \\ \sigma_3 = \sigma'_1 {\sigma'_2}^{-r_1 - r_2} = \mathsf{sk} \cdot \mathcal{F}(M)^s \cdot g^{s(r_1+r_2)}, & \sigma_4 = {\sigma'_2}^{-x_1} = g^{s x_1} = X_1^s, \\ \sigma_5 = {\sigma'_2}^{-x_2} = g^{s x_2} = X_2^s, & \sigma_6 = {\sigma'_2}^{-1} = g^s \end{array}\right)$$

- After a polynomial number of queries, $\mathcal{A}$ outputs, with non-negligible probability, a valid signature σ on a valid ciphertext (c, Π). As above, one can extract the message M, and for a valid forgery, one needs $M \neq \perp$ and $(M, \mathsf{vk}) \notin \mathsf{SM}$. Using SigExt, as shown above, one thus gets a valid Waters signature on M: $\sigma_{\mathcal{B}} = (\sigma_3/(\sigma_1^{1/x_1}\sigma_2^{1/x_2}) = \mathsf{sk} \cdot \mathcal{F}(M)^s, \sigma_6^{-1} = g^{-s})$. This breaks the EUF-CMA property of the Waters signature scheme, that holds under the CDH assumption.

□

4.2.2 An Efficient Instantiation

The construction in the previous section is a concrete and feasible signature on randomizable ciphertexts, which is furthermore extractable, and even in a strong way. We have thus achieved our goal, and all the applications we had in mind can benefit from it. The main drawback, from an efficiency point of view, are the bit-by-bit commitments C_M, C_1 and C_2 of M, r_1 and r_2, respectively. Whereas the message M to be signed could be short (and even a single bit for voting schemes), r_1 and r_2 are necessarily large (the bit length of the order of the group). For a k-bit long message M, Π_M (composed of C_M and a proof) consists of $9\,k + 2$ group elements. The random coins r_1 and r_2 being ℓ-bit long, Π_r (which includes C_1, C_2 and the proof) requires $18\,\ell + 4$ group elements. We now revisit the Waters signature scheme, which will allow us to trade the costly bit-by-bit commitments C_1 and C_2, with two extra-parameters group elements R_1, R_2.

The main idea for the construction is to build a scheme which is unforgeable against a stronger kind of chosen-message attack under the same assumption: the adversary can submit "extended messages" $(M, R_1 = g^{r_1}, R_2 = g^{r_2}, Y_1 = Y^{r_1}, Y_2 = Y^{r_2})$ and the oracle replies with the tuple $(\mathsf{sk} \cdot (\mathcal{F}(M) R_1 R_2)^s, g^{-s}, R_1^{-s}, R_2^{-s})$. We name this attack *chosen-extended-message attack* and note that this security notion implies the classical one, since querying $(M, 1_{\mathbb{G}}, 1_{\mathbb{G}}, \mathsf{vk})$ yields a signature on M. Intuitively, the extra parameters (R_1, R_2) will allow simulation of the signature on the ciphertext without having to know the random coins r_1 and r_2 explicitly.

Revisited Waters Signature

Our variant is defined by the four algorithms.

- $\mathsf{Setup}(1^{\mathfrak{K}})$: The scheme is defined over a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, where $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an admissible bilinear map, $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order p, generated by g and $e(g, g)$ respectively.

  We will sign messages $M = (M_1, \ldots, M_k) \in \{0,1\}^k$. The parameters are a randomly chosen generator $h \xleftarrow{\$} \mathbb{G}$ and a vector $\vec{u} = (u_0, \ldots, u_k) \xleftarrow{\$} \mathbb{G}^{k+1}$ where k is a polynomial in $\mathfrak{K}$, which defines the *Waters Hash* as $\mathcal{F}(M) = u_0 \prod_{i=1}^{k} u_i^{M_i}$. We set $\mathsf{param} = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \vec{u})$.

- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$: Choose a random scalar $y \xleftarrow{\$} \mathbb{Z}_p$, which defines the public key $\mathsf{vk} = Y = g^y$, and the secret key as $\mathsf{sk} = Z = h^y$.

- $\mathsf{Sign}(\mathsf{sk} = Z, M, R_1, R_2, Y_1, Y_2; s)$: First check the consistency of (R_1, R_2, Y_1, Y_2): $e(R_1, Y) \stackrel{?}{=} e(g, Y_1), e(R_2, Y) \stackrel{?}{=} e(g, Y_2)$ then this guarantees that there exists (r_1, r_2) such that $R_i = g^{r_i}, Y_i = Y^{r_i}$. Choose a random $s \xleftarrow{\$} \mathbb{Z}_p$ and define the signature as $\sigma = \big(\sigma_1 = Z \cdot (\mathcal{F}(M) R_1 R_2)^s, \sigma_2 = g^{-s}, \sigma_3 = R_1^{-s}, \sigma_4 = R_2^{-s}\big)$. Again, we may replace the input message M by the pair $(\mathcal{F}(M), \Pi_M)$.

- $\mathsf{Verif}(\mathsf{vk} = Y, M, R_1, R_2, Y_1, Y_2, \sigma)$: Check whether $e(g, \sigma_1) \cdot e(\mathcal{F}(M) R_1 R_2, \sigma_2) \stackrel{?}{=} e(Y, h)$, $e(g, \sigma_3) \stackrel{?}{=} e(\sigma_2, R_1)$, and $e(g, \sigma_4) \stackrel{?}{=} e(\sigma_2, R_2)$, as well as the consistency of (R_1, R_2, Y_1, Y_2).

To randomize a signature, we define: $\mathsf{Random}(\mathsf{vk}, (F, \Pi_M), R_1, R_2, Y_1, Y_2, \sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4); s')$ to output $\sigma' = (\sigma_1 \cdot (F R_1 R_2)^{s'}, \sigma_2 \cdot g^{-s'}, \sigma_3 \cdot R_1^{-s'}, \sigma_4 \cdot R_2^{-s'})$, for a random $s' \xleftarrow{\$} \mathbb{Z}_p$. This simply changes the initial randomness s to $s + s' \mod p$. Hence, if s' is uniform then the internal randomness of σ' is uniform in $\mathbb{Z}_p$.

Theorem 4.2.4 *Our variant of the Waters signature scheme is randomizable, and existentially unforgeable under chosen-extended-message attacks if the* CDH *assumption holds.*

The proof of unforgeability is similar to that for the original Waters scheme:

Proof: Let $\mathcal{A}$ be an adversary breaking the existential unforgeability of the above signature scheme, i.e. after at most q_s signing queries, it succeeds in building a new signature with probability ϵ. Let $(g, A = g^a, B = g^b)$ be a CDH-instance. We show how an adversary $\mathcal{B}$ can compute g^{ab} thanks to $\mathcal{A}$.

$\mathsf{Setup}_{\mathcal{S}}$**:** Pick a random position $j \xleftarrow{\$} \{0, \ldots, k\}$, choose random indices $y_0, y_1, \ldots, y_k \xleftarrow{\$} \{0, \ldots, 2q_s - 1\}$, and random scalars $z_0, z_1, \ldots, z_k \xleftarrow{\$} \mathbb{Z}_p$. One defines $Y = A = g^a$, $h = B = g^b$, $u_0 = h^{y_0 - 2jq_s} g^{z_0}$, $u_i = h^{y_i} g^{z_i}$.

Signing queries**:** To answer a signing query on a message $M = (M_i)$, we define

$$H = -2jq_s + y_0 + \sum_i y_i M_i, \quad J = z_0 + \sum_i z_i M_i : \mathcal{F}(M) = h^H g^J.$$

If $H \equiv 0 \pmod p$ then the simulator aborts, otherwise he sets:

$$\sigma = (Y^{-J/H}(Y_1 Y_2)^{-1/H}(\mathcal{F}(M) R_1 R_2)^s, Y^{1/H} g^{-s}, Y_1^{1/H} R_1^{-s}, Y_2^{1/H} R_2^{-s}).$$

Defining $\tilde{\mu} = s - a/H$, we have:

$$\begin{aligned}
\sigma &= \big(Y^{-J/H}(Y_1 Y_2)^{-1/H}(h^H g^J R_1 R_2)^s,\ Y^{1/H} g^{-s},\ Y_1^{1/H} R_1^{-s},\ Y_2^{1/H} R_2^{-s}\big) \\
&= \big(Y^{-J/H} Y^{-(r_1+r_2)/H}(h^a g^{Ja/H} g^{(r_1+r_2)a/H})(\mathcal{F}(M) R_1 R_2)^{\tilde{\mu}},\ Y^{1/H} g^{-a/H} g^{-\tilde{\mu}},\ Y_1^{1/H} R_1^{-s},\ Y_2^{1/H} R_2^{-s}\big) \\
&= \big(h^a (\mathcal{F}(M) R_1 R_2)^{\tilde{\mu}},\ g^{-\tilde{\mu}},\ R_1^{-\tilde{\mu}},\ R_2^{-\tilde{\mu}}\big).
\end{aligned}$$

After at most q_s signing queries $\mathcal{A}$ outputs a forgery $\sigma^* = (\sigma_1^*, \sigma_2^*, \sigma_3^*, \sigma_4^*)$ on M^*. As before, we define

$$H^* = -2jq_s + y_0 + \sum_i y_i M_i^*, \quad J^* = z_0 + \sum_i z_i M_i^* : \mathcal{F}(M^*) = h^{H^*} g^{J^*}.$$

If $H^* \not\equiv 0 \pmod p$ then abort, otherwise, for some μ^*, $\sigma^* = (h^a(\mathcal{F}(M^*) R_1 R_2)^{\mu^*}, g^{-\mu^*}, R_1^{-\mu^*}, R_2^{-\mu^*})$, and thus $\sigma^* = (h^a g^{J^*\mu^*}(R_1 R_2)^{\mu^*}, g^{-\mu^*}, R_1^{-\mu^*}, R_2^{-\mu^*})$. As a consequence, $\sigma_1^*(\sigma_2^*)^{J^*}\sigma_3\sigma_4 = h^a = g^{ab}$: one has solved the CDH problem.

Success Probability: The Waters hash function is $(1, q)$-programmable , therefore the previous simulation succeeds with non negligible probability $(\Theta(\epsilon/q_s\sqrt{k}))$, and so $\mathcal{B}$ is an efficient adversary against CDH. □

Signatures on Encrypted Messages

In our new scheme, we will sign a linear encryption of $F = \mathcal{F}(M)$ using our Revisited Waters signatures:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: The scheme is based on a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, which constitutes the parameters param_e for encryption. For the signing part, we require moreover a vector $\vec{u} = (u_0, \ldots, u_k) \xleftarrow{\$} \mathbb{G}^{k+1}$, and a generator $h \xleftarrow{\$} \mathbb{G}$ and define $\mathsf{param}_s = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \vec{u})$.
- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param}_e)$: Choose two random scalars $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_p$, which define the secret key $\mathsf{dk} = (x_1, x_2)$, and the public key as $\mathsf{pk} = (X_1 = g^{x_1}, X_2 = g^{x_2})$.
- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param}_s)$: Choose a random scalar $y \xleftarrow{\$} \mathbb{Z}_p$, which defines the public key as $\mathsf{vk} = Y = g^y$, and the secret key as $\mathsf{sk} = Z = h^y$.
- $\mathsf{Encrypt}\big(\mathsf{pk} = (X_1, X_2), \mathsf{vk} = Y, M; (r_1, r_2)\big)$: For a message $M \in \{0,1\}^k$ and random scalars $r_1, r_2 \in \mathbb{Z}_p$, define the ciphertext as $c = \big(c_1 = X_1^{r_1}, c_2 = X_2^{r_2}, c_3 = g^{r_1+r_2} \cdot \mathcal{F}(M)\big)$. To guarantee our notion of unforgeability of signatures on ciphertexts, we add proofs of knowledge of M and an image of r_1 and r_2:

– Proof Π_r contains the commitments $C_r = (C_1 = \mathcal{C}(Y^{r_1}), C_2 = \mathcal{C}(Y^{r_2}))$, from which the simulator can extract Y_1, Y_2 in the reduction (see below). Π_r moreover contains proofs of consistency: $e(\underline{C_i}, X_i) = e(c_i, Y)$. These equations are linear pairing product equations. We require 6 group elements for the commitments and 6 for the proofs, thus 12 group elements instead of $18\,\ell + 4$ in the previous construction.
– Proof Π_M proves knowledge of M s.t. $\mathcal{F}(M)$ is encrypted in c. It consists of a bit-by-bit commitment $C_M = (\mathcal{C}'(M_1), \ldots, \mathcal{C}'(M_k))$ and proofs that each committed value is a bit ($6\,k$ group elements); moreover, a proof that c_3 is well-formed: $e(c_3, Y) = e(u_0 \prod_{i\in\{1,\ldots,k\}} u_i^{M_i}), Y) \cdot e(\underline{C_1 C_2}, g)$, which is a linear pairing product equation (3 additional group elements). Π_M is therefore composed of $9\,k + 3$ group elements.

The global proof (containing the commitments) Π consists therefore of $9\,k + 15$ group elements (instead of $9\,k + 18\,\ell + 6$ when using the original Waters scheme), where k and ℓ are the bit lengths of the message M and elements of $\mathbb{G}$, respectively.

- $\mathsf{Sign}\big(\mathsf{sk} = Z, \mathsf{pk} = (X_1, X_2), (c = (c_1, c_2, c_3), \Pi); s\big)$: To sign a ciphertext $c = (c_1, c_2, c_3)$, first check if Π is valid, and if so, output

$$\sigma = (c_1^s,\ c_2^s,\ Z \cdot c_3^s;\ X_1^s,\ X_2^s, g^s)\ .$$

- $\mathsf{Decrypt}\big(\mathsf{dk} = (x_1, x_2), \mathsf{vk} = Y, (c = (c_1, c_2, c_3), \Pi)\big)$: On a valid ciphertext (verifiable via Π), knowing the decryption key $\mathsf{dk} = (x_1, x_2)$, one can obtain $F = \mathcal{F}(M)$ since $F = c_3/(c_1^{1/x_1} c_2^{1/x_2})$.

- $\mathsf{Verif}\big(\mathsf{vk} = Y, \mathsf{pk} = (X_1, X_2), (c = (c_1, c_2, c_3), \Pi), \sigma = (\sigma_1, \sigma_2, \sigma_3; \sigma_4, \sigma_5, \sigma_6)\big)$: In order to verify the signature, one verifies Π and checks whether the following pairing equations hold: $e(\sigma_3, g) = e(h, Y) \cdot e(c_3, \sigma_6)$ and

$$\begin{aligned} e(\sigma_1, X_1) &= e(c_1, \sigma_4) & \qquad e(\sigma_2, X_2) &= e(c_2, \sigma_5) \\ e(\sigma_1, g) &= e(c_1, \sigma_6) & \qquad e(\sigma_2, g) &= e(c_2, \sigma_6) \end{aligned}$$

- $\mathsf{Random}\big(\mathsf{vk} = Y, \mathsf{pk} = (X_1, X_2), (c = (c_1, c_2, c_3), \Pi), \sigma; r_1', r_2', s'\big)$: In order to randomize the signature and the ciphertext, the algorithm outputs:

$$\begin{aligned} c' = {} & \big(c_1 \cdot X_1^{r_1'}, c_2 \cdot X_2^{r_2'}, c_3 \cdot g^{r_1'+r_2'}\big) \\ \sigma' = {} & \left(\begin{array}{lll} \sigma_1 \cdot c_1^{s'} \cdot \sigma_4^{r_1'} \cdot X_1^{r_1' s'}, & \sigma_2 \cdot c_2^{s'} \cdot \sigma_5^{r_2'} \cdot X_2^{r_2' s'}, & \sigma_3 \cdot c_3^{s'} \cdot \sigma_6^{r_1'+r_2'} \cdot g^{(r_1'+r_2')s'}, \\ \sigma_4 \cdot X_1^{s'}, & \sigma_5 \cdot X_2^{s'}, & \sigma_6 \cdot g^{s'} \end{array} \right) \end{aligned}$$

together with a randomization Π' of Π.

- $\mathsf{SigExt}\big(\mathsf{dk} = (x_1, x_2), \mathsf{vk}, (c = (c_1, c_2, c_3), \Pi), \sigma\big)$: Return the following signature: $\Sigma = \big(\Sigma_1 = \sigma_3/(\sigma_1^{1/x_1} \sigma_2^{1/x_2}), \Sigma_2 = \sigma_6^{-1}\big)$, which is a valid signature on M:

$$\begin{aligned} &\Sigma_1 = \sigma_3/(\sigma_1^{1/x_1} \sigma_2^{1/x_2}) = Z \cdot g^{s(r_1+r_2)} \cdot \mathcal{F}(M)^s / g^{s r_1} g^{s r_2} = Z \cdot \mathcal{F}(M)^s, \\ &\Sigma_2 = \sigma_6^{-1} = g^{-s}. \end{aligned}$$

The same can be obtained from the coins (r_1, r_2) used for encryption.

Theorem 4.2.5 *The above scheme is randomizable and unforgeable (in the UF sense) under the* CDH *assumption in* $\mathbb{G}$.

Correctness of Random follows from inspection of the construction of c' and σ' and the fact that Groth-Sahai proofs are randomizable.

Since we have proved that our variant of Waters signatures is secure under a stronger kind of attack, we can use it for an appropriate simulation of the signing oracle.

Proof: Let us denote $\mathcal{SC}$ our above signature on ciphertexts (but omit it in the subscripts for clarity), and $\mathcal{S}$ our variant of Waters signature scheme. We know that the latter is existentially unforgeable against chosen-extended-message attacks under the CDH assumption. Let us assume that $\mathcal{A}$ is able to break the unforgeability of $\mathcal{SC}$. We will build an adversary $\mathcal{B}$ against our variant of Waters signature scheme. We note that $\mathcal{B}$ generated the parameters for the commitments for the proof Π of knowledge of M, g^{r_1} and g^{r_2}, so that it can extract the values.

- $\mathsf{Setup}(1^{\mathfrak{K}})$: we first run the $\mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$ algorithm, from which we get $\mathsf{param}_{\mathcal{S}} = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \vec{u})$. We set $\mathsf{param}_s = \mathsf{param}_{\mathcal{S}} = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \vec{u})$, and $\mathsf{param}_e = (p, \mathbb{G}, \mathbb{G}_T, e, g)$. $\mathcal{B}$ sets the commitment parameters so that it can extract committed values.

- $\mathsf{KeyGen}_{\mathcal{E}}(\mathsf{param}_e)$: for each new key request, $\mathcal{B}$ chooses two random scalars $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_p$, which define the secret key $\mathsf{dk} = (x_1, x_2)$, and the public key as $\mathsf{pk} = (X_1 = g^{x_1}, X_2 = g^{x_2})$.

- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param}_s)$: for the unique signing key request, one gets the verification key $\mathsf{vk}_{\mathcal{S}}$ from our variant of the Waters unforgeability security game. $\mathcal{B}$ sets $\mathsf{vk} = X = \mathsf{vk}_{\mathcal{S}}$.

- $\mathcal{A}$ can now access a signing oracle, with queries of the form $\mathsf{Sign}(\mathsf{vk}, \mathsf{pk}, \cdot)$, for any pk and ciphertext of its choice. But the ciphertext looks like $c = (c_1, c_2, c_3)$ together with Π, that contains C_M (with extractable M), $C_r = (C_1, C_2, C_3)$ (with extractable g^{r_1}, g^{r_2} and $X^{r_1+r_2}$).

  - If the tuple (c, Π) is not valid, then $\mathcal{B}$ returns $\perp$;
  - Otherwise, $\mathcal{B}$ can extract M from the bit-by-bit proof of knowledge Π_M, as well as $Y_1 = X^{r_1}$, $Y_2 = X^{r_2}$ from C_r and using dk, $R_1 = c_1^{x_1}, R_2 = c_2^{x_2}$. It then queries $\mathsf{Sign}_{\mathcal{S}}(\mathsf{sk}_{\mathcal{S}}, M, R_1, R_2, Y_1, Y_2)$ to the extended-message signing oracle, and adds (vk, M) to the SM set. It receives back $\sigma' = (\sigma'_1 = \mathsf{sk} \cdot (\mathcal{F}(M) R_1 R_2)^s, \sigma'_2 = g^{-s}, \sigma'_3 = R_1^{-s}, \sigma'_4 = R_2^{-s})$.
    It then returns

$$\sigma = \left(\begin{array}{lll} \sigma_1 = {\sigma'_3}^{-x_1} = g^{s r_1 x_1} = X_1^{s r_1}, & \sigma_2 = {\sigma'_4}^{-x_2} = g^{s r_2 x_2} = X_2^{s r_2}, & \sigma_3 = \sigma'_1 = \mathsf{sk} \cdot \mathcal{F}(M)^s \cdot g^{s(r_1+r_2)}, \\ \sigma_4 = {\sigma'_2}^{-x_1} = g^{s x_1} = X_1^s, & \sigma_5 = {\sigma'_2}^{-x_2} = g^{s x_2} = X_2^s, & \sigma_6 = {\sigma'_2}^{-1} = g^s \end{array} \right)$$

- After a polynomial number of queries, $\mathcal{A}$ outputs, with non-negligible probability, a valid signature σ on a valid ciphertext (c, Π). As above, from Π and the extraction key, $\mathcal{B}$ can extract the message M. For a valid forgery, one needs $M \neq \perp$ and $(M, \mathsf{vk}) \notin \mathsf{SM}$. Again, from Π, the extraction key and the commitment key, $\mathcal{B}$ can obtain (R_1, R_2, Y_1, Y_2). One sets

$$\sigma_{\mathcal{B}} = (\Sigma_1 = \mathsf{sk}(\mathcal{F}(M) R_1 R_2)^s, \quad \Sigma_2 = \sigma_6^{-1} = g^{-s}, \quad \Sigma_3 = \sigma_1^{1/x_1} = R_1^s, \quad \Sigma_4 = \sigma_2^{1/x_2} = R_2^s).$$

  And so $(M, R_1, R_2, Y_1, Y_2, \sigma_{\mathcal{B}})$ is a valid forgery. This breaks the security of our variant of the Waters signature scheme, that holds under the CDH assumption.

□

The following table details the size of a ciphertext-signature pair, where the parameter k denotes the bit length of a message:

Symmetric Pairing	$\mathbb{G}$
Waters + Linear	$9\,k + 24$

Asymmetric Pairing	$\mathbb{G}_1$	$\mathbb{G}_2$
Waters + ElGamal	$6\,k + 7$	$6\,k + 5$

4.3 First Applications

So, we have just introduced a basic version of our *extractable signatures on randomizable ciphertexts*, a new primitive that has many applications to anonymity. The first straightforward application is to blind signatures, which yields a similar (yet more efficient) result to [MSF10, GK08]; however, this does not exploit all the power of our new tool. A more interesting application is to receipt-free voting schemes. We discuss this in the following and then show how to construct variants of blind signatures from our primitive.

4.3.1 Non-interactive Receipt-Free E-voting

In voting schemes, anonymity is a crucial property: nobody should be able to learn the content of my vote. This can be achieved with homomorphic encryption schemes. Basically each user will compute his vote v_i and then commit it into c_i, and then, through the homomorphic property of the encryption scheme the voting center will compute $f(c_1, \ldots, c_n)$ and then open it to solely obtain the global result of the election.

However, this does not address the problem of *vote sellers*: a voter may sell his vote and then reveal/prove the content of his encrypted vote to the buyer. He could do so by simply revealing the randomness used when encrypting the vote, which allows to verify that a claimed message was encrypted.

A classical approach to prevent vote selling uses heavy interactive techniques based on randomizable encryption schemes and designated-verifier zero-knowledge proofs: the voter encrypts his vote v as $\mathbf{c}$ and additionally signs it to bar any modification by the voting center. But before doing so, the voting center randomizes $\mathbf{c}$ into $\mathbf{c}'$ (which cannot be opened by the voter anymore since he no longer knows the random coins) and then proves that $\mathbf{c}$ and $\mathbf{c}'$ contain the same plaintext. This proof must be non-transferable, otherwise the voter could open $\mathbf{c}$ (by revealing the random coins) and transfer the proof to the buyer, which together yields a proof of opening for $\mathbf{c}'$. The used proof is thus a designated-verifier zero-knowledge proof. Finally, after receiving $\mathbf{c}'$ and being convinced by the proof, the voter signs $\mathbf{c}'$.

Our randomizable signatures on ciphertexts allow to avoid interactions altogether: all a voter does is encrypt his vote v as $\mathbf{c}$ and make a signature σ on $\mathbf{c}$. The voting center can now consistently randomize both $\mathbf{c}$ and σ as $\mathbf{c}'$ and σ', so that the randomness used in $\mathbf{c}'$ is unknown to the signer, who is however guaranteed that the vote was not modified by the voting center because of the unforgeability notion for ESRC: nobody can generate a signature on a ciphertext that contains a different plaintext. We have thus constructed a non-interactive *receipt-free* voting scheme.

Since our ESRC candidates use not only randomizable but homomorphic encryption schemes (the encryption of the vote is actually the bit-commitments of the M_i's, which are either linear encryptions or ElGamal encryptions of g^{M_i}), classical techniques for voting schemes with homomorphic encryption and threshold decryption can be used [BFP+01]: there is no risk for the signature on the ciphertext to be converted into a signature on the plaintext if the board of authorities uses the decryption capability on the encrypted tally only.

If the vote consists of one box to be checked, the size of the ballot is only 33 group elements in the instantiation with linear encryption, and even smaller for the instantiation using ElGamal: 13 $\mathbb{G}_1$ elements and 11 $\mathbb{G}_2$ elements. Furthermore, if the vote consists of several (say k) boxes to be checked or not, with various constraints, the ballot size grows only slowly in k, since while the votes are committed bit by bit, the proofs can be global. Hence, the size basically corresponds to the signature on a ciphertext of a k-bit message. The extended ciphertexts already contains proofs that plaintexts are bits only, and all the proofs are randomizable.

4.3.2 Blind Signatures and Variants

Since the beginning of e-cash, blind signatures have been their most important tool. They provide an interactive protocol between a bank and a user, letting a user have a message signed by the bank without revealing it. Moreover, the message-signature pair obtained by the user is uncorrelated to the view of the protocol execution by the bank, which enables the user to withdraw anonymous coins. Several signature schemes have been turned into blind signature schemes. The best-known is the first scheme by Chaum [Cha83], which is derived from RSA signatures [RSA78], and has been proven secure [BNPS01] under the one-more RSA assumption in the random-oracle model [BR93]. As defined in [PS96, PS00], for e-cash, the security requirement is the resistance to one-more forgeries: after interacting q times with the signer, an adversary should not be able to output more than q valid signed messages.

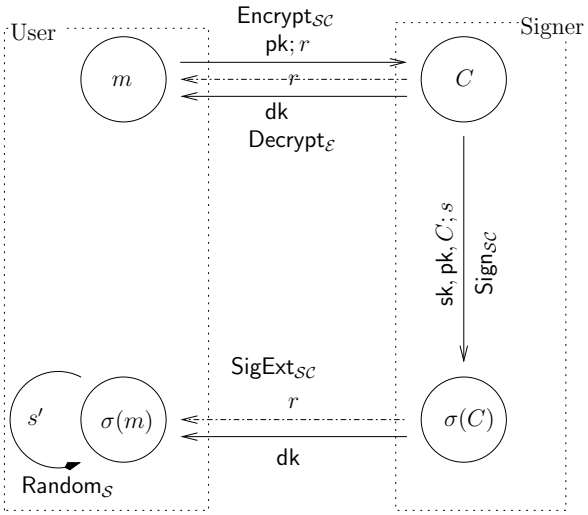
With *Extractable Signatures on Randomizable Ciphertexts*, one can build a computationally blind signature scheme: the user encrypts the message m into the ciphertext $\mathbf{c}$ under his own key, and asks for a signature on $\mathbf{c}$. He gets back a signature on the ciphertext $\mathbf{c}$ from which he can then extract σ, a valid signature on m. This signature is not yet blind, since the signer knows the coins used to compute it, and can thus link σ to the transcript. However, due to the randomizability of the signature, the user can randomize σ into σ' that is a secure blind signature:

- the blindness property relies on the semantic security of the encryption scheme (here DLin) and the randomization, which is information-theoretic;
- the one-more unforgeability relies on unforgeability of the signature scheme (here CDH), since the user cannot generate a signature for a message that has not been asked, encrypted, to the signer. Of course, we do not obtain strong one-more unforgeability (where several signatures on the same message would be counted several times), which is impossible with randomizable signatures.

This construction is similar to [GK08] but with better efficiency and much less bandwidth consumption since the latter relies on inefficient NIZK techniques [DFN06].

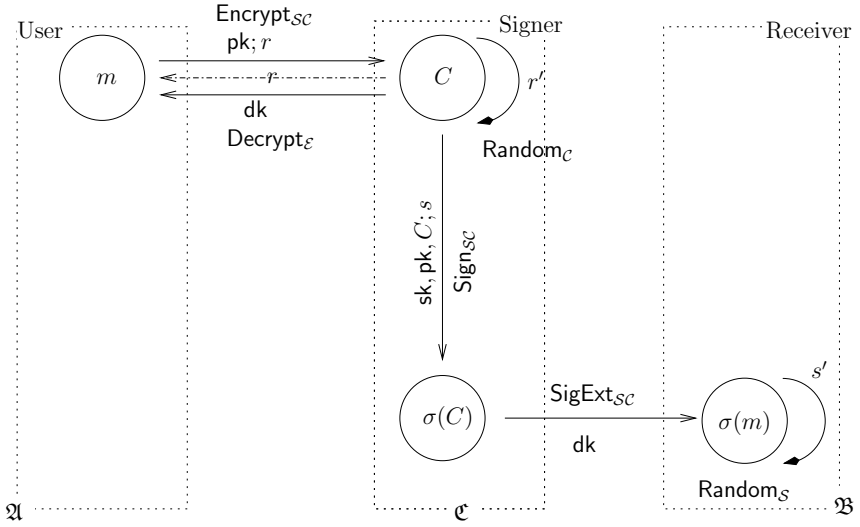
One-Round Fair Blind Signatures

With a *strong* extractable randomizable signature on ciphertexts, we get more than just standard blind signatures: we have *fair* blind signatures [SPC95]. Using a *strong* Extractable Signature on Randomizable Ciphertext scheme, the user does not need to encrypt m under his own key, since the random coins suffice to extract the signature. He can thus encrypt the message m under a tracing authority's key. Using the decryption key, the authority can extract the message from $\mathbf{c}$ (or at least check if $\mathbf{c}$ encrypts a purported message) w.r.t. the signed message and thus revoke anonymity in case of abuse.



One-Round Three-Party Blind Signatures

Our primitive also allows to design a *three-party* blind signature scheme, which we define as follows: a party $\mathfrak{A}$ makes a signer $\mathfrak{C}$ sign a message m for $\mathfrak{B}$ so that neither $\mathfrak{A}$ nor $\mathfrak{C}$ can later link the final message-signature pair (for $\mathfrak{A}$ among all the signatures for the message m, and for $\mathfrak{C}$ among all the valid message-signature pairs). To realize this primitive, the party $\mathfrak{A}$ encrypts the message m under the key of $\mathfrak{B}$, and sends it to the signer $\mathfrak{C}$, who signs the ciphertext and applies the randomization algorithm to the ciphertext-signature pair (this is useful only in case $\mathfrak{A}$ and $\mathfrak{B}$ are distinct, as then $\mathfrak{A}$ does not know the randomness for encryption and therefore cannot extract a signature). $\mathfrak{C}$ sends the encrypted signature to $\mathfrak{B}$ (possibly via $\mathfrak{A}$, who cannot decrypt anyway) and $\mathfrak{B}$ also applies the randomization algorithm (so that $\mathfrak{C}$ does not know the random coins used for signing) and then extracts the signature. With such a 2-flow scheme, $\mathfrak{B}$ can obtain a signature, unknown to $\mathfrak{A}$, on a message chosen by $\mathfrak{A}$, unknown and even indistinguishable from any message-signature pair to $\mathfrak{C}$. Applied to group signatures, such a primitive allows a group manager $\mathfrak{A}$ to add a new member $\mathfrak{B}$ without learning his certificate provided by the authority $\mathfrak{C}$: $\mathfrak{A}$ can define the rights in the message, but only $\mathfrak{B}$ receives the certificate generated by $\mathfrak{C}$.



Additional Properties

Using our instantiation of Extractable Signature on Randomizable Ciphertext, we can define an additional trapdoor: the extraction key for the commitments. It is not intended to be known by anybody (except the simulator in the security analysis), since the commitment key is in the CRS, but one could consider a scenario where it is given to a trusted authority that gets revocation capabilities.

Our construction is similar to previous efficient round-optimal blind signatures [Fuc09, AFG+10] in that it uses Groth-Sahai proofs. However, we rely on standard assumptions only, and our resulting blind signature is a standard Waters signature, which is much shorter (only 2 group elements) than the proof of knowledge of a signature used in all previous constructions.

4.4 To Further Applications of Signatures on Randomizable Ciphertexts

In the previous applications one can see some room for several improvements. First the extraction key is not always required : as long as the signer does not randomize the commitment (*i.e.* in case of Strong Extractability) the committer is able to recover the signature by using his random. A first step is then to switch to a perfectly hiding setting, where such manipulation can still be done, while we remove the extraction key, guaranteeing therefore the impossibility to recover the signature for a powerful eavesdropper.

In our previous applications, the message signed was only chosen by the committer. While it may seem to be a good idea ; in many applications, the signer wants to be able to limit the type of message he can sign. A bank will probably want to impose some date of validity, a poll center may want to say something specific about the election, ... This kind of loophole in standard blind signatures was detailed by Abe and Okamoto [AO00]: the signer has no control over the signed messages (except in some sense the unforgeability which limits their number), and reinforced the proposition for Partially-Blind Signatures made by Abe and Fujisaki [AF96]. We decided to supersede the ciphertext by a tuple composed of the ciphertext itself, a public message chosen by the committer and a public message chosen, possibly on the fly, by the signer. We call our version Signer-Friendly Blind Signature, because instead of having a previous communication where they have to agree on the public part, here the signer can add his own part during the process.

We will combine those two ideas in the next section 4.4.1, page 82.

Discarding the perfect blindness, we will then take advantage of this asynchronous property (the user and the signer can independently choose their inputs) and we will consider the new context where the message to be signed comes from several independent sources that cannot communicate together in section 4.4.2, page 87.

4.4.1 To Perfectly Blind Signature with Partial Blindness

Definition, and security properties

This section presents the global framework and the security model for partially-blind signature schemes, we will focus on stressing the differences with *classical* blind signatures.

Blind signatures introduced a nice feature, however in some circumstances it may be undesirable that requesters can ask the signer to blindly sign any message. For example, in an e-cash scheme, some expiration date information should be embedded in the e-coin, so the bank won't have to store an ever-growing number of coins information for double-spending checking, or a server will want to certify a vote for a specific election only. Partially-blind signatures are thus a natural extension of blind signatures: instead of signing an unknown message, the signer signs a message which contains a shared piece of information in addition to the hidden part. This piece is called info and, in the standard definition, is expected to have been defined before the execution of the protocol.

Partially-Blind Signature Scheme
$\ulcorner\ \mathcal{PBS} = (\mathsf{Setup}_{\mathcal{PBS}}, \mathsf{KeyGen}_{\mathcal{PBS}}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Verif}_{\mathcal{PBS}})$ where

- $\mathsf{Setup}_{\mathcal{PBS}}(1^{\mathfrak{K}})$ generates the global parameters param_{pbs} of the system;
- $\mathsf{KeyGen}_{\mathcal{PBS}}(\mathsf{param}_{pbs})$ generates a pair of keys $(\mathsf{pk}_{\mathcal{PBS}}, \mathsf{sk}_{\mathcal{PBS}})$;
- Signature Issuing is an interactive protocol between the algorithms $\mathcal{S}(\mathsf{sk}_{\mathcal{PBS}}, \mathsf{info})$ and $\mathcal{U}(\mathsf{pk}_{\mathcal{PBS}}, m, \mathsf{info})$, for a message $m \in \{0,1\}^n$ and shared information info. It generates an output σ for the user: $\sigma \leftarrow \langle \mathcal{S}(\mathsf{sk}_{\mathcal{PBS}}, \mathsf{info}), \mathcal{U}(\mathsf{pk}_{\mathcal{PBS}}, m, \mathsf{info}) \rangle$.

- $\mathsf{Verif}_{\mathcal{PBS}}(\mathsf{pk}_{\mathcal{PBS}}, m, \mathsf{info}, \sigma)$ outputs 1 if the signature σ is valid with respect to the message $m\|\mathsf{info}$ and $\mathsf{pk}_{\mathcal{PBS}}$, 0 otherwise.

⌟

Quick note on security: The security requirements are a direct extension of the classical ones: for unforgeability, we consider $m\|\mathsf{info}$ instead of m, and for the blindness, we condition the unlinkability between signatures with the same public part info.

We consider the same public part, in the unlinkability property because otherwise anyone can distinguish which message was signed simply by comparing the public information. The unforgeability is strengthened by considering also the public information so that the signer can be sure that the user won't be able to exploit his signature in another context.

Signer-Friendly Partially-Blind Signature Scheme
⌜ $(\mathsf{Setup}_{\mathcal{PBS}}, \mathsf{KeyGen}_{\mathcal{PBS}}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{Verif}_{\mathcal{PBS}})$ where

- $\mathsf{Setup}(1^{\mathfrak{K}})$ generates the global parameters param_{pbs} of the system;
- $\mathsf{KeyGen}(\mathsf{param}_{pbs})$ generates a pair of keys $(\mathsf{pk}_{\mathcal{PBS}}, \mathsf{sk}_{\mathcal{PBS}})$;
- Signature Issuing: this is an interactive protocol between the algorithms $\mathcal{S}(\mathsf{sk}_{\mathcal{PBS}}, \mathsf{info}, \mathsf{info}_s)$ and $\mathcal{U}(\mathsf{pk}_{\mathcal{PBS}}, m, \mathsf{info})$, for a message $m \in \{0,1\}^n$, public information info, additional public information info_s. It generates an output σ for the user: $\sigma \leftarrow \langle \mathcal{S}(\mathsf{sk}_{\mathcal{PBS}}, \mathsf{info}, \mathsf{info}_s), \mathcal{U}(\mathsf{pk}_{\mathcal{PBS}}, m, \mathsf{info}) \rangle$.
- $\mathsf{Verif}(\mathsf{pk}_{\mathcal{PBS}}, m, \mathsf{info}, \mathsf{info}_s, \sigma)$ outputs 1 if the signature σ is valid with respect to the message $m\|\mathsf{info}\|\mathsf{info}_s$ and $\mathsf{pk}_{\mathcal{PBS}}$, 0 otherwise.

⌟

One can note that with $\mathsf{info}_s = \emptyset$, we have a standard partially-blind signature; whereas the case $\mathsf{info} = \mathsf{info}_s = \emptyset$ is the standard blind signature.

In previous existing partially-blind signature protocols, the participants had to pre-agreed on the public information, in order to be able to run the protocol. Through this definition, it is no longer mandatory and so we handle more cases, but of course they can still do that, and anyway, both the signer and the user can stop the process if the public information is not like the one pre-agreed.

The signer always has the last word in the process, and so if he does not want to sign a specific info, he will simply abort the protocol several times until the shared part suits his will. So, in the following, we decided that it was wiser to let him choose this input. If the user wants a specific word in the final message he can always add it to the blinded message. Intuitively this strengthens the unforgeability notion as the adversary (the user in this case) won't be able to choose the whole message to be signed because of info_s. This is ensured in the security game, because the adversary should outputs valid signatures, therefore they should be done with the chosen info_s. For the blindness property, the adversary should guess on signatures with the same public $\mathsf{info}\|\mathsf{info}_s$ component, if it is not the case we answer with a blind-signature $\perp$. This requirement is crucial, because if the public components are not the same in the two challenges then there is a trivial attack distinguishing the signatures thanks to those parts.

$\mathsf{Exp}^{\mathsf{bl}_b}_{\mathcal{PBS},\mathcal{S}^*}(\mathfrak{K})$
1. $(\mathsf{pk}_{\mathcal{BS}}, m_0, m_1, st_{\texttt{FIND}}, \mathsf{info}, \mathsf{info}_s) \leftarrow \mathcal{S}^*(\texttt{FIND}, 1^{\mathfrak{K}})$;
2. $b \leftarrow \{0,1\}$;
3. $st_{\texttt{ISSUE}} \leftarrow \mathcal{S}^{*\langle \cdot, \mathcal{U}(\mathsf{pk}_{\mathcal{BS}}, m_b) \rangle^1, \langle \cdot, \mathcal{U}(\mathsf{pk}_{\mathcal{BS}}, m_{1-b}, \mathsf{info}, \mathsf{info}_s) \rangle^1}(\texttt{ISSUE}, st_{\texttt{FIND}})$;
4. IF $\sigma_0 = \perp$ OR $\sigma_1 = \perp, (\sigma_0, \sigma_1) \leftarrow (\perp, \perp)$;
5. $b^* \leftarrow \mathcal{S}^*(\texttt{GUESS}, \sigma_0, \sigma_1, st_{\texttt{ISSUE}})$;
5. IF $b = b^*$RETURN 1 ELSE RETURN 0.

Figure 4.3: Blindness for User-Friendly Partially Blind signatures

$\mathcal{PBS}$ is *blind* if, for any polynomial adversary $\mathcal{S}^*$ (malicious signer), the advantage $\mathsf{Succ}^{\mathsf{bl}_b}_{\mathcal{PBS},\mathcal{S}^*}(\mathfrak{K})$ is negligible, where $\mathsf{Succ}^{\mathsf{bl}}_{\mathcal{PBS},\mathcal{S}^*}(\mathfrak{K}) = |\Pr[\mathsf{Exp}^{\mathsf{bl}}_{\mathcal{PBS},\mathcal{S}^*}(\mathfrak{K}) = 1] - 1/2|$, in the security game presented in Figure 4.3. If $\mathcal{S}^*$ refuses to sign one of the input (i.e. $\sigma_i = \perp$), then the two resulting signatures are set to $\perp$, therefore he cannot have any advantage if he decides to prevent the normal game execution and he has to sign both inputs. $\mathcal{S}^*$ is able to chose both pieces of the public information, in the real

$\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{PBS},\mathcal{U}^*}(\mathfrak{K})$
1. $(\mathsf{param}_{bs}) \leftarrow \mathsf{Setup}_{\mathcal{BS}}(1^{\mathfrak{K}})$;
2. $(\mathsf{pk}_{\mathcal{BS}}, \mathsf{sk}_{\mathcal{BS}}) \leftarrow \mathsf{KeyGen}_{\mathcal{BS}}(\mathsf{param}_{bs})$;
3. $\big((m_1, \mathsf{info}_{c,1}, \mathsf{info}_{s,1}, \sigma_1), \ldots, (m_{q_s+1}, \mathsf{info}_{q_s+1}, \mathsf{info}_{s,q_s+1}, \sigma_{q_s+1})\big) \leftarrow \mathcal{U}^{*\mathcal{S}^{q_s}(\mathsf{sk}_{\mathcal{BS}},\cdot)}(\mathsf{pk}_{\mathcal{BS}})$;
4. IF $\exists i \neq j, (m_i, \mathsf{info}_i, \mathsf{info}_{s,i}) = (m_j, \mathsf{info}_j, \mathsf{info}_{s,j})$
   OR $\exists i, \mathsf{Verif}_{\mathcal{BS}}(\mathsf{pk}_{\mathcal{BS}}, m_i, \mathsf{info}_i, \mathsf{info}_{s,i}, \sigma_i) = 0$, RETURN 0
5. ELSE RETURN 1

Figure 4.4: Unforgeability for User-Friendly Partially Blind signatures (One-More Forgery)

case the signer can abort as long as the user's public information does not suit him, however the public information should be the same on both challenged message.

$\mathcal{PBS}$ is *unforgeable* if, for any polynomial adversary $\mathcal{U}^*$ (malicious user), the advantage $\mathsf{Succ}^{\mathsf{uf}}_{\mathcal{PBS},\mathcal{U}^*}(\mathfrak{K})$ is negligible, where $\mathsf{Succ}^{\mathsf{uf}}_{\mathcal{PBS},\mathcal{U}^*}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{PBS},\mathcal{U}^*}(\mathfrak{K}) = 1]$, in the security game presented in Figure 4.4, page 84. In this experiment, the adversary $\mathcal{U}^*$ can interact q_s times with the signing oracle $\mathcal{S}(\mathsf{sk}_{\mathcal{BS}}, \cdot)$ (hence the notation $\mathcal{U}^{*\mathcal{S}^{q_s}(\mathsf{sk}_{\mathcal{BS}},\cdot)}(\mathsf{pk}_{\mathcal{BS}})$) to execute the user-friendly partially blind signature protocol: the adversary should not be able to produce more signatures on distinct tuple $(m, \mathsf{info}, \mathsf{info}_s)$ than interactions with the signer. Once again we consider the adversary has full control over the public information.

Perfectly blind signature with partial blindness

Combining the double linear commitments, and the Waters signature while following the intuition described in our *Signature on Randomizable Ciphertexts*, we design a partially-blind signature scheme, which basically consists in committing the message to be signed. And granted the random coins of the commitment, the user can unblind the signature sent by the signer exploiting the *Strong-Extractability*. Eventually, using the randomizability of the Waters signature, the user breaks all the links that could remain between the message-signature pair and the transaction. Our protocol proceeds as follows, on a commitment of $F = \mathcal{F}(M)$, a public common message info, and a public message info_s chosen by the signer. It is split into five steps, that correspond to an optimal 2-flow protocol: $\mathsf{Blind}_{\mathcal{BS}}$, which is first run by the user, $\mathsf{Sign}_{\mathcal{BS}}$, which is thereafter run by the signer, and $\mathsf{Verif}_{\mathcal{BS}}$, $\mathsf{Unblind}_{\mathcal{BS}}$, $\mathsf{Random}_{\mathcal{S}}$ that are eventually successively run by the user to generate the final signature. We thus have $\mathcal{U} = (\mathsf{Blind}_{\mathcal{BS}}; \mathsf{Verif}_{\mathcal{BS}}, \mathsf{Unblind}_{\mathcal{BS}}, \mathsf{Random}_{\mathcal{S}})$ and $\mathcal{S} = \mathsf{Sign}_{\mathcal{BS}}$:

- $\mathsf{Setup}_{\mathcal{BS}}(1^{\mathfrak{K}})$ first chooses a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$. We need an additional vector $\vec{u} = (u_0, \ldots, u_k) \xleftarrow{\$} \mathbb{G}^{k+1}$ which defines the Waters function $\mathcal{F}$ (where k is a polynomial in $\mathfrak{K}$ and represents the global length of $M||\mathsf{info}||\mathsf{info}_s$), a generator $h \xleftarrow{\$} \mathbb{G}$, and a tuple of Groth-Sahai parameters $(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$ in the perfectly hiding setting with defines the commitment parameters $\mathcal{C}$: $\mathsf{param}_{bs} = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \mathcal{F}, \mathcal{C})$;

- $\mathsf{KeyGen}_{\mathcal{BS}}(\mathsf{param}_{bs})$ chooses a random scalar $x \xleftarrow{\$} \mathbb{Z}_p$, which defines the public key as $\mathsf{pk}_{\mathcal{BS}} = Y = g^x$, and the secret key as $\mathsf{sk}_{\mathcal{BS}} = Z = h^x$;

- Signature Issuing $(\mathcal{S}(\mathsf{sk}_{\mathcal{BS}}, \mathsf{info}, \mathsf{info}_s), \mathcal{U}(\mathsf{pk}_{\mathcal{BS}}, M, \mathsf{info}))$, which is split in the following steps:

  1. $\mathsf{Blind}_{\mathcal{BS}}(M, \mathsf{pk}_{\mathcal{BS}}; (r_1, r_2, r_3))$: For a message $M \in \{0,1\}^\ell$ and random scalars $(r_1, r_2, r_3) \xleftarrow{\$} \mathbb{Z}_p$, define the commitment as $c = \big(c_1 = u_{1,1}^{r_1} u_{3,1}^{r_3}, c_2 = u_{2,2}^{r_2} u_{3,2}^{r_3}, c_3 = g^{r_1+r_2} u_{3,3}^{r_3} \cdot \mathcal{F}(M)\big)$ and compute $Y_{1,2} = Y^{r_1+r_2}, Y_3 = Y^{r_3}$. One also generates additional proofs of validity of the commitment:
     - A proof Π_M of knowledge of M in c, the encrypted $\mathcal{F}(M)$, which consists of a bit-by-bit commitment $C_M = (\mathcal{C}'(M_1), \ldots, \mathcal{C}'(M_\ell))$ and proofs that each committed value is a bit, and a proof that c_3 is well-formed. Π_M is therefore composed of $9\,\ell + 3$ group elements.
     - A proof Π_r containing the commitments $C_r = (\mathcal{C}(Y_{1,2}), \mathcal{C}(Y_3))$ and proofs asserting that they are correctly generated. It requires 9 additional group elements.

     Π thus consists of $9\,\ell + 12$ group elements, where ℓ is the bit-length of the message M

  2. $\mathsf{Sign}_{\mathcal{BS}}(\mathsf{sk}_{\mathcal{BS}}, (c, \Pi), \mathsf{info}, \mathsf{info}_s; s)$: To sign the commitment c, one first checks if the proof Π is valid. It then appends the public message $\mathsf{info} = \mathsf{info}||\mathsf{info}_s$ to c_3 to create $c_3' = c_3 \cdot \prod u_{i+\ell}^{\mathsf{info}_i}$, which thus becomes a commitment of the Waters function evaluation on $M||\mathsf{info}||\mathsf{info}_s$ of

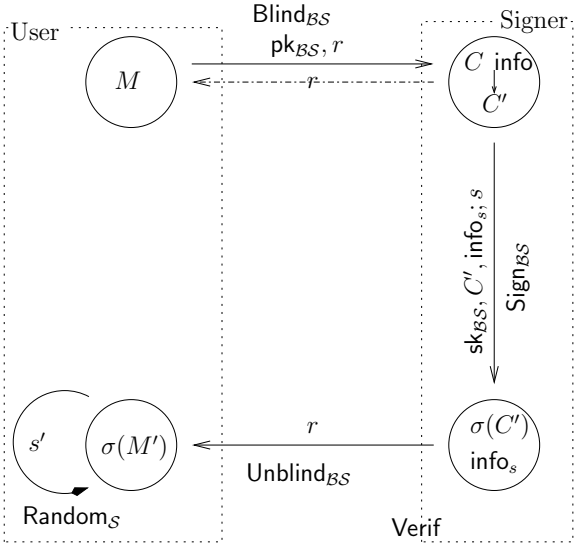
global length k. It eventually outputs $\sigma = (Z \cdot {c'_3}^s, u_{3,3}^s, g^s)$ together with the additional public information info_s, for a random scalar $s \in \mathbb{Z}_p$.

3. – $\mathsf{Verif}(\mathsf{pk}_{\mathcal{BS}}, (c, \mathsf{info}, \mathsf{info}_s), \sigma = (\sigma_1, \sigma_2, \sigma_3))$: In order to check the validity of the signature, one first computes c'_3 as above, and then checks whether the following pairing equations are verified: $e(\sigma_1, g) \stackrel{?}{=} e(h, \mathsf{pk}_{\mathcal{BS}}) \cdot e(c'_3, \sigma_3)$ and $e(\sigma_2, g) \stackrel{?}{=} e(u_{3,3}, \sigma_3)$. If it is not the case, then this is not a valid signature on the original ciphertext, and the blind signature is set as $\Sigma = \perp$.
   – $\mathsf{Unblind}_{\mathcal{BS}}((r_1, r_2, r_3), \mathsf{pk}_{\mathcal{BS}}, (c, \mathsf{info}, \mathsf{info}_s), \sigma)$: If the previous tests are positive, one can use the random coins r_1, r_2, r_3 to get back a valid signature on $M\|\mathsf{info}\|\mathsf{info}_s$: $\sigma' = (\sigma'_1 = \sigma_1/(\sigma_3^{r_1+r_2}\sigma_2^{r_3}), \sigma'_2 = \sigma_3)$, which is a regular valid Waters signature.
   – $\mathsf{Random}_{\mathcal{S}}(\mathsf{pk}_{\mathcal{BS}}, (c, \mathsf{info}, \mathsf{info}_s), \sigma'; s')$: The latter can eventually be rerandomized as any Waters signature to get $\Sigma = (\sigma'_1 \cdot \mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)^{s'}, \sigma'_2 \cdot g^{s'})$.

One can note that Σ is a random Waters signature on $M||\mathsf{info}||\mathsf{info}_s$:

$$\begin{aligned}
\Sigma &= (\sigma'_1 \cdot \mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)^{s'}, \sigma'_2 \cdot g^{s'}) \\
&= (\mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)^{s'} \cdot \sigma_1/(\sigma_3^{r_1+r_2}\sigma_2^{r_3}), g^{s'} \cdot \sigma_3) \\
&= (\mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)^{s'} \cdot Z \cdot {c'_3}^s/(g^{s(r_1+r_2)}u_{3,3}^{sr_3}), g^{s+s'}) \\
&= (\mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)^{s'} \cdot Z \cdot g^{s(r_1+r_2)}u_{3,3}^{sr_3} \cdot \mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)^s/(g^{s(r_1+r_2)}u_{3,3}^{sr_3}), g^{s+s'}) \\
&= (\mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)^{s+s'} \cdot Z, g^{s+s'})
\end{aligned}$$

- $\mathsf{Verif}_{\mathcal{BS}}(\mathsf{pk}_{\mathcal{BS}}, (M, \mathsf{info}, \mathsf{info}_s), \Sigma = (\Sigma_1, \Sigma_2))$: One checks whether the following pairing equation holds (Waters signature): $e(\Sigma_1, g) \stackrel{?}{=} e(h, \mathsf{pk}_{\mathcal{BS}}) \cdot e(\mathcal{F}(M||\mathsf{info}||\mathsf{info}_s), \Sigma_2)$.



A message M can be hidden using random coins r ($\mathsf{Blind}_{\mathcal{BS}}$).
The signer can adapt this commitment and concatenate a public message info_s into the original commitment, with also the common public information info_s, creating a commitment C' on $M' = M||\mathsf{info}||\mathsf{info}_s$.
A signature on the plaintext can be obtained using the randomness r (for $\mathsf{Unblind}_{\mathcal{BS}}$); the result is the same as a direct signature on $M||\mathsf{info}||\mathsf{info}_s$ by the signer.
Randomizing this signature is easy, and prevents the signer to actually know which ciphertext was the one involved.

Figure 4.5: Partially-Blind Signatures with Perfect Blindness

Theorem 4.4.1 *This signer-friendly partially-blind signature scheme is unforgeable under the* CDH *and* DLin *assumption in* $\mathbb{G}$.

Proof: Let us denote $\mathcal{PBS}$ our above partially-blind signature (but omit it in the subscripts for clarity). Let us assume there is an adversary $\mathcal{A}$ against the unforgeability that succeeds within probability ϵ, we will build an adversary $\mathcal{B}$ against the CDH problem.

DLin **Assumption:** The unforgeability means that after q_s interactions with the signer, the adversary manages to output $q_s + 1$ valid message-signature pairs on distinct messages. If the adversary $\mathcal{A}$ can do that with probability ϵ with the above commitment scheme using a perfectly hiding setting, under the DLin assumption, $\mathcal{A}$ can also generate $q_s + 1$ valid message-signature pairs in a perfectly binding setting, with not too small probability ϵ'.

Signer Simulation: Let us thus now consider the above blind signature scheme with a commitment scheme using a perfectly binding setting (named $\mathcal{PBS}'$), and our simulator $\mathcal{B}$ can extract values from the commitments since it knows ν and μ. We thus now assume that $\mathcal{A}$ is able to break the unforgeability of $\mathcal{PBS}'$ with probability ϵ' after q_s interactions with the signer. And we build an adversary $\mathcal{B}$ against the CDH problem: Let $(A = g^a, B = g^b)$ be a CDH-instance in a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$.

We now generate the global parameters using this instance: for simulating $\mathsf{Setup}_{\mathcal{BS}}/\mathsf{KeyGen}_{\mathcal{BS}}$, $\mathcal{B}$ picks a random position $j \xleftarrow{\$} \{0, \ldots, k\}$, chooses random indexes $y_0, y_1, \ldots, y_k \xleftarrow{\$} \{0, \ldots, 2q_s - 1\}$, and random scalars $z_0, z_1, \ldots, z_k \xleftarrow{\$} \mathbb{Z}_p$. One defines $Y = A = g^a$, $h = B = g^b$, $u_0 = h^{y_0 - 2jq_s} g^{z_0}$, and $u_i = h^{y_i} g^{z_i}$ for $i = 1, \ldots, k$. $\mathcal{B}$ also picks two random scalars ν, μ, and generates the Groth-Sahai parameters $(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$ in the perfectly binding setting, and thus with $(\mathbf{u}_1 = (u_{1,1} = g^{x_1}, 1, g), \mathbf{u}_2 = (1, u_{2,2} = g^{x_2}, g), \mathbf{u}_3 = \mathbf{u}_1^{\nu} \odot \mathbf{u}_2^{\mu})$, for two random scalars x_1, x_2. Note that $u_{3,3} = g^{\nu+\mu}$. It outputs $\mathsf{param}_{bs} = (p, \mathbb{G}, \mathbb{G}_T, e, g, h, \mathcal{F}, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$; one can note that the signing key is implicitly defined as $Z = h^a = B^a = g^{ab}$, and is thus the expected Diffie-Hellman value.

To answer a signing query on ciphertext $c = (c_1, c_2, c_3)$, with the additional proofs, one first checks the proof Π. From the proof Π and the commitment secret parameters x_1, x_2, $\mathcal{B}$ can extract M from the bit-by-bit commitments in Π_M, and $Y_{1,2} = Y^{r_1+r_2}$, $Y_3 = Y^{r_3}$, from Π_r, where $c_1 = u_{1,1}^{r_1} u_{3,1}^{r_3}$ and $c_2 = u_{2,2}^{r_2} u_{3,2}^{r_3}$. Furthermore, we can compute $c'_3 = g^{r_1+r_2} u_{3,3}^{r_3} \cdot \mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)$. We then denote $M' = M||\mathsf{info}||\mathsf{info}_s$. $\mathcal{B}$ defines

$$H = -2jq_s + y_0 + \sum_i y_i M'_i, \quad J = z_0 + \sum_i z_i M'_i \quad : \quad \mathcal{F}(M||\mathsf{info}||\mathsf{info}_s) = h^H g^J.$$

If $H \equiv 0 \pmod p$ then $\mathcal{B}$ aborts, otherwise it sets

$$\sigma = (Y^{-J/H}(Y_{1,2} Y_3^{\nu+\mu})^{-1/H} (\mathcal{F}(M||\mathsf{info}||\mathsf{info}_s)(c_1^{1/x_1} c_2^{1/x_2}))^s, (Y^{-1/H} g^s)^{\nu+\mu}, Y^{-1/H} g^s).$$

Defining $\tilde{\mu} = s - a/H$, we have

$$\begin{aligned}
\sigma_1 =& Y^{-J/H}(Y_{1,2} Y_3^{\nu+\mu})^{-1/H} (h^H g^J (c_1^{1/x_1} c_2^{1/x_2}))^s = Y^{-J/H} Y^{-(r_1+r_2)/H} (Y^{\nu+\mu})^{-r_3/H} (h^H g^J g^{r_1+r_2} u_{3,3}^{r_3})^s \\
=& Y^{-J/H} Y^{-(r_1+r_2)/H} Y^{-(\nu+\mu)r_3/H} (h^H g^J g^{r_1+r_2} u_{3,3}^{r_3})^{\tilde{\mu}} (h^a g^{aJ/H} g^{a(r_1+r_2)/H} u_{3,3}^{ar_3/H}) \\
=& Y^{-J/H} Y^{-(r_1+r_2)/H} Y^{-(\nu+\mu)r_3/H} (h^H g^J g^{r_1+r_2} u_{3,3}^{r_3})^{\tilde{\mu}} (Z Y^{J/H} Y^{(r_1+r_2)/H} Y^{(\nu+\mu)r_3/H}) = Z \cdot (c'_3)^{\tilde{\mu}} \\
\sigma_3 =& Y^{-1/H} g^s = Y^{-1/H} g^{\tilde{\mu}+a/H} = g^{\tilde{\mu}} \\
\sigma_2 =& (\sigma_3)^{\nu+\mu} = g^{(\nu+\mu)\tilde{\mu}} = u_{3,3}^{\tilde{\mu}}
\end{aligned}$$

It thus exactly looks like a real signature sent by the signer.

Diffie-Hellman Extraction: After at most q_s signing queries $\mathcal{A}$ outputs q_s+1 valid Waters signatures. Since there are more than the number of signing queries, there is a least one message M^* that is different from all the messages $M||\mathsf{info}||\mathsf{info}_s$ involved in the signing queries. We define

$$H^* = -2jq_s + y_0 + \sum_i y_i M^*_i, \quad J^* = z_0 + \sum_i z_i M^*_i \quad : \quad \mathcal{F}(M^*) = h^{H^*} g^{J^*}.$$

If $H^* \not\equiv 0 \pmod p$ then $\mathcal{B}$ abort, otherwise, for some μ^*, $\sigma^* = (h^a \mathcal{F}(M^*)^{\mu^*}, g^{\mu^*}) = (h^a g^{\mu^* J^*}, g^{\mu^*})$. As a consequence, $\sigma_1^*/(\sigma_2^*)^{J^*} = h^a = g^{ab}$: one has solved the CDH problem.

Success Probability: The Waters hash function is $(1, q)$-programmable, therefore the previous simulation succeeds with non negligible probability $(\Theta(\epsilon/q_s\sqrt{k}))$, and so $\mathcal{B}$ breaks CDH. □

Theorem 4.4.2 *This signer-friendly partially-blind signature scheme achieves perfect blindness.*

Proof: The transcript sent to the signer contains a commitment on the message to be signed, but in a perfectly hiding setting: no information leaks about M. The additional proofs are perfectly witness-indistinguishable and thus do not provide any additional information about M. This is due to the fact that in the Groth-Sahai framework in the perfectly hiding setting, for any message M, committed with randomness r and a message M', one can find random r' such that $c(M, r) = c(M', r')$. Granted the randomizability of the Waters signature, the final output signature is a random signature on $M||\mathsf{info}||\mathsf{info}_s$, on which no information leaked, and so the resulting signature is perfectly independent from the transcript seen by the signer, and any adversary. □

4.4.2 Multi-Blind Signature

We first present a way to obtain a signature on the concatenation of the input messages.

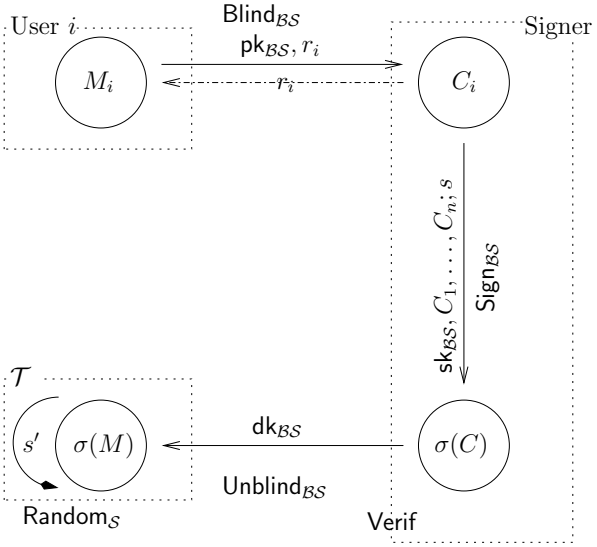
We also present a shorter instantiation which gives a signature on the sum of the input messages. Such a sum can be useful when working on ballots, sensor informations, etc. Since we still apply the Waters signature, this led us use the Waters function programmability over a non-binary alphabet, and so to prove the two results, we showed in section 2.6.4, page 42.

Concatenation

What happens when the original blinded message instead of coming from one single user, is in fact coming from various users? We now present a new way to obtain a blind signature without requiring multiple users to combine their messages before sending it, providing once again a round-optimal way to achieve our goal.

We thus consider a variation of our blind signature scheme. In the Setup phase we no longer create perfectly hiding Groth-Sahai generators, but perfectly binding parameters, so we do not need to compute $u^s_{3,3}$ to run Unblind, since it will be performed with the decryption key and not the random coins. In addition, in this scenario, we do not consider a unique user providing a ciphertext, but several users. As a consequence, the signer will have to produce a signature on a multi-source message, provided as ciphertexts. The signature and the messages will actually be encrypted under a third-party key. The third-party only will be able to extract the message and the signature.

Basically the instantiation is similar to the previous ones in the perfectly binding setting. For the sake of clarity, we remove the partially-blind part, but of course it could be adapted in the same way.



Several messages M_i can be hidden using random coins r_i ($\mathsf{Blind}_{\mathcal{BS}}$) by different users.
The signer can adapt these commitments and concatenate the messages inside them, creating a commitment C on $M = ||_i M_i$.
A signature on the plaintext can be obtained by the tallier using the decryption key $\mathsf{dk}_{\mathcal{BS}}$ (for $\mathsf{Unblind}_{\mathcal{BS}}$); the result is the same as a direct signature on $||M_i$ by the signer.
Randomizing this signature is easy, and prevent the signer from knowing which ciphertexts were involved.

Figure 4.6: Multi-Source Blind Signature on Concatenation

With the previous building blocks, we will sign several commitments of $F_i = \mathcal{F}_i(M_i)$, instead of the standard $(\mathcal{U}, \mathcal{S})$ interactions we now have three main kind of users, $\mathcal{U}_i$, the user i will blind a commitment on $\mathcal{F}_i(M_i)$, $\mathcal{S}$ who signs the blinded message, and $\mathcal{T}$ the tallier who will verify/unblind/randomize this signature:

- $\mathsf{Setup}_{\mathcal{BS}}(1^{\mathfrak{K}})$: In a pairing-friendly environment $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, the algorithm outputs a vector

$$\vec{u} = (u_0, (u_{i,1}, \ldots, u_{i,k})_{1 \le i \le j}) \xleftarrow{\$} \mathbb{G}^{jk+1}$$

  where k is a polynomial in $\mathfrak{K}$, and a generator $h \xleftarrow{\$} \mathbb{G}$. We define $\mathcal{F}_i(M_i) = \prod_\ell u_{i,\ell}^{m_{i,\ell}}$.

- $\mathsf{KeyGen}_{\mathcal{BS}}(\mathsf{param}_{bs})$: Choose $x \xleftarrow{\$} \mathbb{Z}_p$, which defines $\mathsf{pk}_{\mathcal{BS}} = Y = g^x$, and $\mathsf{sk}_{\mathcal{BS}} = Z = h^x$ and generates a pair of perfectly-binding Groth-Sahai generators, which define a decryption key $\mathsf{dk}_{\mathcal{BS}} = (x_1, x_2)$ composed of two scalars.

- $(\mathcal{U}_i, \mathcal{S}, \mathcal{T})$:

- $\mathsf{Blind}_{\mathcal{BS}}(M, \mathsf{pk}_{\mathcal{BS}}; (r_1, r_2, r_3))$ (where we omit the subscripts i): For a message $M \in \{0,1\}^k$ and random scalars in $\mathbb{Z}_p$, define the commitment $c = \mathcal{C}(\mathcal{F}(M)) = (c_1, c_2, c_3)$. We also add, as before, proofs of validity of this commitment:
  - A proof Π_M of knowledge of M in c, the encrypted $\mathcal{F}(M)$, which consists of a bit-by-bit commitment $C_M = (\mathcal{C}'(M_1), \ldots, \mathcal{C}'(M_k))$ and proofs that each committed value is a bit. A proof that c_3 is well-formed *i.e.* c is a double linear encryption of the message M committed in C_M.
  - A proof Π_r containing the commitments $C_r = (\mathcal{C}(Y^{r_1+r_2}), \mathcal{C}(Y^{r_3}))$ together with proofs asserting that they are well-formed.
- $\mathsf{Sign}_{\mathcal{BS}}(\mathsf{sk}_{\mathcal{BS}}, (c = (c_{1,i}, c_{2,i}, c_{3,i}), \Pi_i)_{1 \le i \le j}; s)$: To sign several commitments, first check if they are valid with respect to the proofs Π's, and after some randomizations of those commitments, compute the global commitment $C = (\prod c_{1,i}, \prod c_{2,i}, u_0 \prod c_{3,i})$ which is still verifiable thanks to the previous (randomized) proofs, and then output $C = (C_1, C_2, C_3)$ and $\sigma = (C_1^s, C_2^s, Z \cdot C_3^s; g^s)$. We want to emphasize that u_0 is primordial in C_3 for the unforgeability.
- $\mathsf{Verif}(\mathsf{pk}_{\mathcal{BS}}, (C = (C_1, C_2, C_3)), \sigma = (\sigma_1, \sigma_2, \sigma_3; \sigma_4))$: In order to check the validity of the signature, one checks whether the following equations are verified: $e(\sigma_1, g) \stackrel{?}{=} e(C_1, \sigma_4)$, $e(\sigma_2, g) \stackrel{?}{=} e(C_2, \sigma_4)$, and $e(\sigma_3, g) \stackrel{?}{=} e(h, \mathsf{pk}_{\mathcal{BS}}) \cdot e(C_3, \sigma_4)$
- $\mathsf{Unblind}_{\mathcal{BS}}(\mathsf{dk}_{\mathcal{BS}}, \mathsf{pk}_{\mathcal{BS}}, (c = (C_1, C_2, C_3), \Pi, \sigma))$: On a valid signature, knowing the decryption key (x_1, x_2), one can obtain $F = \mathcal{F}(M)$, and extract the message M from the bit-by-bit commitments. One can also extract the corresponding valid signature: $\sigma' = (\sigma'_1 = \sigma_3 / (\sigma_1^{1/x_1} \sigma_2^{1/x_2}), \sigma'_2 = \sigma_4)$, which is a valid Waters signature on the concatenation of the messages.
- $\mathsf{Random}_{\mathcal{S}}(\mathsf{pk}_{\mathcal{BS}}, M, \sigma'; s')$: The latter can eventually be rerandomized to get a signature: $\Sigma = (\sigma'_1 \cdot \mathcal{F}(M)^{s'}, \sigma'_2 \cdot g^{s'})$.

- $\mathsf{Verif}_{\mathcal{BS}}(\mathsf{pk}_{\mathcal{BS}}, M, \sigma = (\sigma_1, \sigma_2))$: In order to check the validity of the signature, one checks whether: $e(\sigma_1, g) \stackrel{?}{=} e(h, \mathsf{pk}_{\mathcal{BS}}) \cdot e(\mathcal{F}(M), \sigma_2)$.

Theorem 4.4.3 *This multi-source blind signature scheme for concatenation is blind and unforgeable under the* CDH *and* DLin *assumptions: no adversary can generate more message-signature pairs on distinct messages, than the number of interactions with the signer.*

It directly follows from the previous result, combining the different partial Waters hashes into a global one does not weaken the security as we are still using single exponents on the u_i elements. Groth-Sahai proofs are in the perfectly binding setting to guarantee that each user really outputs Waters hash of their message on their generators and so no strange collision may occur and alter the final message.

Addition

The previous scheme presents a way to combine multiple blind messages into one in order to sign it. However it requires a huge number of generators and the final unblinded signature gives a lot of information on the repartition of the original messages, since they are simply concatenated. We now want to improve the previous scheme to drastically reduce the public key size, and the information leaked about the individual messages when one would like a signature on some computation on these messages, such as the addition or the mean. Instead of signing the concatenation of the messages, we now allow the users to use the same generators, and thus the messages will add together instead of concatenating.

The resulting algorithm is the same as before except during the Setup phase where $\vec{u} = (u_0, \ldots, u_k) \stackrel{\$}{\leftarrow} \mathbb{G}^{k+1}$. We then proceed as before considering $\mathcal{F}(M_i) = \prod_\ell u_\ell^{m_{i,\ell}}$. The $\mathsf{Unblind}$ algorithm now returns a valid signature on the sum of the messages. The various Groth-Sahai proofs help to ensure that the messages given to the Waters hash function are of reasonable size.

With this construction, the exponents in the Waters hash function are no longer bits but belong to a larger alphabet (*e.g.* $\{0, \ldots, t\}$ if t users sign only bit strings). Following the work done in [HK08], we have shown in section 2.6.4, page 42 that over a non-binary alphabet the Waters function remains $(1, poly)$-programmable as long as the size of the alphabet remains a polynomial in the security parameter. This result readily implies the security of the multi-source blind signature scheme for addition:

Theorem 4.4.4 *This multi-source blind signature scheme for addition is blind and unforgeable under the* CDH *assumption as long the alphabet size and the number of sources are polynomial in the security parameter.*

4.4.3 Other Applications

Our blind signature schemes find applications in various practical settings.

E-voting. We said earlier, that the security of several e-voting protocols relies on the fact that each ballot is certified by an election authority. Since this authority should not learn the voter's choice, a blind signature scheme is useful. In order to achieve privacy of the ballot in an information-theoretic sense, it is necessary to use a signature scheme that achieves perfect blindness. Our scheme is the first to achieve this property in the standard model and under classical complexity assumptions. Moreover if the authority wants to format the ballot (for example to add the date of the election), our signature being partially-blind it can be achieved easily.

E-cash. As mentioned above, partially-blind signatures played an important role in many electronic commerce applications. In e-cash systems, for instance, the bank issuing coins must ensure that the message contains accurate information such as the face value of the e-cash without seeing it and moreover in order to prevent double-spending, the bank's database has to record all spent coins. Partially-blind signatures can cope with these problems, since the bank can explicitly include some information such as the expiration date and the face value in the coin. Thanks to our proposal, the coin issuing protocol can be done without prior agreement between the bank and the client.

Data aggregation in networks. A *wireless (ad hoc) sensor network* (WSN) consists of many sensor nodes that are deployed for sensing the environment and collecting data from it. Since transmitting and receiving data are the most energy consuming operations, data aggregation has been put forward as an essential paradigm in these networks. The idea is to combine the data coming from different sources – minimizing the number of transmissions and thus saving energy. In this setting, a WSN consists usually of three types of nodes:

- *sensor nodes* that are small devices equipped with one or more sensors, a processor and a radio transceiver for wireless communication.
- *aggregation nodes* (or aggregators) performing the data aggregation (*e.g.* average, sum, minimum or maximum of data).
- *base stations* responsible for querying the nodes and gathering the data collected by them.

WSNs are at high security risk and two important security goals when doing in-network data aggregation are *data confidentiality* and *data integrity*. When homomorphic encryption is used for data aggregation, end-to-end encryption allows aggregation of the encrypted data so that the aggregators do not need to decrypt and get access to the data and thus provides end-to-end data confidentiality. Achieving data integrity is a harder problem and usually we do not consider the attack where a sensor node reports a false reading value (the impact of such an attack being usually limited). The main security flaw is a *data pollution attack* in which an attacker tampers with the intermediate aggregation result at an aggregation node. The purpose of the attack is to make the base station receive the wrong aggregation result, and thus make the improper or wrong decisions.

While in most conventional data aggregation protocols, data integrity and privacy are not preserved at the same time, our multi-source blind signature primitive permits to achieve data confidentiality and to prevent data pollution attacks simultaneously by using the following simple protocol:

1. Data aggregation is initiated by a base station, which broadcasts a query to the whole network.
2. Upon receiving the query, sensor nodes report encrypted values of their readings (for the base station public key) to their aggregators
3. The aggregators check the validity of the received values, perform data aggregation via the homomorphic properties of the encryption scheme, (blindly) sign the result and route the aggregated results back to the base station.
4. The base station decrypts the aggregated data and the signature which proves the validity of the gathered information to the base station (but also to any other third party).

Conclusion

In this part, we have presented several results, coming primarily from [BFPV11, BP12].

First, we focused on some applications of Groth-Sahai methodology to provide enhanced version of Group Signatures, i.e. *Traceable Signatures*, and *List Signatures*, while remaining in the standard model, solving this way the open problem of List Signatures in the standard model.

We also presented, and instantiated under classical hypothesis, a new primitive called *Signature on Randomizable Ciphertext*. This primitive allowed us to build a round optimal blind signature scheme which outputs a short signature. It also lead us to diverse instantiations (perfectly blind signature, signer-friendly blind signature, multi-blind signature).

Those primitives are both done following the Groth-Sahai methodology, and based on the same tools. One can imagine a combination of both to create a Group Blind Signature [LR98] in the standard model, where we would have two sets of Groth-Sahai CRS, one for the *anonymity/opening* (So to commit to $\sigma_1, \sigma_2, \sigma_3$ in the traceable signature) and the other to blind the message/signature (To commit to σ_5). The final blinded signature would be $(\mathsf{ID}, \mathcal{C}_1(\sigma_1), \mathcal{C}_1(\sigma_2), \mathcal{C}_1(\sigma_3), \sigma_4, \mathcal{C}_2(\sigma_5), \sigma_6)$, where knowing the extraction key associated with $\mathcal{C}_2$ gives our traceable signature on the plaintext, whereas knowing the extraction key associated with $\mathcal{C}_1$ lets the opener break the anonymity.

However, these last primitives, while being the most efficient when they were introduced, can still be improved. This made us consider that while Groth-Sahai proofs are the most efficient NIWI/NIZK proofs in the standard model, one may consider another approach in interactive scenarios (like in *Blind Signatures*). This will be the starting point of our next chapter.

$\star$

Part II

Using Smooth Projective Hash Functions to Create Implicit Proofs of Knowledge

In the previous part, we presented various schemes relying on Groth-Sahai proof systems to show the knowledge of a secret. However, the second example, the *Blind Signature* is interactive, and while our protocol was more efficient than existing ones, we wondered why we should use a non-interactive proof system when there already is an interaction. What if, we could supplant this proof by an implicit one, more efficient, without increasing the number of rounds?

Starting from this idea we now focus on *Smooth Projective Hash Functions* [CS02]. See section 2.2.4, page 24 for a formal definition, and chapter 5, page 93 for the sets of languages we manage to handle.

We show in the following chapters that this approach is suitable for designing schemes that rely on standard security assumptions in the standard model with a common-reference string and are more efficient than those obtained using the Groth-Sahai methodology.

In Chapter 6, page 102, we focus on applications of our new methodology. We start with Oblivious Signature-Based Envelope [LDB03], and show that we can manage round optimality thanks to SPHF. Using the same design, we are then going to improve our blind signature scheme from the previous chapter by simply superseding Groth-Sahai proofs by a *Smooth Projective Hash Function* on an appropriate language, and so supplanting the NIZK by an implicit proof of knowledge.

We then proceed forward, and use SPHF as implicit proofs of knowledge in other interactive protocols. We present our *Language Authenticated Key Exchange* in Section 6.3, page 111, which is a generalization of *Password Authenticated Key-Exchange* where two users agree on a shared key if both possess a word in a language expected by the other. We are going to prove it in the UC framework with static corruptions under classical hypothesis, and present efficient PAKE, verifier-based PAKE, and Secret Handshakes instantiations. We also compare our scheme to the CAKE primitive which is closely related.

Chapter 5

Manageable Languages

Contents

In this chapter we cover the languages than can be handled by our methodology of implicit proofs on knowledge thanks to a Smooth Projective Hash Function. In section 2.2.4, page 24, we recalled how to combine two SPHF on different languages to create a new one on the conjunction or the disjunction of those languages. However the original set of manageable languages was not really developed, and so we are going to present several steps to extend it.

Intuitively our methodology aims to transform every language to the evaluation of a Linear Tuple / Diffie Hellman with respect to the public information. We will construct progressively more and more complicated SPHF on more and more vast languages.

In [ACP09], Abdalla *et al.* already formalized languages to be considered for SPHF. In the following, we will however use a more simple formalism, which is nevertheless more general: we consider any efficiently computable binary relation $\mathcal{R} : \{0,1\}^* \times \mathcal{P} \times \mathcal{S} \to \{0,1\}$, where the parameters $\mathsf{pub} \in \{0,1\}^*$ and $\mathsf{priv} \in \mathcal{P}$ define the language $\mathcal{L}_{\mathcal{R}}(\mathsf{pub}, \mathsf{priv}) \subseteq \mathcal{S}$ of the words W such that $\mathcal{R}(\mathsf{pub}, \mathsf{priv}, W) = 1$:

- pub are public parameters;
- priv are private parameters the two players have in mind, and they should think to the same values: they will be committed to, but never revealed;
- W is the word the sender claims to know in the language: it will be committed to, but never revealed.

In the whole chapter, we will consider the existence of a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, with a Linear Encryption scheme which will only be seen as a commitment scheme. We will do everything with a linear encryption scheme as a commitment, however this can be done with an ElGamal encryption if we do not need pairing, or are in an asymmetric setting. We can even use a CCA-2 variant, as explained in section 5.3, page 99.

In the following we are going to assume, that the following commitment scheme is defined in the CRS:

- Setup: Outputs a commitment key $\mathsf{ck} = (u, v, w) \in \mathbb{G}^3$.
- $\mathsf{Commit}(\mathsf{ck}, X; \alpha, \beta)$: For $X \in \mathbb{G}$ and $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_p^2$, defines $\mathbf{c} = \big(c_1 = u^\alpha, c_2 = v^\beta, c_3 = w^{\alpha+\beta} \cdot X\big)$.
- $\mathsf{Decommit}(\mathbf{c} = (c_1, c_2, c_3), X, \alpha, \beta)$: One checks that $\mathbf{c} = \iota(X) \odot (u^\alpha, v^\beta, w^{\alpha+\beta})$.

5.1 First Languages

In this section, we consider the language of Valid Signatures. This language is easily checkable so we can not directly expect pseudo-randomness, however if we consider the language composed of commitment of valid signatures, under the indistinguishability of the commitment this is a hard-partitioned subset.

In the following, we will assume there exists a Waters Signature scheme.

- $\mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$: Outputs $h \in \mathbb{G}$, and $(u_i)_{[\![0,k]\!]}$ for the Waters function $\mathcal{F}$.
- $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$: Picks a random $x \xleftarrow{\$} \mathbb{Z}_p$ and outputs $\mathsf{sk} = Y = h^x$, and $\mathsf{vk} = X = g^x$;
- $\mathsf{Sign}(\mathsf{sk}, m; \mu)$: outputs $\sigma(m) = (Y\mathcal{F}(m)^{\mu}, g^{-\mu})$;
- $\mathsf{Verif}(\mathsf{vk}, m, \sigma)$: checks if the following pairing equation holds: $e(g, \sigma_1) \cdot e(\mathcal{F}(m), \sigma_2) \stackrel{?}{=} e(X, h)$.

5.1.1 Commitment of a Valid Signature

We first start, with language $\mathcal{L}_{\mathsf{vk},m}$, of commitments under ck of a valid signature on m under vk. We will note it $\mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, m)$. Here, the private component of the language is empty $\emptyset$.

A given ciphertext $\mathbf{c} = (c_1, c_2, c_3, \sigma_2)$ contains a valid signature of m if and only if (c_1, c_2, c_3) actually encrypts σ_1 such that (σ_1, σ_2) is a valid Waters signature on m. If we note $\oslash$ the component-wise division, we can consider the ciphertext $\mathbf{C}$ defines from the latter as:

$$\begin{aligned}\mathbf{C} &= e(\mathbf{c}, g) \oslash \iota_T(e(\sigma_1, g)) \\ &= e(\mathbf{c}, g) \oslash \iota_T(e(h, \mathsf{vk}) \cdot e(\mathcal{F}(m), \sigma_2))) \\ &= (C_1 = e(c_1, g), C_2 = e(c_2, g), C_3 = e(c_3, g)/(e(h, \mathsf{vk}) \cdot e(\mathcal{F}(m), \sigma_2))\end{aligned}$$

$\mathbf{C}$ is a linear tuple in basis $(U = e(Y_1, g), V = e(Y_2, g), g_T = e(w, g))$ in $\mathbb{G}_T$. Since the basis consists of 3 elements of the form $e(\cdot, g)$, the projected key can be compacted in $\mathbb{G}$. We thus consider the language $\mathsf{WLin}(vk, m)$ that contains these quadruples $(c_1, c_2, c_3, \sigma_2)$, and proceeds as described below:

- Pre-Commit: The Prover does nothing (priv is empty, no helper values are required)
- Projection Keys: The Verifier computes and sends hp thanks to his random-tape ω_V.

$$\begin{aligned}\omega_V = \mathsf{HashKG}(\mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, m)) &= \mathsf{hk} = (x_1, x_2, x_3) \xleftarrow{\$} \mathbb{Z}_p^3 \\ \mathsf{ProjKG}(\mathsf{hk}; \mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, m), \mathbf{c}) &= \mathsf{hp} = (\mathsf{ck}_1^{x_1} g^{x_3}, \mathsf{ck}_2^{x_2} g^{x_3})\end{aligned}$$

- Commit: The Prover commits to his signature σ_1, σ_2 thanks to ck and so produces $(c_1, c_2, c_3, \sigma_2) = (\mathsf{ck}_1^{\alpha}, \mathsf{ck}_2^{\beta}, g^{\alpha+\beta} h^x \mathcal{F}(m)^s, g^s)$ for some scalars $\alpha, \beta, t \xleftarrow{\$} \mathbb{Z}_p$,
- Hashing: The Verifier computes his view of the Hash, and so does the Prover:

$$\begin{aligned}\mathsf{Hash}(\mathsf{hk}; \mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, m), \mathbf{c}) &= \\ &\quad e(c_1, g)^{x_1} e(c_2, g)^{x_2} (e(c_3, g)/(e(h, \mathsf{vk}) e(\mathcal{F}(m), \sigma_2)))^{x_3} \\ \mathsf{ProjHash}(\mathsf{hp}; \mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, m), \mathbf{c}; \alpha, \beta) &= e(\mathsf{hp}_1^{\alpha} \mathsf{hp}_2^{\beta}, g)\end{aligned}$$

We use the *public* information vk and σ_2 to remove the signature part of the commitment, and only check if $\mathbf{C}$ is a linear tuple in $\mathbb{G}_T$. If $\mathbf{c}$ is not in the language, then the decommit process does not lead to a valid signature on m, so $\mathbf{C}$ is not a linear tuple, and by the smoothness of the SPHF, the prover cannot guess the Hash value. Here follows the proof of such smoothness:

Proof: Let us show that from an information theoretic point of view, $v = \mathsf{Hash}(\mathsf{hk}, \mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, m), \mathbf{c})$ is unpredictable, even knowing hp, when $\mathbf{c}$ is not a correct ciphertext on a valid signature: $C = (u^{\alpha}, v^{\beta}, w^{\gamma}\sigma_1, \sigma_2)$, for $\gamma \neq \alpha+\beta$. (i.e. for all ciphertext, there exists a unique $\gamma \in \mathbb{Z}_p$ such that σ_1, σ_2 is indeed a valid signature on $m$under vk.) We recall that we have previously defined $\mathsf{Hash}(\mathsf{hk}; \mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, m), \mathbf{c})$ as $e(u, g)^{\alpha x_1} e(v, g)^{\beta x_2} e(w, g)^{\gamma x_3}$ and $\mathsf{hp} = (u^{x_1} w^{x_3}, u^{x_2} w^{x_3})$:

If we denote $u = w^{y_1}$ and $v = w^{y_2}$, and consider discrete logarithm in basis g_T we then have:

$$\begin{pmatrix} \log e(\mathsf{hp}_1, g) \\ \log e(\mathsf{hp}_2, g) \\ \log v \end{pmatrix} = \begin{pmatrix} y_1 & 0 & 1 \\ 0 & y_2 & 1 \\ y_1\alpha & y_2\beta & \gamma \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

The determinant of this matrix is $y_1y_2(\gamma-\alpha-\beta)$, which is non-zero if $\mathbf{C}$ (and so $\mathbf{c}$) does not belong to the language. So v is independent from hp and $\mathbf{c}$. □

Theorem 5.1.1 *This Smooth Projective Hash Function is pseudo-random under the* DLin *assumption (the semantic security of the Linear encryption).*

Proof: As shown above, when c encrypts $\sigma(m')$ for a $m' \neq m$, then the following distributions are perfectly indistinguishable:

$$\begin{aligned}\mathcal{D}_1 &= \{\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m), \mathbf{c}=\mathcal{C}_{\mathsf{ck}}(\sigma(m')), \mathsf{hp}, v \xleftarrow{\$} \mathbb{G}_T\}\\ \mathcal{D}_2 &= \{\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m), \mathbf{c}=\mathcal{C}_{\mathsf{ck}}(\sigma(m')), \mathsf{hp}, v=\mathsf{Hash}(\mathsf{hk},\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m),\mathbf{c})\}\end{aligned}$$

Under the semantic security of the Linear encryption, $\mathcal{C}_{\mathsf{ck}}(\sigma(m))$ and $\mathcal{C}_{\mathsf{ck}}(\sigma(m'))$ are computationally indistinguishable, and so are the distributions

$$\begin{aligned}\mathcal{D}_0 &= \{\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m), \mathbf{c}=\mathcal{C}_{\mathsf{ck}}(\sigma(m)), \mathsf{hp}, v \xleftarrow{\$} \mathbb{G}_T\}\\ \mathcal{D}_1 &= \{\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m), \mathbf{c}=\mathcal{C}_{\mathsf{ck}}(\sigma(m')), \mathsf{hp}, v \xleftarrow{\$} \mathbb{G}_T\}\end{aligned}$$

and the distributions

$$\begin{aligned}\mathcal{D}_2 &= \{\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m), \mathbf{c}=\mathcal{C}_{\mathsf{ck}}(\sigma(m')), \mathsf{hp}, v=\mathsf{Hash}(\mathsf{hk};\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m),\mathbf{c})\}\\ \mathcal{D}_3 &= \{\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m), \mathbf{c}=\mathcal{C}_{\mathsf{ck}}(\sigma(m)), \mathsf{hp}, v=\mathsf{Hash}(\mathsf{hk};\mathsf{WLin}(\mathsf{ck},\mathsf{vk},m),\mathbf{c})\}\end{aligned}$$

As a consequence, $\mathcal{D}_0$ and $\mathcal{D}_3$ are computationally indistinguishable, which proves the result. □

With this extension to the original set of languages, we are now able to instantiate Oblivious Signature-based Envelope thanks to the Smooth Projective Hash Function, as we will wee in section 6.1.3, page 109.

5.1.2 Extended Commitment of a Message

We are now going to continue to try to combine different languages. Now we can wonder what happens if instead of combining a Hard-Partitioned Subset language, with a verifiable one, we simply combine two together.

In our previous construction of Signature on Randomizable Ciphertexts in Section 4.2.2, page 76, we instantiated Blind Signatures, for that we had to commit to an extended message (the message, and some variation of the verification key). We had to combine two languages, one showing that we indeed committed the message bit per bit in $(d_{i,1}, d_{i,2}, d_{i,3})$ with randomness α_i, β_i, but this is simply a conjunction of disjunctions of a standard language, we will note it BLin, the other showing that is we note $d_1 = \prod d_{i,1} = \mathsf{ck}_1^{\sum \alpha_i} = \mathsf{ck}_1^{\alpha}, d_2 = \prod d_{i,2} = \mathsf{ck}_1^{\sum \beta_i} = \mathsf{ck}_1^{\beta}$ then, we do have in c_1, c_2, c_3, d_1, d_2 a commit of a linear tuple.

Bit Encryption Language

When we need to "prove" that a ciphertext encrypts a bit in exponent of a basis u_i, we consider the language $\mathsf{BLin}(\mathsf{ck}, u_i) = \mathsf{Lin}(\mathsf{ck}, 1_{\mathbb{G}}) \cup \mathsf{Lin}(\mathsf{ck}, u_i)$. This is thus a simple disjunction of two SPHF:

- $\mathsf{HashKG}(\mathsf{BLin}(\mathsf{ck}, u_i))$: $\mathsf{hk} = ((x_1,x_2,x_3),(y_1,y_2,y_3)) \xleftarrow{\$} \mathbb{Z}_p^6$
- $\mathsf{ProjKG}(\mathsf{hk}, \mathsf{BLin}(\mathsf{ck}, u_i), W)$: $\mathsf{hp} = ((u^{x_1}w^{x_3}, v^{x_2}w^{x_3}), (u^{y_1}w^{y_3}, v^{y_2}w^{y_3}), \mathsf{hp}_\Delta)$ where
$$\mathsf{hp}_\Delta = c_1^{x_1}c_2^{x_2}(c_3)^{x_3} \cdot c_1^{y_1}c_2^{y_2}(c_3/u_i)^{y_3}$$
- $\mathsf{Hash}(\mathsf{hk}, \mathsf{BLin}(\mathsf{ck}, u_i), W)$: $v = c_1^{x_1}c_2^{x_2}c_3^{x_3}$
- $\mathsf{ProjHash}(\mathsf{hp}, \mathsf{BLin}(\mathsf{ck}, u_i), W, w)$: If $W \in \mathcal{L}_1$, $v' = \mathsf{hp}_{1,1}^{\alpha_i} \cdot \mathsf{hp}_{1,2}^{\beta_i}$,
else (if $W \in \mathcal{L}_2$), $v' = \mathsf{hp}_\Delta / \mathsf{hp}_{2,1}^{\alpha_i} \cdot \mathsf{hp}_{2,2}^{\beta_i}$

The correctness, smoothness and pseudo-randomness properties of such function directly follow from those of the SPHF on $\mathsf{Lin}(\mathsf{pk}, 1_{\mathbb{G}})$ and $\mathsf{Lin}(\mathsf{pk}, u_i)$. Each final projection key is composed of 5 group elements.

Encryption of a Linear Tuple

So we need to consider a language $\mathsf{ELin}(\mathsf{ck},\mathsf{vk})$ of tuples (d_1,d_2,c_1,c_2,c_3), where (c_1,c_2,c_3) is a commit to d_3 under the public key $\mathsf{ck}=(u,v,w)$, such that (d_1,d_2,d_3) is a linear tuple in basis (u,v,vk). This can also be expressed as $c_3=\alpha\times d_3$, where d_3 is the plaintext in (c_1,c_2,c_3) under ck, which means that (c_1,c_2,α) is a linear tuple in basis (u,v,w), and (d_1,d_2,d_3) should be a linear tuple in basis (u,v,vk).

More concretely, we consider words $W=(d_1=u^\alpha,d_2=v^\beta,c_1=u^{s_1},c_2=v^{s_2},c_3=w^{s_1+s_2}\cdot\mathsf{vk}^{\alpha+\beta})$, with witness $\mathcal{W}=(\alpha,\beta,s_1,s_2)$. We have $d_3=\mathsf{vk}^{\alpha+\beta}$ and $c_3/d_3=w^{s_1+s_2}$, but they should remain secret, which requires a specific function, and not a simple conjunction of languages:

- $\mathsf{HashKG}(\mathsf{ELin}(\mathsf{ck},\mathsf{vk}))$: $\mathsf{hk}=(x_1,x_2,x_3,x_4,x_5)$
- $\mathsf{ProjKG}(\mathsf{hk},\mathsf{ELin}(\mathsf{ck},\mathsf{vk}),W)$: $\mathsf{hp}=(u^{x_1}g^{x_5},v^{x_2}g^{x_5},u^{x_3}w^{x_5},v^{x_4}w^{x_5})$
- $\mathsf{Hash}(\mathsf{hk},\mathsf{ELin}(\mathsf{ck},\mathsf{vk}),W)$: $v=e(d_1,\mathsf{vk})^{x_1}\cdot e(d_2,\mathsf{vk})^{x_2}\cdot e(c_1,g)^{x_3}\cdot e(c_2,g)^{x_4}\cdot e(c_3,g)^{x_5}$
- $\mathsf{ProjHash}(\mathsf{hp},\mathsf{ELin}(\mathsf{ck},\mathsf{vk}),W,\mathcal{W})$: $v'=e(\mathsf{hp}_1,\mathsf{vk})^{\alpha}\cdot e(\mathsf{hp}_2,\mathsf{vk})^{\beta}\cdot e(\mathsf{hp}_3,g)^{s_1}\cdot e(\mathsf{hp}_4,g)^{s_2}$

We now study the security of this SPHF:

Theorem 5.1.2 *This Smooth Projective Hash Function is correct.*

Proof: With the above notations:

$$\begin{aligned}
v &= e(d_1,\mathsf{vk})^{x_1}\cdot e(d_2,\mathsf{vk})^{x_2}\cdot e(c_1,g)^{x_3}\cdot e(c_2,g)^{x_4}\cdot e(c_3,g)^{x_5}\\
&= e(u^{x\alpha x_1},g)\cdot e(v^{x\beta x_2},g)\cdot e(u^{s_1x_3},g)\cdot e(v^{s_2x_4},g)\cdot e(g^{x(\alpha+\beta)x_5}w^{(s_1+s_2)x_5},g)\\
&= e(u^{x\alpha x_1+s_1x_3},g)\cdot e(v^{x\beta x_2+s_2x_4},g)\cdot e(g^{x(\alpha+\beta)x_5}w^{(s_1+s_2)x_5},g)\\
v' &= e(\mathsf{hp}_1,\mathsf{vk})^{\alpha}\cdot e(\mathsf{hp}_2,\mathsf{vk})^{\beta}\cdot e(\mathsf{hp}_3,g)^{s_1}\cdot e(\mathsf{hp}_4,g)^{s_2}\\
&= e(u^{x\alpha x_1}g^{x\alpha x_5},g)\cdot e(v^{x\beta x_2}g^{x\beta x_5},g)\cdot e(u^{s_1x_3}w^{s_1x_5},g)\cdot e(v^{s_2x_4}w^{s_2x_5},g)\\
&= e(u^{x\alpha x_1+s_1x_3},g)\cdot e(v^{x\beta x_2+s_2x_4},g)\cdot e(g^{x(\alpha+\beta)x_5}w^{(s_1+s_2)x_5},g)
\end{aligned}$$

□

Theorem 5.1.3 *This Smooth Projective Hash Function is smooth.*

Proof: Let us show that from an information theoretic point of view, v is unpredictable, even knowing hp, when W is not in the language: $W=(d_1=u^\alpha,d_2=v^\beta,c_1=u^{s_1},c_2=v^{s_2},c_3=w^t\cdot\mathsf{vk}^{\alpha+\beta})$, for $t\neq s_1+s_2$. We recall that

$$v=e(u^{x\alpha x_1+s_1x_3},g)\cdot e(v^{x\beta x_2+s_2x_4},g)\cdot e(g^{x(\alpha+\beta)x_5}w^{(s_1+s_2))x_5},g)=e(H,g)$$

for

$$H=u^{x\alpha x_1+s_1x_3}\cdot v^{x\beta x_2+s_2x_4}\cdot g^{x(\alpha+\beta)}w^{(s_1+s_2)x_5}$$

and

$$\mathsf{hp}=((u^{x_1}g^{x_5},v^{x_2}g^{x_5}),(u^{x_3}w^{x_5},v^{x_4}w^{x_5}))$$

If we denote $u=g^{y_1},v=g^{y_2}$ and $w=g^{y_3}$, we have:

$$\begin{pmatrix}\log\mathsf{hp}_1\\ \log\mathsf{hp}_2\\ \log\mathsf{hp}_3\\ \log\mathsf{hp}_4\\ \log H\end{pmatrix}=\begin{pmatrix}y_1 & 0 & 0 & 0 & 1\\ 0 & y_2 & 0 & 0 & 1\\ 0 & 0 & y_1 & 0 & y_3\\ 0 & 0 & 0 & y_2 & y_3\\ x\alpha y_1 & x\beta y_2 & s_1y_1 & s_2y_2 & y_3t+x(\alpha+\beta)\end{pmatrix}\cdot\begin{pmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{pmatrix}$$

The determinant of this matrix is $(y_1y_2)^2(y_3(t-(s_1+s_2))+(x(\alpha+\beta)-x(\alpha+\beta)))=(y_1y_2)^2y_3(t-(s_1+s_2))$, which is non-zero if W does not belong to the language ($t\neq s_1+s_2$). So v is independent from hp and W. □

Theorem 5.1.4 *This Smooth Projective Hash Function is pseudo-random under the* DLin *assumption (the semantic security of the Linear encryption).*

Proof: The fact that c_3 really encrypts d_3 that completes well (d_1,d_2) is hidden by the semantic security of the linear encryption, and so under the DLin assumption. So the proof works as above, on the Linear Language. □

Combination of both

The resulting language is then the conjunction of:

- Each $\mathsf{BLin}(\mathsf{ck}, u_i)$, that emphasizes that each each commitment is indeed to a bit in basis u_i
- $\mathsf{ELin}(\mathsf{ck}, \mathsf{vk})$, to show that the original commitment was indeed to a revisited Waters signature, with respect to the verification key vk.

With this we can now commit bit per bit to a message, and build a SPHF to check if we indeed committed bits (BLin). And then, as explained in the introduction, considering the component-wise product of all bit commitments, we can show it fits with the language described as (ELin).

Such approach can be continued to show more complex predicate. Like "I possess a signature on a message I do not want to give, under a verification key in a pre-agreed subset I do not want to give".

However, this all boils down to prove the validity of some (linear) pairing-equations. So we will show a generic construction allowing us to do so in the next session.

5.2 Linear Languages

For the Waters signature, we considered the equation $e(\underline{\sigma_1}, g) \stackrel{?}{=} e(h, \mathsf{vk}) \cdot e(\mathcal{F}(m), \sigma_2)$, where σ_1 only was encrypted, and all the other elements were public. We can consider more general language families. Their instantiations through our framework can be handled the same way, where the final hash is $H_{\mathcal{L}}$ for "the commitment belongs to the language $\mathcal{L}$". For the latter, one can consider more general equations, such as any linear pairing product equations: t equations of the form $\left(\prod_{i\in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i})\right) \cdot \left(\prod_{i\in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}}\right) = \mathcal{B}_k$, for $k = 1, \ldots, t$, where $\mathcal{A}_{k,i} \in \mathbb{G}$, $\mathcal{B}_k \in \mathbb{G}_T$, and $\mathfrak{z}_{k,i} \in \mathbb{Z}_p$, as well as $A_k \subseteq [\![1, m]\!]$ and $B_k \subseteq [\![m+1, n]\!]$ are public. This is more general than the relations covered by [CCGS10]. When compared to the Groth and Sahai methodology, we cannot handle (yet) quadratic equations (i.e. with a pairing between two committed variables), but we can sometimes work without pairings (when the language is not solely defined by a pairing), and also with committed variables in $\mathbb{G}_T$.

5.2.1 A Single Equation

Let us assume that we have an equation over secret variables $\mathcal{Y}_i$ to be committed in $\mathbb{G}$, in $\mathbf{c}_i$, for $i \in [\![1, m]\!]$ and $\mathcal{Z}_i$ to be committed in $\mathbb{G}_T$, in $\mathbf{C}_i$, for $i \in [\![m+1, n]\!]$, and we want to show they satisfy $\left(\prod_{i=1}^{m} e(\mathcal{Y}_i, \mathcal{A}_i)\right) \cdot \left(\prod_{i=m+1}^{n} \mathcal{Z}_i^{\mathfrak{z}_i}\right) = \mathcal{B}$, where $\mathcal{A}_i \in \mathbb{G}$, $\mathcal{B} \in \mathbb{G}_T$, and $\mathfrak{z}_i \in \mathbb{Z}_p$ are public.

If we note $\mathsf{ck} = (u, v, w)$ the commitment key in $\mathbb{G}$, the commitment key in $\mathbb{G}_T$ is simply $e(\mathsf{ck}, g) = (U = e(u, g), V = e(v, g), W = e(w, g))$.

We also define $(U_i = e(u, \mathcal{A}_i), V_i = e(v, \mathcal{A}_i), W_i = e(w, \mathcal{A}_i)), \mathcal{Z}_i = e(\mathcal{Y}_i, \mathcal{A}_i)$ for $i \in [\![1, m]\!]$, and $(U_i = U^{\mathfrak{z}_i}, V_i = V^{\mathfrak{z}_i}, W_i = W^{\mathfrak{z}_i}), \mathcal{A}_i = g^{\mathfrak{z}_i}$ for $i \in [\![m+1, n]\!]$.

For $i \in [\![1, m]\!]$, we can simply consider $\mathbf{C}_i$, the $\mathbf{c}_i$ in $\mathbb{G}_T$:

$$\begin{array}{rcll} \mathbf{C}_i & = & e(\mathbf{c}_i, \mathcal{A}_i) & \text{for } i \in [\![1, m]\!] \\ & = & (U_i^{r_i}, V_i^{s_i}, W_i^{r_i+s_i} e(\mathcal{Y}_i, \mathcal{A}_i)) & \text{for } i \in [\![1, m]\!] \\ & = & (U_i^{r_i}, V_i^{s_i}, W_i^{r_i+s_i} \mathcal{Z}_i) & \text{for } i \in [\![1, m]\!] \end{array}$$

And we already had:

$$\mathbf{C}_i = (U_i^{r_i}, V_i^{s_i}, W_i^{r_i+s_i} \mathcal{Z}_i) \quad \text{for } i \in [\![m+1, n]\!]$$

- Pre-Commit: With such linear encryption, and pairing equations we do not have private languages, nor helper values.
- Projected Hash: For the hashing keys, the Verifier picks scalars $(\lambda, (\eta_i, \theta_i)_{i=1,\ldots,n}) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^{2n+1}$, and sets $\mathsf{hk}_i = (\eta_i, \theta_i, \lambda)$. And then computes and sends the projection keys as $\forall i \in [\![1, n]\!], \mathsf{hp}_i = (u^{\eta_i} w^{\lambda}, v^{\theta_i} w^{\lambda}) \in \mathbb{G}^2$. The associated projection keys in $\mathbb{G}_T$ are $\mathsf{HP}_i = (e(\mathsf{hp}_{i,1}, \mathcal{A}_i), e(\mathsf{hp}_{i,2}, \mathcal{A}_i))$, for $i \in [\![1, n]\!]$.
- Commit: The Prover now sends all his $\mathbf{c}_i$ and for $i > m, \mathbf{C}_i$. (The previous $\mathbf{C}_i$ are computable thanks to the $\mathbf{c}_i$, and elements in $\mathbb{G}$ are far less expensive than those in $\mathbb{G}_T$.)

- Hash: The hash value is then

$$\begin{aligned} H &= \left(\prod_i C_{i,1}^{\eta_i} \cdot C_{i,2}^{\theta_i} \cdot C_{i,3}^{\lambda}\right) \times \mathcal{B}^{-\lambda} \\ &= \left(\prod_i U_i^{r_i\eta_i} \cdot V_i^{s_i\theta_i} \cdot W_i^{(r_i+s_i)\lambda} \mathcal{Z}_i^{\lambda}\right) \times \mathcal{B}^{-\lambda} \\ &= \left(\prod_i \mathsf{HP}_{i,1}^{r_i} \mathsf{HP}_{i,2}^{s_i} \mathcal{Z}_i^{\lambda}\right) \times \mathcal{B}^{-\lambda} \\ &= \prod_i \mathsf{HP}_{i,1}^{z_{r_i}} \mathsf{HP}_{i,2}^{z_{s_i}} = \left(\prod_{i=1}^{m} e(\mathsf{hp}_{i,1}^{r_i}, \mathcal{A}_i) \cdot e(\mathsf{hp}_{i,2}^{s_i}, \mathcal{A}_i)\right) \cdot \left(e(\prod_{i=m+1}^{n} \mathsf{hp}_{i,1}^{r_i}, g^{\mathfrak{z}_i}) \cdot e(\prod_{i=m+1}^{n} \mathsf{hp}_{i,2}^{s_i}, g^{\mathfrak{z}_i})\right) \end{aligned}$$

  which can be computed either from the commitments and the hashing keys, or from the projection keys and the witnesses.

We are first going to extend this protocol to several equations at once, and then prove the smoothness of such protocol, the pseudo-randomness being a direct consequence of the indistinguishability of the encryption scheme.

5.2.2 Multiple Equations

Let us assume that we have $\mathcal{Y}_i$ committed in $\mathbb{G}$, in $\mathbf{c}_i$, for $i \in [\![1, m]\!]$ and $\mathcal{Z}_i$ committed in $\mathbb{G}_T$, in $\mathbf{C}_i$, for $i \in [\![m+1, n]\!]$, and we want to show they simultaneously satisfy

$$\left(\prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i})\right) \cdot \left(\prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}}\right) = \mathcal{B}_k, \text{ for } k \in [\![1, t]\!].$$

where $\mathcal{A}_{k,i} \in \mathbb{G}$, $\mathcal{B}_k \in \mathbb{G}_T$, and $\mathfrak{z}_{k,i} \in \mathbb{Z}_p$, as well as $A_k \subseteq [\![1, m]\!]$ and $B_k \subseteq [\![m+1, n]\!]$ are public. As above,one can also derive the commitments in $\mathbb{G}_T$, $\mathbf{C}_{k,i}$ that correspond to the encryption of $\mathcal{Z}_{k,i} = e(\mathcal{Y}_i, \mathcal{A}_{k,i})$ under the keys $e(\mathsf{ck}, \mathcal{A}_{k,i})$.

- Pre-Commit: With such linear encryption, and pairing equations we do not have private languages, nor helper values.
- Projected Hash: For the hashing keys, the Verifier picks scalars $(\lambda, (\eta_i, \theta_i)_{i=1,\dots,n}) \xleftarrow{\$} \mathbb{Z}_p^{2\,n+1}$, and sets $\mathsf{hk}_i = (\eta_i, \theta_i, \lambda)$. And then computes and sends the projection keys as $\forall i \in [\![1, n]\!], \mathsf{hp}_i = (u^{\eta_i} w^{\lambda}, v^{\theta_i} w^{\lambda}) \in \mathbb{G}^2$. The associated projection keys in $\mathbb{G}_T$ are $\forall i \in [\![1, n]\!], \forall k \in 1, t, \mathsf{HP}_{k,i} = (e(\mathsf{hp}_{i,1}, \mathcal{A}_{k,i}), e(\mathsf{hp}_{i,2}, \mathcal{A}_{k,i}))$.
- Commit: The Prover now sends all his $\mathbf{c}_i$ and for $i > m, \mathbf{C}_i$. (The previous $\mathbf{C}_i$ are computable thanks to the $\mathbf{c}_i$, and elements in $\mathbb{G}$ are far less expensive than those in $\mathbb{G}_T$)
- Small-Exponents Test: The Verifier, now picks and sends $\{\varepsilon_k\}_{k \in [\![1,t]\!]} \xleftarrow{\$} \mathbb{Z}_p^t$. We insist on the fact that the ε_k's have to be sent after the commitments have been sent. (Or at least committed to, under a computationally binding commitment.)
- Hash: The hash value is then

$$\begin{aligned} H &= \prod_k \left(\left(\prod_{i \in A_k \cup B_k} C_{k,i,1}^{\eta_i} \cdot C_{k,i,2}^{\theta_i} \cdot C_{k,i,3}^{\lambda}\right) \times \mathcal{B}_k^{-\lambda}\right)^{\varepsilon_k} \\ &= \prod_k \left(\prod_{i \in A_k \cup B_k} \mathsf{HP}_{k,i,1}^{r_i} \cdot \mathsf{HP}_{k,i,2}^{s_i}\right)^{\varepsilon_k} \times \prod_k \left(\prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}} \times \mathcal{B}_k^{-1}\right)^{\lambda\varepsilon_k} \\ &= \prod_k \left(\prod_{i \in A_k \cup B_k} \mathsf{HP}_{k,i,1}^{r_i} \cdot \mathsf{HP}_{k,i,2}^{s_i}\right)^{\varepsilon_k} \times 1 \end{aligned}$$

  which can be computed either from the commitments and the hashing keys, or from the projection keys and the witnesses.

It should be noted that this protocol requires $3m$ elements in $\mathbb{G}$ for each variable in $\mathbb{G}$, $3n - m$ in $\mathbb{G}_T$ for each in $\mathbb{G}_T$, $2n$ in $\mathbb{G}$ for the projection keys, and t scalars for the small exponents.

5.2.3 Smoothness of a SPHF on Linear Pairing Product Equations

The list of commitments should be considered in the language if and only if:

- the commitments are all valid Linear ciphertexts (in either $\mathbb{G}$ or $\mathbb{G}_T$), which is easily checkable with a linear encryption
- the plaintexts satisfy the linear pairing product equations

Here is a proof of the smoothness of such SPHF:

Proof: Let us assume that the equation k is not verified, then it means that for some index $i \in [\![1, n]\!]$, the ciphertext $\mathbf{C}_{i,k}$ in $\mathbb{G}_T$ satisfies $\mathbf{C}_{i,k} = (U_{i,k}^{r_i}, V_{i,k}^{s_i}, W^{t_i}\mathcal{Z}_{k,i})$ with $t_i \neq r_i + s_i$. Then, the contribution of this ciphertext in the hash value is $(\mathbf{C}_{i,k,1}^{\eta_i} \cdot \mathbf{C}_{i,k,2}^{\theta_i} \cdot \mathbf{C}_{i,k,3}^{\lambda})^{\varepsilon'_i}$, where $\varepsilon'_i = \sum_{k,i\in A_k\cup B_k} \varepsilon_k$, knowing the projection keys that reveal, at most,

$$\log_u \mathsf{hp}_{i,1} = \eta_i + x_3 \times \lambda \quad \text{and} \log_u \mathsf{hp}_{i,2} = x_2 \times \theta_i + x_3 \times \lambda,$$

where $v = u^{x_2} \quad w = u^{x_3}$. This contribution is thus $(U_{i,k}^{r_i\eta_i + x_2 s_i \theta_i + x_3 t_i \lambda})^{\varepsilon'_i}$. But even if all the discrete logarithms were known, and also λ, one has to guess $M = r_i\eta_i + x_2 s_i \theta_i + x_3 t_i \lambda$, given $\eta_i + x_3\lambda$ and $x_2 \times \theta_i + x_3 \times \lambda$:

$$M^\top = \begin{pmatrix} 1 & 0 & x_3 \\ 0 & x_2 & x_3 \\ r_i & x_2 s_i & x_3 t_i \end{pmatrix} \cdot \begin{pmatrix} \eta_1 \\ \theta_i \\ \lambda \end{pmatrix}.$$

This matrix determinant is $x_2 x_3 (t_i - (r_i + s_i))$, that is non-zero as soon as $t_i \neq r_i + s_i$. In this case, there is no way to guess the correct value better than by chance: $1/p$. Hence the smoothness of this hash function when one equation is not verified.

About the equation validity, the c_i or C_i of the involved ciphertexts contain plaintexts $\mathcal{Y}_i$ or $\mathcal{Z}_i$, and contribute to the hash value: from the projection keys, the k-th equation contributes to

$$H_k = \left(\prod_{i\in A_k} \mathsf{HP}_{k,i,1}^{r_i} \cdot \mathsf{HP}_{k,i,2}^{s_i} \times \prod_{i\in B_k} \left(\mathsf{HP}_{i,1}^{r_i} \cdot \mathsf{HP}_{i,2}^{s_i}\right)^{\mathfrak{d}_{k,i}} \right)^{\varepsilon_k} \times \left(\prod_{i\in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i\in B_k} \mathcal{Z}_i^{\mathfrak{d}_{k,i}} \times \mathcal{B}_k^{-1} \right)^{\lambda\varepsilon_k}.$$

Let us denote $\alpha_k = \prod_{i\in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i\in B_k} \mathcal{Z}_i^{\mathfrak{d}_{k,i}} \times \mathcal{B}_k^{-1}$, then the uncertainty about H is $(\prod_k \alpha_k^{\varepsilon_k})^\lambda$. As soon as one of the equations is not satisfied, one of the α_k is different from 1. Since the ε_k's are unknown at the commitment time, one cannot make the α_k to compensate themselves, but by chance: if one equation is not satisfied, the probability that $\prod_k \alpha_k^{\varepsilon_k} = 1$ is $1/p$. Excepted this negligible case, $(\prod_k \alpha_k^{\varepsilon_k})^\lambda$ is totally unpredictable since λ is random. □

5.3 With Other Kinds of Commitments

In the previous sections, we have handled the cases where the commitment scheme used was a linear encryption, and what we have seen is that for all the possible languages we were able to create projection key verifying that some specific tuple was indeed a linear encryption of a precise plaintext. Now what happens when we use another kind of commitment?

In this section, we are first going to present briefly how to create SPHF when the commitment used in a Multi-Linear Cramer-Shoup (see Section 2.6.5, page 51), and then consider our commitment *à la* Lindell (see Section 2.6.6, page 52).

5.3.1 With a Multi-Linear Cramer-Shoup

In Section 2.6.5, page 51, we defined a Multi-linear Cramer-Shoup of a vector $\vec{M}$ as $\mathsf{Encrypt}(\ell, \mathsf{pk}, \vec{M}; \vec{r}, \vec{s})$, for a vector $\vec{M} \in \mathbb{G}^n$ and vectors $\vec{r}, \vec{s} \in \mathbb{Z}_p^n$, computes

$$\mathcal{C} \quad = \quad (C_1, \ldots, C_n), \text{ where } C_i = (\mathbf{u}_i = (g_1^{r_i}, g_2^{s_i}, g_3^{r_i+s_i}), e_i = M_i \cdot h_1^{r_i} h_2^{s_i}, v_i = (c_1 d_1^\xi)^{r_i} (c_2 d_2^\xi)^{s_i})$$

where the v_i's are computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}_1, \ldots, \mathbf{u}_n, e_1, \ldots, e_n)$.

The committer may decommit such vector $\mathcal{C}$ by simply sending the randomness he used during the commitment. So like with the*regular* linear encryption we build a SPHF on this commitment.

The verifier will compute a Hash key hk by picking blocks of five random scalars $\eta_i, \theta_i, \kappa_i, \lambda_i, \mu_i \xleftarrow{\$} \mathbb{Z}_p^{5n}$, and produce blocks of two projection keys $\mathsf{hp}_i = (g_1^{\eta_i} g_3^{\kappa_i} h_1^{\lambda_i} (c_1 d_1^{\xi})^{\mu_i}, g_2^{\theta_i} g_3^{\kappa_i} h_2^{\lambda_i} (c_2 d_2^{\xi})^{\mu_i})$.

The initial committer will now be able to compute a hash value $H = \prod_i \mathsf{hp}_{i,1}^{r_i} \mathsf{hp}_{i,2}^{s_i}$ and sends it to the verifier together with $\vec{M}$.

Now the verifier can check that $\mathcal{C}$ was indeed a valid commitment to $\vec{M}$, by computing the product of each H_i, where $H_i = \prod_i C_{i,1}^{\eta_i} C_{i,2}^{\theta_i} C_{i,3}^{\kappa_i} (C_{i,4}/M_i)^{\lambda_i} C_{i,5}^{\mu_i}$.

Such SPHF simultaneously checks that $\mathcal{C}$ was well-formed, and that the associated plaintext was M.

Theorem 5.3.1 *Under the* DLin *assumption, the above smooth projective hash function is both smooth and pseudo-random:*

- *Smoothness:* $\mathsf{Adv}_{\Pi}^{\mathsf{smooth}} = 0$*;*
- *Pseudo-Randomness:* $\mathsf{Adv}_{\Pi}^{\mathsf{pr}}(t) \leq \mathsf{Adv}_{p,\mathbb{G},g}^{\mathsf{dlin}}(t)$.

Proof: For the correctness, one can easily check that if $\mathcal{C}$ contains $M' = M$, then $H = H'$:

$$\begin{aligned} H &= u_1^{\eta} u_2^{\theta} u_3^{\kappa} (e/W)^{\lambda} v^{\mu} = (g_1^{z_r})^{\eta} (g_2^{z_s})^{\theta} (g_3^{z_r+z_s})^{\kappa} (h_1^{z_r} h_2^{z_s} W'/W)^{\lambda} (v_1^{z_r} v_2^{z_s})^{\mu} \\ &= (g_1^{\eta} g_3^{\kappa} h_1^{\lambda} v_1^{\mu})^{z_r} \times (g_2^{\theta} g_3^{\kappa} h_2^{\lambda} v_2^{\mu})^{z_s} \times (W'/W)^{\lambda} = \mathsf{hp}_1^{z_r} \mathsf{hp}_2^{z_s} \times (W'/W)^{\lambda} = H' \times (W/W')^{\lambda}. \end{aligned}$$

Smoothness: if $\mathcal{C}$ is not a correct encryption of W, then H is unpredictable: let us denote W' and z'_s such that $\mathcal{C} = (\vec{u} = (g_1^{z_r}, g_2^{z_s}, g_3^{z_t}), e = W' h_1^{z_r} h_2^{z_s}, v = v_1^{z_r} v_2^{z'_s})$. Then, if we denote $g_2 = g_1^{\beta_2}$ and $g_3 = g_1^{\beta_3}$, and $h_1 = g_1^{\rho_1}$ and $h_2 = g_1^{\rho_2}$, but also $v_1 = g_1^{\delta_1}$ and $v_2 = g_1^{\delta_2}$, and $\Delta = \log_{g_1}(W'/W)$:

$$\begin{aligned} H &= g_1^{\eta z_r} g_1^{\beta_2 \theta z_s} g_1^{\beta_3 \kappa z_t} (W'/W)^{\lambda} (g_1^{\rho_1 z_r + \rho_2 z_s})^{\lambda} (v_1^{z_r} v_2^{z'_s})^{\mu}, \\ \log_{g_1} H &= \eta z_r + \beta_2 \theta z_s + \beta_3 \kappa z_t + \lambda(\rho_1 z_r + \rho_2 z_s) + \mu(\delta_1 z_+ \delta_2 z'_s) + \lambda\Delta. \end{aligned}$$

The information leaked by the projected key is $\log_{g_1} \mathsf{hp}_1 = \eta + \beta_3\kappa + \rho_1\lambda + \delta_1\mu$ and $\log_{g_1} \mathsf{hp}_2 = \beta_2\theta + \beta_3\kappa + \rho_2\lambda + \delta_2\mu$, which leads to the matrix

$$\begin{pmatrix} 1 & 0 & \beta_3 & \rho_1 & \delta_1 \\ 0 & \beta_2 & \beta_3 & \rho_2 & \delta_2 \\ z_r & \beta_2 z_s & \beta_3 z_t & \Delta + \rho_1 z_r + \rho_2 z_s & \delta_1 z_r + \delta_2 z'_s \end{pmatrix}$$

One remarks that if $z_t \neq z_r + z_s \bmod p$, then the three rows are not linearly dependent even considering the 3 first components only, and then H is unpredictable. Hence, we can assume that $z_t = z_r + z_s \bmod p$. The third row must thus be the first multiplied by z_r plus the second multiplied by z_s: $\rho_2 z_s = \Delta + \rho_2 z_s \bmod p$ and $z_s = z'_s \bmod p$, which implies $z'_s = s$ and $\Delta = 0$, otherwise, H remains unpredictable.

As a consequence, if $\mathcal{C}$ is not a correct encryption of W, H is perfectly unpredictable in $\mathbb{G}$:

$$\begin{aligned} &\{(\mathsf{hp}, H), \mathsf{hk} = (\eta, \theta, \kappa, \lambda, \mu) \xleftarrow{\$} \mathbb{Z}_p^5, \mathsf{hp} = (\mathsf{hp}_1 = g_1^{\eta} g_3^{\kappa} h_1^{\lambda} v_1^{\mu}, \mathsf{hp}_2 = g_2^{\theta} g_3^{\kappa} h_2^{\lambda} v_2^{\mu}), H \leftarrow \mathsf{Hash}(\mathsf{hk}, W, \mathcal{C})\} \\ &\approx_s \{(\mathsf{hp}, H), \mathsf{hk} = (\eta, \theta, \kappa, \lambda, \mu) \xleftarrow{\$} \mathbb{Z}_p^5, \mathsf{hp} = (\mathsf{hp}_1 = g_1^{\eta} g_3^{\kappa} h_1^{\lambda} v_1^{\mu}, \mathsf{hp}_2 = g_2^{\theta} g_3^{\kappa} h_2^{\lambda} v_2^{\mu}), H \xleftarrow{\$} \mathbb{G}\}. \end{aligned}$$

Pseudo-Randomness: we have just shown that if $\mathcal{C}$ is not a correct encryption of W, then H is statistically unpredictable. Let us be given a triple (g_1, g_2, g_3) together with another triple $\vec{u} = (u_1 = g_1^a, u_2 = g_2^b, u_3 = g_3^c)$. We choose random exponents $(x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3)$, and for $i = 1, 2$, we set $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. We generate $\mathcal{C} = (\vec{u}, e = W \times u_1^{z_1} u_2^{z_2} u_3^{z_3}, v = u_1^{x_1+\xi y_1} u_2^{x_2+\xi y_2} u_3^{x_3+\xi y_3})$. If $c = a + b \bmod p$ (i.e. , $\vec{u}$ is a linear tuple in basis $\vec{g}$), then $\mathcal{C}$ is a valid encryption of W, otherwise this is not, and we can apply the smoothness property:

$$\mathsf{Adv}_{\Pi}^{\mathsf{pr}}(t) \leq \mathsf{Adv}_{\Pi}^{\mathsf{smooth}} + \mathsf{Adv}_{p,\mathbb{G},g}^{\mathsf{dlin}}(t) \leq \mathsf{Adv}_{p,\mathbb{G},g}^{\mathsf{dlin}}(t).$$

□

5.3.2 With our Equivocable Commitment *à la* Lindell

In Section 2.6.6, page 52, we upgraded the previous commitment to allow some equivocability, and we can also compute a Smooth Projective Hash Function to again supersede the Decommit process.

To allow an implicit verification with SPHF, one omits to send $\vec{m}$ and $\mathbf{z}$, but makes an implicit proof of their existence. Therefore, $\vec{m}$ cannot be committed/verified in $\mathcal{C}''$, which has an impact on the binding property: $\mathcal{C}$ and $\mathcal{C}''$ are not binded to a specific $\vec{m}$, even in a computational way.

However, as said for the standard decommit, if $\mathcal{C}''$ contains a ciphertext $\mathcal{C}'$ of $\vec{N} \neq (1_{\mathbb{G}})^n$, the actual committed value will depend on $\vec{\varepsilon}$: $\vec{M}' = \vec{M}\vec{N}^{\vec{\varepsilon}}$ has its i-component, where $N_i \neq 1_{\mathbb{G}}$, uniformly distributed in $\mathbb{G}$ when ε is uniformly distributed in $\mathbb{Z}_p^*$. In addition, if $\vec{\varepsilon} \xleftarrow{\$} (\mathbb{Z}_p^*)^n$, all these i-component where $N_i \neq 1_{\mathbb{G}}$ are randomly and independently distributed in $\mathbb{G}$. Then, if the committed value has to satisfy a specific relation, with very few solutions, $\vec{m}'$ such that $\vec{M}' = \mathcal{G}(\vec{m}')$ will unlikely satisfy it.

Chapter 6

Applications

Contents

In the previous chapter, we presented an overview of easily manageable languages with our implicit proofs framework. We are now going to show how to apply this approach to different protocols and show how to increase their existing security while improving their efficiency without breaking the pre-existing round-optimality.

First we are going to instantiate Oblivious Signature-Based Envelope while providing some extra protection to the user without breaking the round-optimality.

Then, we are going to re-instantiate our blind signatures swapping the Groth-Sahai proofs by a Smooth Projective Hash Function on an extended language. This emphasizes the contribution of our result, because with no other alteration we manage to improve the efficiency.

We are then going to focus on advanced protocols revolving around Authenticated Key Exchange, and extend the notion of PAKE to LAKE. This notion supersedes the existing approach for PAKE: to establish a common shared key the users do not require a shared password anymore but instead they need to possess a word and a witness that this word belongs to the language expected by the other user.

6.1 Oblivious Signature-Based Envelope

Oblivious Signature-Based Envelope (OSBE) were introduced in [LDB03]. It can be viewed as an efficient way to ease the asymmetrical aspect of several authentication protocols. Alice is a member of an organization and possesses a certificate produced by an authority attesting she is in this organization. Bob wants to send a private message P to members of this organization. However due to the sensitive nature of the organization, Alice does not want to give Bob neither her certificate nor a proof she belongs to the organization. OSBE lets Bob sends an obfuscated version of this message P to Alice, in such a way that Alice will be able to find P if and only if Alice is in the required organization. In the process,

Bob cannot decide whether Alice does really belong to the organization. They are part of a growing field of protocols, around *automated trust negotiation*, which also include Secret Handshakes [BDS+03], Password-based Authenticated Key-Exchange [BM93, BPR00], and Hidden Credentials [BHS04]. Those schemes are all closely related, [CJT04] showed that if you tweak two OSBE, you can produce any of the other AKE.

6.1.1 Definition and Security Properties

In this section, we are going to precise the security notions of an OSBE scheme. We strengthen the usual security requirements, in order to prevent some interference by the authority, where before it was not considered a liability when the authority can find the message sent to the user.

We now define an OSBE protocol, where a sender $\mathcal{S}$ wants to send a private message $P \in \{0,1\}^\ell$ to a recipient $\mathcal{R}$ in possession of a certificate/signature on a message M.

Oblivious Signature-Based Envelope
⌜ An OSBE scheme is defined by four algorithms $(\mathsf{OSBESetup}, \mathsf{OSBEKeyGen}, \mathsf{OSBESign}, \mathsf{OSBEVerif})$, and one interactive protocol $\mathsf{OSBEProtocol}\langle\mathcal{S},\mathcal{R}\rangle$:

- $\mathsf{OSBESetup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param;
- $\mathsf{OSBEKeyGen}(\mathsf{param})$ generates the keys $(\mathsf{vk}, \mathsf{sk})$ of the certification authority;
- $\mathsf{OSBESign}(\mathsf{sk}, m)$ produces a signature σ on the input message m, under the signing key sk;
- $\mathsf{OSBEVerif}(\mathsf{vk}, m, \sigma)$ checks whether σ is a valid signature on m, w.r.t. the public key vk; it outputs 1 if the signature is valid, and 0 otherwise.
- $\mathsf{OSBEProtocol}\langle\mathcal{S}(\mathsf{vk}, M, P), \mathcal{R}(\mathsf{vk}, M, \sigma)\rangle$ between the sender $\mathcal{S}$ with the private message P, and the recipient $\mathcal{R}$ with a certificate σ. If σ is a valid signature under vk on the common message M, then $\mathcal{R}$ receives P, otherwise it receives nothing. In any case, $\mathcal{S}$ does not learn anything.

⌟

Such an OSBE scheme should be (the three last properties are additional —or stronger— security properties from the original definitions [LDB03]):

- *correct*: the protocol actually allows $\mathcal{R}$ to learn P, whenever σ is a valid signature on M under vk;
- *oblivious*: the sender should not be able to distinguish whether $\mathcal{R}$ uses a valid signature σ on M under vk as input. More precisely, if $\mathcal{R}_0$ knows and uses a valid signature σ and $\mathcal{R}_1$ does not use such a valid signature, the sender cannot distinguish an interaction with $\mathcal{R}_0$ from an interaction with $\mathcal{R}_1$;
- *(weakly) semantically secure*: the recipient learns nothing about $\mathcal{S}$ input P if it does not use a valid signature σ on M under vk as input. More precisely, if $\mathcal{S}_0$ owns P_0 and $\mathcal{S}_1$ owns P_1, the recipient that does not use a valid signature cannot distinguish an interaction with $\mathcal{S}_0$ from an interaction with $\mathcal{S}_1$;
- *semantically secure* (denoted sem): the above indistinguishability should hold even if the receiver has seen several interactions $\langle\mathcal{S}(\mathsf{vk}, M, P), \mathcal{R}(\mathsf{vk}, M, \sigma)\rangle$ with valid signatures, and the same sender's input P;
- *oblivious with respect to the authority* (denoted obl_A): the authority (owner of the signing key sk), playing as the sender or just eavesdropping, is unable to distinguish whether $\mathcal{R}$ used a valid signature σ on M under vk as input. This notion supersedes the above *oblivious* property, since this is basically oblivious w.r.t. the authority, without any restriction.
- *semantically secure w.r.t. the authority* (denoted sem^*): after the interaction, the authority (owner of the signing key sk) learns nothing about P.

We insist that the oblivious with respect to the authority property (obl_A) is stronger than the oblivious property, hence we will consider the former only. However, the semantic security w.r.t. the authority (sem^*) is independent from the basic semantic security (sem) since in the latter the adversary interacts

$\mathsf{Exp}^{\mathsf{obl}\mathcal{A}-b}_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K})$ [Oblivious w.r.t. the authority]
1. $\mathsf{param} \leftarrow \mathsf{OSBESetup}(1^{\mathfrak{K}})$
2. $\mathsf{vk} \leftarrow \mathcal{A}(\texttt{INIT} : \mathsf{param})$
3. $(M, \sigma) \leftarrow \mathcal{A}(\texttt{FIND} : \texttt{Send}(\mathsf{vk}, \cdot, \cdot), \texttt{Rec}^*(\mathsf{vk}, \cdot, \cdot, 0), \texttt{Exec}^*(\mathsf{vk}, \cdot, \cdot, \cdot))$
4. $\mathsf{OSBEProtocol}\langle \mathcal{A}, \texttt{Rec}^*(\mathsf{vk}, M, \sigma, b)\rangle$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \texttt{Send}(\mathsf{vk}, \cdot, \cdot), \texttt{Rec}^*(\mathsf{vk}, \cdot, \cdot, 0), \texttt{Exec}^*(\mathsf{vk}, \cdot, \cdot, \cdot))$
6. RETURN b'

$\mathsf{Exp}^{\mathsf{sem}^*-b}_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K})$ [Semantic security w.r.t. the authority]
1. $\mathsf{param} \leftarrow \mathsf{OSBESetup}(1^{\mathfrak{K}})$
2. $\mathsf{vk} \leftarrow \mathcal{A}(\texttt{INIT} : \mathsf{param})$
3. $(M, \sigma, P_0, P_1) \leftarrow \mathcal{A}(\texttt{FIND} : \texttt{Send}(\mathsf{vk}, \cdot, \cdot), \texttt{Rec}^*(\mathsf{vk}, \cdot, \cdot, 0), \texttt{Exec}^*(\mathsf{vk}, \cdot, \cdot, \cdot))$
4. $\mathsf{transcript} \leftarrow \mathsf{OSBEProtocol}\langle \texttt{Send}(\mathsf{vk}, M, P_b), \texttt{Rec}^*(\mathsf{vk}, M, \sigma, 0)\rangle$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathsf{transcript}, \texttt{Send}(\mathsf{vk}, \cdot, \cdot), \texttt{Rec}^*(\mathsf{vk}, \cdot, \cdot, 0), \texttt{Exec}^*(\mathsf{vk}, \cdot, \cdot, \cdot))$
6. RETURN b'

$\mathsf{Exp}^{\mathsf{sem}-b}_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K})$ [Semantic Security]
1. $\mathsf{param} \leftarrow \mathsf{OSBESetup}(1^{\mathfrak{K}})$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{OSBEKeyGen}(\mathsf{param})$
3. $(M, P_0, P_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{vk}, \texttt{Sign}^*(\mathsf{vk}, \cdot), \texttt{Send}(\mathsf{vk}, \cdot, \cdot), \texttt{Rec}(\mathsf{vk}, \cdot, 0), \texttt{Exec}(\mathsf{vk}, \cdot, \cdot))$
4. $\mathsf{OSBEProtocol}\langle \texttt{Send}(\mathsf{vk}, M, P_b), \mathcal{A}\rangle$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \texttt{Sign}(\mathsf{vk}, \cdot), \texttt{Send}(\mathsf{vk}, \cdot, \cdot), \texttt{Rec}(\mathsf{vk}, \cdot, 0), \texttt{Exec}(\mathsf{vk}, \cdot, \cdot))$
6. IF $M \in \mathcal{SM}$ RETURN 0 ELSE RETURN b'

Figure 6.1: Security Games for $\mathcal{OSBE}$

with the sender whereas in the former the adversary (who generated the signing keys) has only passive access to a challenge transcript.

These security notions can be formalized by the security games presented on Figure 6.1, page 104, where the adversary keeps some internal state between the various calls INIT, FIND and GUESS. They make use of the oracles described below, and the advantages of the adversary are, for all the security notions,

$$\mathsf{Adv}^*_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K}) = \Pr[\mathsf{Exp}^{*-1}_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K}) = 1] - \Pr[\mathsf{Exp}^{*-0}_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K}) = 1]$$

$$\mathsf{Adv}^*_{\mathcal{OSBE}}(\mathfrak{K}, t) = \max_{\mathcal{A} \leq t} \mathsf{Adv}^*_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K}).$$

- $\texttt{Sign}(\mathsf{vk}, m)$: This oracle outputs a valid signature on m under the signing key sk associated to vk (where the pair $(\mathsf{vk}, \mathsf{sk})$ has been outputted by the $\mathsf{OSBEKeyGen}$ algorithm);
- $\texttt{Sign}^*(\mathsf{vk}, m)$: This oracle first queries $\texttt{Sign}(\mathsf{vk}, m)$. It additionally stores the query m to the list $\mathcal{SM}$;
- $\texttt{Send}(\mathsf{vk}, m, P)$: This oracle emulates the sender with private input P, and thus may consist of multiple interactions;
- $\texttt{Rec}(\mathsf{vk}, m, b)$: This oracle emulates the recipient either with a valid signature σ on m under the verification key vk (obtained from the signing oracle $\texttt{Sign}$) if $b = 0$ (as the above $\mathcal{R}_0$), or with a random string if $b = 1$ (as the above $\mathcal{R}_1$). This oracle is available when the signing key has been generated by $\mathsf{OSBEKeyGen}$ only;
- $\texttt{Rec}^*(\mathsf{vk}, m, \sigma, b)$: This oracle does as above, with a valid signature σ provided by the adversary. If $b = 0$, it emulates the recipient playing with σ; if $b = 1$, it emulates the recipient playing with a random string;
- $\texttt{Exec}(\mathsf{vk}, m, P)$: This oracle outputs the transcript of an honest execution between a sender with private input P and the recipient with a valid signature σ on m under the verification key vk (obtained from the signing oracle $\texttt{Sign}$). It basically activates the $\texttt{Send}(\mathsf{vk}, m, P)$ and $\texttt{Rec}(\mathsf{vk}, m, 0)$ oracles.
- $\texttt{Exec}^*(\mathsf{vk}, m, \sigma, P)$: This oracle outputs the transcript of an honest execution between a sender with private input P and the recipient with a valid signature σ (provided by the adversary). It basically activates the $\texttt{Send}(\mathsf{vk}, m, P)$ and $\texttt{Rec}^*(\mathsf{vk}, m, \sigma, 0)$ oracles.

Remark The OSBE schemes proposed in [LDB03] do not satisfy the semantic security w.r.t. the authority. This is obvious for the generic construction based on identity-based encryption which consists in only one flow of communication (since a scheme that achieves the strong security notions requires at

least two flows). This is also true (to a lesser extent) for the RSA-based construction: for any third party, the semantic security relies (in the random oracle model) on the CDH assumption in a 2048-bit RSA group; but for the authority, it can be broken by solving two 1024-bit discrete logarithm problems. This task is much simpler in particular if the authority generates the RSA modulus $N = pq$ dishonestly (*e.g.* with $p-1$ and $q-1$ smooth). In order to make the scheme secure in our strong model, one needs (at least) to double the size of the RSA modulus and to make sure that the authority has selected and correctly employed a truly random seed in the generation of the RSA key pair [JG02].

6.1.2 High Level Instantiation

In this section, we present a high-level instantiation of OSBE with our usual primitives as black boxes.

We assume we have an extractable commitment scheme $\mathcal{C}$, a signature scheme $\mathcal{S}$ and a SPHF system onto a set $\mathbb{G}$. We additionally use a key derivation function KDF to derive a pseudo-random bit-string $K \in \{0,1\}^\ell$ from a pseudo-random element v in $\mathbb{G}$. One can use the Leftover-Hash Lemma [HILL99], with a random seed defined in param during the global setup, to extract the entropy from v, then followed by a pseudo-random generator to get a long enough bit-string. Many uses of the same seed in the Leftover-Hash-Lemma just leads to a security loss linear in the number of extractions. We describe an oblivious signature-based envelope system $OSBE$, to send a private message $P \in \{0,1\}^\ell$:

- $\mathsf{OSBESetup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter:
  - it first generates the global parameters for the signature scheme, the commitment scheme , and the SPHF system (using their respective Setup);
  - it then generates the public commitment key ck (while the extraction key will not be used);

  The output param consists of all the individual param and the commitment key ck;
- $\mathsf{OSBEKeyGen}(\mathsf{param})$ runs $\mathsf{KeyGen}_{\mathcal{S}}(\mathsf{param})$ to generate a pair $(\mathsf{vk}, \mathsf{sk})$ of verification-signing keys;
- The OSBESign and OSBEVerif algorithms are exactly Sign and Verif from the signature scheme;
- $\mathsf{OSBEProtocol}\langle\mathcal{S}(\mathsf{vk}, M, P), \mathcal{R}(\mathsf{vk}, M, \sigma)\rangle$: In the following, $\mathcal{L} = \mathcal{L}(\mathsf{vk}, M)$ will describe the language of the ciphertexts under the above commitment key ck of a valid signature of the input message M under the input verification key vk (hence vk and M as inputs, while param contains ck).
  - $\mathcal{R}$ generates and sends $c = \mathsf{Commit}(\mathsf{ck}, \sigma; r)$;
  - $\mathcal{S}$ computes $\mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param})$, $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, $v = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, and $Q = P \oplus \mathsf{KDF}(v)$; $\mathcal{S}$ sends hp, Q to $\mathcal{R}$;
  - $\mathcal{R}$ computes $v' = \mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), c, r)$ and $P' = Q \oplus \mathsf{KDF}(v')$.

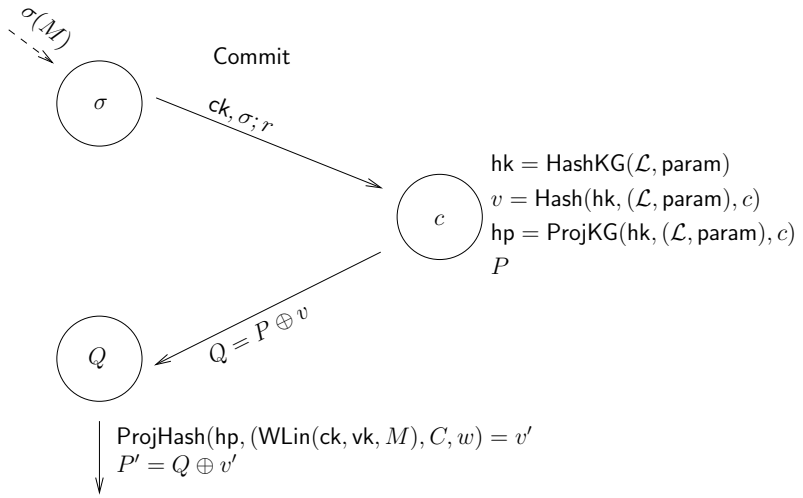


Figure 6.2: A One-Round OSBE based on Smooth Projective Hash Function

Security Properties

Theorem 6.1.1 (Correct) *$\mathcal{OSBE}$ is **sound**.*

Proof: Under the correctness of the SPHF system, $v' = v$, and thus $P' = (P \oplus \mathsf{KDF}(v)) \oplus \mathsf{KDF}(v') = P$. □

Theorem 6.1.2 (Oblivious w.r.t. the Authority) *$OSBE$ is **oblivious w.r.t. the authority** if the commitment scheme $\mathcal{C}$ is computationally hiding (*semantically secure*):*

$$\mathsf{Adv}^{\mathsf{obl}_A}_{\mathcal{OSBE}}(\mathfrak{K}, t) \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{C}}(\mathfrak{K}, t') \text{ with } t' \approx t.$$

Proof: Let us assume $\mathcal{A}$ is an adversary against the oblivious w.r.t. the authority property of our scheme: The malicious adversary $\mathcal{A}$ is able to tell the difference between an interaction with $\mathcal{R}_0$ (who knows and uses a valid signature) and $\mathcal{R}_1$ (who does not use a valid signature), with advantage ε.

We now build an adversary $\mathcal{B}$ against the semantic security of the commitment scheme $\mathcal{C}$:

- $\mathcal{B}$ is first given the parameters for $\mathcal{C}$ and a commitment key ck;
- $\mathcal{B}$ emulates $\mathsf{OSBESetup}$: it runs SSetup and $\mathsf{SPHFSetup}$ by itself. For the commitment scheme $\mathcal{C}$, the parameters and the key have already been provided by the challenger of the commitment security game;
- $\mathcal{A}$ provides the verification key vk;
- $\mathcal{B}$ has to simulate all the oracles:
  - $\mathtt{Send}(\mathsf{vk}, M, P)$, for a message M and a private input P: upon receiving c, one computes $\mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param})$, $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, $v = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, and $Q = P \oplus \mathsf{KDF}(v)$. One sends back (hp, Q);
  - $\mathtt{Rec}^*(\mathsf{vk}, M, \sigma, 0)$, for a message M and a valid signature σ: $\mathcal{B}$ outputs $c = \mathsf{Commit}(\mathsf{ck}, \sigma; r)$;
  - $\mathtt{Exec}^*(\mathsf{vk}, M, \sigma, P)$: one first runs $\mathtt{Rec}(\mathsf{vk}, M, \sigma, 0)$ to generate c and provide it to $\mathtt{Send}(\mathsf{vk}, M, P)$, to generate (hp, Q).
- At some point, $\mathcal{A}$ outputs a message M and a valid signature σ, and $\mathcal{B}$ has to simulate $\mathtt{Rec}^*(\mathsf{vk}, M, \sigma, b)$: $\mathcal{B}$ sets $\sigma_0 \leftarrow \sigma$ and sets σ_1 as a random string. It sends (σ_0, σ_1) to the challenger of the semantic security of the commitment scheme and gets back c, a commitment of σ_β, for a random unknown bit β. It outputs c;
- $\mathcal{B}$ provides again access to the above oracles, and $\mathcal{A}$ outputs a bit b', that $\mathcal{B}$ forwards as its guess β' for the β involved in the semantic security game for $\mathcal{C}$.

Note that the above simulation perfectly emulates $\mathsf{Exp}^{\mathsf{obl}_A - \beta}_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K})$ (since basically b is β, and b' is β'):

$$\varepsilon = \mathsf{Adv}^{\mathsf{obl}_A}_{\mathcal{OSBE},\mathcal{A}}(\mathfrak{K}) = \Pr[\beta' = 1 | \beta = 1] - \Pr[\beta' = 1 | \beta = 0] = \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{C},\mathcal{B}}(k) \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{C}}(\mathfrak{K}, t).$$

□

Theorem 6.1.3 (Semantically Secure) *$\mathcal{OSBE}$ is **semantically secure** if the signature is* unforgeable*, the SPHF is* smooth *and the commitment scheme is* semantically secure *(and under the pseudo-randomness of the* KDF*):*

$$\mathsf{Adv}^{\mathsf{sem}}_{\mathcal{OSBE}}(\mathfrak{K}, t) \leq q_U \, \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{C}}(\mathfrak{K}, t') + 2\,\mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S}}(k, q_S, t'') + 2\,\mathsf{Adv}^{\mathsf{smooth}}_{\mathrm{SPHF}}(\mathfrak{K}) \text{ with } t', t'' \approx t.$$

In the above formula, q_U denotes the number of interactions the adversary has with the sender, and q_S the number of signing queries the adversary asked.

Proof: Let us assume $\mathcal{A}$ is an adversary against the semantic security of our scheme: The malicious adversary $\mathcal{A}$ is able to tell the difference between an interaction with $\mathcal{S}_0$ (who owns P_0) and $\mathcal{S}_1$ (who owns P_1), with advantage ε. We start from this initial security game, and make slight modifications to bound ε.

Game $\mathcal{G}_0$: Let us emulate this security game:

- $\mathcal{B}$ emulates the initialization of the system: it runs $\mathsf{OSBESetup}$ by itself, and then $\mathsf{OSBEKeyGen}$ to generate $(\mathsf{vk}, \mathsf{sk})$;
- $\mathcal{B}$ has to simulate all the oracles:
  - $\mathtt{Sign}(\mathsf{vk}, M)$ and $\mathtt{Sign}^*(\mathsf{vk}, M)$: it runs the corresponding algorithm by itself;
  - $\mathtt{Send}(\mathsf{vk}, M, P)$, for a message M and a private input P: upon receiving c, one computes $\mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param})$, $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, $v = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, and $Q = P \oplus \mathsf{KDF}(v)$. One sends back (hp, Q);
  - $\mathtt{Rec}(\mathsf{vk}, M, 0)$, for a message M: $\mathcal{B}$ asks for a valid signature σ on M, computes and outputs $c = \mathsf{Commit}(\mathsf{ck}, \sigma; r)$;
  - $\mathtt{Exec}(\mathsf{vk}, M, P)$: one simply first runs $\mathtt{Rec}(\mathsf{vk}, M, 0)$ to generate c, and provide it to $\mathtt{Send}(\mathsf{vk}, M, P)$, to generate (hp, Q).
- At some point, $\mathcal{A}$ outputs a message M and two inputs (P_0, P_1) to distinguish the sender, and $\mathcal{B}$ call back the above $\mathtt{Send}(\mathsf{vk}, M, P_b)$ simulation to interact with $\mathcal{A}$;
- $\mathcal{B}$ provides again access to the above oracles, and $\mathcal{A}$ outputs a bit b'.

In this game, $\mathcal{A}$ has an advantage ε in guessing b:

$$\varepsilon = \Pr_0[b' = 1 | b = 1] - \Pr_{\mathcal{G}_0}[b' = 1 | b = 0] = 2 \times \Pr_{\mathcal{G}_0}[b' = b] - 1.$$

Game ${\mathcal{G}_1}^\beta$: This game involves the semantic security of the commitment scheme: $\mathcal{B}$ is already provided the parameters and the commitment key ck by the challenger of the semantic security of the commitment scheme, hence the initialization is slightly modified. In addition, $\mathcal{B}$ sets the bit $b = \beta$, and modifies the $\mathtt{Rec}$ oracle simulation:

- $\mathtt{Rec}(\mathsf{vk}, M, 0)$, for a message M: $\mathcal{B}$ asks for a valid signature σ_0 on M, and sets σ_1 as a random string, computes and outputs $c = \mathsf{Commit}(\mathsf{ck}, \sigma_b; r)$.

Since $\mathcal{B}$ knows b, it finally outputs $\beta' = (b' = b)$.

Note that $\mathcal{G}_1^0$ is exactly $\mathcal{G}_0$, and the distance between $\mathcal{G}_1^0$ and $\mathcal{G}_1^1$ relies on the Left-or-Right security of the commitment scheme, which can be shown equivalent to the semantic security, with a lost linear in the number of commitment queries, which is actually the number q_U of interactions with a user (the sender in this case), due to the hybrid argument [BDJR97]:

$$\begin{aligned} q_U \times \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(k) &\geq \Pr[\beta' = 1 | \beta = 0] - \Pr[\beta' = 1 | \beta = 1] \\ &= \Pr[b' = b | \beta = 0] - \Pr[b' = b | \beta = 1] \\ &= (2 \times \Pr_{\mathcal{G}_1^0}[b' = b] - 1) - (2 \times \Pr_{\mathcal{G}_1^1}[b' = b] - 1) \end{aligned}$$

As a consequence: $\varepsilon \leq q_U \times \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{C}}(\mathfrak{K}) + (2 \times \Pr_{\mathcal{G}_1^1}[b' = b] - 1)$.

Game $\mathcal{G}_2$: This game involves the unforgeability of the signature scheme: $\mathcal{B}$ is already provided the parameters and the verification vk for the signature scheme, together with access to the signing oracle (note that all the signing queries Sign^* asked by the adversary in the $\mathtt{FIND}$ stage, i.e. , before the challenge interaction with $\mathtt{Send}(\mathsf{vk}, M, P_b)$, are stored in $\mathcal{SM}$). The simulator $\mathcal{B}$ generates itself all the other parameters and keys, an namely the commitment key ck, together with the associated extraction key ek. For the $\mathtt{Rec}$ oracle simulation, $\mathcal{B}$ keeps the random version (as in $\mathcal{G}_1^1$). In the challenge interaction with $\mathtt{Send}(\mathsf{vk}, M, P_b)$, one stops the simulation and makes the adversary win if it uses a valid signature on a message $M \notin \mathcal{SM}$:

- $\mathtt{Send}(\mathsf{vk}, M, P_b)$, during the challenge interaction: upon receiving c, if $\mathcal{M} \notin \mathcal{SM}$, it first extracts c to get the input signature σ. If σ is a valid signature, one stops the game, sets $b' = b$ and outputs b'. If the signature is in not valid, the simulation remains unchanged;
- $\mathtt{Rec}(\mathsf{vk}, M, 0)$, for a message M: $\mathcal{B}$ sets σ as a random string, computes and outputs $c = \mathsf{Commit}(\mathsf{ck}, \sigma; r)$.

Because of the abort in the case of a valid signature on a new message, we know that the adversary cannot use such a valid signature in the challenge. So, since M should not be in $\mathcal{SM}$, the signature will be invalid. Actually, the unique difference from the previous game $\mathcal{G}_1^1$ is the abort in case of valid signature on a new message in the challenge phase, which probability is bounded by $\mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S}}(\mathfrak{K}, q_S)$. Using Shoup's Lemma [Sho02]:

$$\Pr_{\mathcal{G}_1^1}[b' = b] - \Pr_{\mathcal{G}_2}[b' = b] \leq \mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S}}(\mathfrak{K}, q_S).$$

As a consequence: $\varepsilon \leq q_U \times \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{C}}(\mathfrak{K}) + 2 \times \mathsf{Succ}^{\mathsf{euf}}_{\mathcal{S}}(\mathfrak{K}, q_S) + (2 \times \Pr_{\mathcal{G}_2}[b' = b] - 1)$.

Game $\mathcal{G}_3$: The last game involves the smoothness of the SPHF: The unique difference is in the computation of v in `Send` simulation, in the challenge phase only: $\mathcal{B}$ chooses a random $v \in \mathbb{G}$. Due to the statistical randomness of v in the previous game, in case the signature is not valid (a word that is not in the language), this game is statistically indistinguishable from the previous one:

$$\Pr_{\mathcal{G}_2}[b' = b] - \Pr_{\mathcal{G}_3}[b' = b] \leq \mathsf{Adv}^{\mathsf{smooth}}_{\mathrm{SPHF}}(\mathfrak{K}).$$

Since P_b is now masked by a truly random value, no information leaks on b: $\Pr_{\mathcal{G}_3}[b' = b] = 1/2$. □

Theorem 6.1.4 *$\mathcal{OSBE}$ is **semantically secure w.r.t. the authority** if the SPHF is* pseudo-random *(and under the pseudo-randomness of the* KDF*):*

$$\mathsf{Adv}^{\mathsf{sem}^*}_{\mathcal{OSBE}}(\mathfrak{K}, t) \leq 2 \times \mathsf{Adv}^{\mathsf{pr}}_{\mathrm{SPHF}}(\mathfrak{K}, t).$$

Proof: Let us assume $\mathcal{A}$ is an adversary against the semantic security w.r.t. the authority: The malicious adversary $\mathcal{A}$ is able to tell the difference between an eavesdropped interaction with $\mathcal{S}_0$ (who owns P_0) and $\mathcal{S}_1$ (who owns P_1), with advantage ε. We start from this initial security game, and make slight modifications to bound ε.

Game $\mathcal{G}_0$: Let us emulate this security game:

- $\mathcal{B}$ emulates the initialization of the system: it runs $\mathsf{OSBESetup}$ by itself;
- $\mathcal{A}$ provides the verification key vk;
- $\mathcal{B}$ has to simulate all the oracles:
  - $\mathtt{Send}(\mathsf{vk}, M, P)$, for a message M and a private input P: upon receiving c, one computes $\mathsf{hk} = \mathsf{HashKG}(\mathcal{L}, \mathsf{param})$, $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, $v = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), c)$, and $Q = P \oplus \mathsf{KDF}(v)$. One sends back (hp, Q);
  - $\mathtt{Rec}^*(\mathsf{vk}, M, \sigma, 0)$, for a message M and a valid signature σ: $\mathcal{B}$ outputs $c = \mathsf{Commit}(\mathsf{ck}, \sigma; r)$;
  - $\mathtt{Exec}^*(\mathsf{vk}, M, \sigma, P)$: one first runs $\mathtt{Rec}(\mathsf{vk}, M, \sigma, 0)$ to generate c, that is provided to $\mathtt{Send}(\mathsf{vk}, M, P)$, to generate (hp, Q).
- At some point, $\mathcal{A}$ outputs a message M with a valid signature σ, and two inputs (P_0, P_1) to distinguish the sender, and $\mathcal{B}$ call back the above $\mathtt{Send}(\mathsf{vk}, M, P_b)$ and $\mathtt{Rec}^*(\mathsf{vk}, M, \sigma, 0)$ simulations to interact together and output the transcript $(c; \mathsf{hp}, Q)$;
- $\mathcal{B}$ provides again access to the above oracles, and $\mathcal{A}$ outputs a bit b'.

In this game, $\mathcal{A}$ has an advantage ε in guessing b:

$$\varepsilon = \Pr_{\mathcal{G}_0}[b' = 1 | b = 1] - \Pr_{\mathcal{G}_0}[b' = 1 | b = 0] = 2 \times \Pr_{\mathcal{G}_0}[b' = b] - 1.$$

Game $\mathcal{G}_1$: This game involves the pseudo-randomness of the SPHF: The unique difference is in the computation of v in `Send` simulation of the eavesdropped interaction, and so for the transcript: $\mathcal{B}$ chooses a random $v \in \mathbb{G}$ and computes $Q = P_b \oplus \mathsf{KDF}(v)$. Due to the pseudo-randomness of v in the previous game, since $\mathcal{A}$ does not know the random coins r used to commit σ, this game is computationally indistinguishable from the previous one.

$$\Pr_{\mathcal{G}_1}[b' = b] - \Pr_{\mathcal{G}_0}[b' = b] \leq \mathsf{Adv}^{\mathsf{pr}}_{\mathrm{SPHF}}(\mathfrak{K}, t).$$

Since P_b is now masked by a truly random value v, no information leaks on b: $\Pr_{\mathcal{G}_1}[b' = b] = 1/2$.

□

6.1.3 Concrete Instantiation

Our first construction combines the linear encryption scheme [BBS04], the Waters signature scheme [Wat05] and a SPHF on linear ciphertexts [CS02, Sha07] (See section 5.2, page 97, for an explanation of how to adapt a SPHF on linear ciphertext to a SPHF on a valid signature in a linear ciphertext). The overall security then relies on the DLin assumption, a quite standard assumption in the standard model. Its efficiency is of the same order of magnitude than the construction based on identity-based encryption [LDB03] (that only achieves weaker security notions) and better than the RSA-based scheme which provides similar security guarantees (in the random oracle model).

The linear encryption is once again viewed as a commitment scheme, as the decryption key is never to be used except maybe in simulation during security proofs. To stress this point, we will use ck to design the encryption key.

We now define our OSBE protocol, where a sender $\mathcal{S}$ wants to send a private message $P \in \{0,1\}^\ell$ to a recipient $\mathcal{R}$ in possession of a Waters signature on a message M. We will use a smooth projective hash function on the language of valid commitment of Waters Signature as explained in section 5.2, page 97

- $\mathsf{OSBESetup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, defines a bilinear environment $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, the public parameters $h \xleftarrow{\$} \mathbb{G}$, a commitment key $\mathsf{ck} = (Y_1 = g^{y_1}, Y_2 = g^{y_2})$, where $(y_1, y_2) \xleftarrow{\$} \mathbb{Z}_p^2$, and $\vec{u} = (u_0, \ldots, u_k) \xleftarrow{\$} \mathbb{G}^{k+1}$ for the Waters signature. All these elements constitute the string param;
- $\mathsf{OSBEKeyGen}(\mathsf{param})$, the authority generates a pair of keys $(\mathsf{vk} = g^z, \mathsf{sk} = h^z)$ for a random scalar $z \xleftarrow{\$} \mathbb{Z}_p$;
- $\mathsf{OSBESign}(\mathsf{sk}, M)$ produces a signature $\sigma = (h^z \mathcal{F}(M)^s, g^s)$;
- $\mathsf{OSBEVerif}(\mathsf{vk}, M, \sigma)$ checks if $e(\sigma_1, g) = e(\sigma_2, \mathcal{F}(M)) \cdot e(h, \mathsf{vk})$.
- $\mathsf{OSBEProtocol}\langle \mathcal{S}(\mathsf{vk}, M, P), \mathcal{R}(\mathsf{vk}, M, \sigma) \rangle$ runs as follows:
  - $\mathcal{R}$ chooses random $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sends a linear encryption of σ: $C = (c_1 = \mathsf{ck}_1^{r_1}, c_2 = \mathsf{ck}_2^{r_2}, c_3 = g^{r_1+r_2} \cdot \sigma_1, \sigma_2)$
  - $\mathcal{S}$ chooses random $x_1, x_2, x_3 \xleftarrow{\$} \mathbb{Z}_p^3$ and computes:
    - $\mathsf{HashKG}(\mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, M)) = \mathsf{hk} = (x_1, x_2, x_3)$;
    - $\mathsf{Hash}(\mathsf{hk}; \mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, M), C) = v = e(c_1, g)^{x_1} e(c_2, g)^{x_2} (e(c_3, g)/(e(h, \mathsf{vk}) e(\mathcal{F}(M), \sigma_2)))^{x_3}$;
    - $\mathsf{ProjKG}(\mathsf{hk}; \mathsf{WLin}(\mathsf{ck}, \mathsf{vk}, M), C) = \mathsf{hp} = (\mathsf{ck}_1^{x_1} g^{x_3}, \mathsf{ck}_2^{x_2} g^{x_3})$.
  - $\mathcal{S}$ then sends $(\mathsf{hp}, Q = P \oplus \mathsf{KDF}(v))$ to $\mathcal{R}$;
  - $\mathcal{R}$ computes $v' = e(\mathsf{hp}_1^{r_1} \mathsf{hp}_2^{r_2}, g)$ and $P' = Q \oplus \mathsf{KDF}(v')$.

As always, we have also proposed an asymmetric instantiation with an ElGamal encryption in $\mathbb{G}_1$, and relying on an asymmetric Waters so CDH^+. In should be noted that in this case, we would only rely on XDH, as we do not need to commit anything in $\mathbb{G}_2$.

Security and Efficiency

We now provide a security analysis of this scheme. This instantiation differs, from the high-level instantiation presented before, in the ciphertext C of the signature $\sigma = (\sigma_1, \sigma_2)$. The second half of the signature indeed remains in clear. It thus does not guarantee the semantic security on the signature used in the ciphertext. However, granted Waters signature randomizability, one can re-randomize the signature each time, and thus provide a totally new σ_2: it does not leak any information about the original signature. The first part of the ciphertext (c_1, c_2, c_3) does not leak any additional information under the DLin assumption. As a consequence, the global ciphertext guarantees the semantic security of the original signature if a new re-randomized signature is encrypted each time. We can now apply the high-level construction security, and all the assumptions hold under the DLin one:

Theorem 6.1.5 *Our OSBE scheme is secure (i.e. oblivious w.r.t. the authority, semantically secure, and semantically secure w.r.t. the authority) under the* DLin *assumption (and the pseudo-random generator in the* KDF*).*

Our proposed scheme needs one communication for $\mathcal{R}$ and one for $\mathcal{S}$, so it is round-optimal. Communication also consists of few elements, $\mathcal{R}$ sends 4 group elements, and $\mathcal{S}$ answers with 2 group elements only and an ℓ-bit string for the masked $P \in \{0,1\}^\ell$. As explained in Remark 6.1.1, page 104, this has to be compared with the RSA-based scheme from [LDB03] which requires 2 elements in RSA groups (with double-length modulus). For a 128-bit security level, using standard Type-I bilinear groups implementation, we obtain a 62.5% improvement[1] in communication complexity over the RSA-based scheme proposed in the original paper [LDB03].

While reducing the communication cost of the scheme, we have improved its security and it now fits the proposed applications. In [LDB03], such schemes were proposed for applications where someone wants to transmit a confidential information to an agent belonging to a specific agency. However the agent does not want to give away his signature. As they do not consider eavesdropping and replay in their semantic security nothing prevents an adversary to replay a part of a previous interaction to impersonate a CIA agent (to recall their example). In practice, an additional secure communication channel, such as with SSL, was required in their security model, hence increasing the communication cost: our protocol is secure by itself.

6.2 Round-Optimal Blind Signature Revamped

We now present a new way to obtain a blind signature scheme in the standard model under classical assumptions with a common-reference string. This is an improvement over [BFPV11], presented in chapter 4, page 70. We are going to use the same building blocks as before, so linear encryption, Waters signatures and a Smooth Projective Hash Function on linear ciphertexts. More elaborated languages will be required, but just conjunctions and disjunctions of classical languages, as done in [ACP09] (see section 2.2.4, page 24, and section 5.1.2, page 95), hence the efficient construction. Once again we will use, ck to denote the public key of the encryption scheme, as we won't use the decryption key in the real world. Our blind signature scheme is defined by:

- $\mathsf{BSSetup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates a pairing-friendly system $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ and an extractable commitment key $\mathsf{ck} = (u, v, g) \in \mathbb{G}^3$. It also chooses at random $h \in \mathbb{G}$ and generators $\vec{u} = (u_i)_{i \in [\![1,\ell]\!]} \in \mathbb{G}^\ell$ for the Waters function. It outputs the global parameters $\mathsf{param} = (p, \mathbb{G}, \mathbb{G}_T, e, g, \mathsf{ck}, h, \vec{u})$;
- $\mathsf{BSKeyGen}(\mathsf{param})$ picks at random a secret key $\mathsf{sk} = x$ and computes the verification key $\mathsf{vk} = g^x$;
- $\mathsf{BSProtocol}\langle\mathcal{S}(\mathsf{sk}), \mathcal{U}(\mathsf{vk}, m)\rangle$ runs as follows, where $\mathcal{U}$ wants to get a signature on M
  - $\mathcal{U}$ computes the bit-per-bit encryption of M by encrypting each $u_i^{M_i}$ in b_i, $\forall i \in [\![1,\ell]\!], b_i = \mathsf{Commit}(\mathsf{ck}, u_i^{M_i}; (r_{i,1}, r_{i,2})) = (u^{r_{i,1}}, v^{r_{i,2}}, g^{r_{i,1}+r_{i,2}} u_i^{M_i})$ where $(r_{i,1}, r_{i,2}) \xleftarrow{\$} \mathbb{Z}_p^2$. Then writing $r_1 = \sum r_{i,1}$ and $r_2 = \sum r_{i,2}$, he computes the encryption c of $\mathsf{vk}^{r_1+r_2}$ for two random scalars $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$ with $\mathsf{Commit}(\mathsf{ck}, \mathsf{vk}^{r_1+r_2}; s_1, s_2) = (u^{s_1}, v^{s_2}, g^{s_1+s_2}\mathsf{vk}^{r_1+r_2})$. $\mathcal{U}$ then sends $(c, (b_i))$;
  - On input of these ciphertexts, the algorithm $\mathcal{S}$ computes the corresponding SPHF, considering the language $\mathcal{L}$ of valid ciphertexts. This is a conjunction of several languages:
    1. One checking that each b_i encrypts a bit in basis u_i: in $\mathsf{BLin}(\mathsf{ck}, u_i)$;
    2. One considering $(d_1, d_2, c_1, c_2, c_3)$, that checks if (c_1, c_2, c_3) encrypts an element d_3 such that (d_1, d_2, d_3) is a linear tuple in basis (u, v, vk): in $\mathsf{ELin}(\mathsf{ck}, \mathsf{vk})$, where $d_1 = \prod_i b_{i,1}$ and $d_2 = \prod_i b_{i,2}$.
  - $\mathcal{S}$ computes the corresponding Hash-value v, extracts $K = \mathsf{KDF}(v) \in \mathbb{Z}_p$, generates the blinded signature $(\sigma_1'' = h^x \delta^s, \sigma_2' = g^s)$, where $\delta = u_0 \prod_i b_{i,3} = \mathcal{F}(M) g^{r_1+r_2}$, and sends $(\mathsf{hp}, Q = \sigma_1'' \times g^K, \sigma_2')$;
  - Upon receiving $(\mathsf{hp}, Q, \sigma_2')$, using its witnesses and hp, $\mathcal{U}$ computes the $\mathsf{ProjHash}$-value v', extracts $K' = \mathsf{KDF}(v')$ and unmasks $\sigma_1'' = Q \times g^{-K'}$. Thanks to the knowledge of r_1 and r_2, it can compute $\sigma_1' = \sigma_1'' \times (\sigma_2')^{-r_1-r_2}$. Note that if $v' = v$, then $\sigma_1' = h^x \mathcal{F}(M)^s$, which together with $\sigma_2' = g^s$ is a valid Waters signature on M. It can thereafter re-randomize the final signature $\sigma = (\sigma_1' \cdot \mathcal{F}(M)^{s'}, \sigma_2' \cdot g^{s'})$.
- $\mathsf{BSVerif}(\mathsf{vk}, M, \sigma)$, checks whether $e(\sigma_1, g) = e(h, \mathsf{vk}) \cdot e(\mathcal{F}(M), \sigma_2)$.

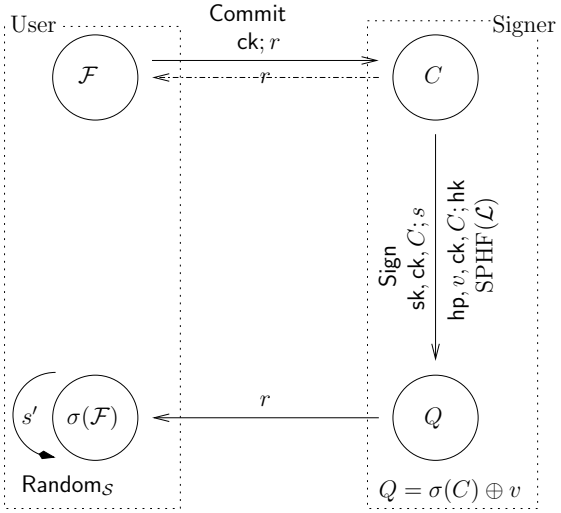
[1] The improvement is even more important for the asymmetric scheme since, using standard Type-II or Type-III bilinear groups, the communication complexity is only 3/16-th of the one of the RSA-based scheme.

Our idea is to remove any kind of proof of knowledge in the protocol, which was the main concern in section 4.2.2, page 77, and so instead we use the SPHF to assure the signer that the user will not be able to exploit the signature on a different message than the one it had committed to in the ciphertext when it asked for its signature. This way, we obtain a protocol where the user first sends $3\,\ell+6$ group elements for the ciphertext, and receives back $5\,\ell+4$ elements for the projection key and 2 group elements for the blinded signature. So $8\,\ell+12$ group elements are used in total. This has to be compared to $9\,\ell+24$ before. We both reduce the linear and the constant parts in the number of group elements involved while relying on the same hypotheses. And the final result is still a standard Waters signature.

In the table below we give a compared evaluation costs between this construction based on smooth projective hash functions, and our previous one relying on Groth-Sahai methodology.

Symmetric Pairing	$\mathbb{G}$
Groth-Sahai based	$9\,k+24$
with SPHF	$8\,k+12$

Asymmetric Pairing	$\mathbb{G}_1$	$\mathbb{G}_2$
Groth-Sahai based	$6\,k+9$	$6\,k+7$
with SPHF	$5\,k+6$	1



Remark In [GRS+11], Garg *el al.* proposed the first round-optimal blind signature scheme in the standard model, without CRS. In order to remove the CRS, their scheme makes use of ZAPs [DN07], a two-round public-coin witness-indistinguishable proof system, and is quite inefficient. Moreover, its security relies on a stronger assumption (namely, sub-exponential hardness of one-to-one one-way functions). A natural idea is to replace the CRS in our scheme with Groth-Ostrovsky-Sahai ZAP [GOS06a] based on the DLin assumption. This change would only double the communication complexity, but we do not know how to prove the security of the resulting scheme[2]. It remains a tantalizing open problem to design an efficient round-optimal blind signature in the standard model without CRS.

6.3 Language Authenticated Key Exchange

In Section 5.3, page 99 we have shown how to build a Smooth Projective Hash Function on variant of Linear Cramer-Shoup, in other words, we have presented a protocol, that lets a user commit to word in a language, and then show that the word indeed belonged to a language.

We are going to use this primitive as the primary building block to build our LAKE protocols. Informally, this primitive allows two users to agree on a common key, if each possess a word and a witness that this word belongs to the language expected by the other user. As we will explain, a PAKE, where both users have to possess the same password is a simple case of LAKE.

This new primitive encompasses the previous notions of PAKE and Secret Handshakes. It is closely related to CAKE and we call it LAKE, for *Language-Authenticated Key-Exchange*, since parties establish a common key if and only if they hold credentials that belong to specific (and possibly independent) languages that the players do not need to agree on prior to the protocol execution. The definition of the primitive is more practice-oriented than the definition of CAKE from [CCGS10] but the two notions are very similar[3]. In particular, the new primitive enables privacy-preserving authentication and key

[2] Indeed, opening the commitment scheme in the ZAP and forging a signature relies on the same computational assumption, which makes it impossible to apply the complexity leveraging argument from [GRS+11].

[3] Actually we believe that any *interesting* AKE primitive can be formalized in either way.

exchange protocols by allowing two members of the same group to secretly and privately authenticate to each other without revealing this group beforehand.

In order to define the security of this primitive, we use the UC framework and an appropriate definition for languages that permits to dissociate the public part of the policy, the private common information the users want to check and the (possibly independent) secret values each user owns that assess the membership to the languages. We provide an ideal functionality for LAKE and give efficient realizations of the new primitive (for a large family of languages) secure under classical mild assumptions, in the standard model (with a common reference string – CRS), with static corruptions.

We significantly improve the efficiency of several CAKE protocols [CCGS10] for specific languages and we enlarge the set of languages for which we can construct practical schemes. Notably, we obtain a very practical realization of Secret Handshakes that provides very strong security properties.

6.3.1 Definitions

The main goal of an *Authenticated Key Exchange* (AKE) protocol is to enable two parties to establish a shared, cryptographically strong key over an insecure network under the complete control of an adversary. AKE is one of the most widely used and fundamental cryptographic primitives. In order for AKE to be possible, the parties must have authentication means, *e.g.* (public or secret) cryptographic keys, short (*i.e.*, low-entropy) secret keys or *credentials* that satisfy a (public or secret) policy.

Password-Authenticated Key Exchange (PAKE) was formalized by Bellovin and Merritt [BM92] and followed by many proposals based on different cryptographic assumptions (see [ACP09, CCGS10] and references therein). It allows users to generate a strong cryptographic key based on a shared "human-memorable" (*i.e.* low-entropy) password without requiring a public-key infrastructure. In this setting, an adversary controlling all communication in the network and able to corrupt participants at any time should not be able to mount an off-line dictionary attack.

The concept of *Secret Handshakes* has been introduced in 2003 by Balfanz, Durfee, Shankar, Smetters, Staddon and Wong [BDS+03] (see also [JL09, AKB07]). It allows two members of the same group to identify each other secretly, in the sense that each party reveals his affiliation to the other only if they are members of the same group. At the end of the protocol, the parties can set up an ephemeral session key for securing further communication between them and an outsider is unable to determine if the handshake succeeded.

More recently, *Credential-Authenticated Key Exchange* (CAKE) were presented by Camenisch, Casati, Groß and Shoup [CCGS10]. In this primitive, a common key is established if and only if a specific relation is satisfied between credentials hold by the two players.

The LAKE primitive includes PAKE and Secret Handshakes: in the former case, the credentials are just passwords, and the policy says that the two passwords must be equal, and in the latter case, owning a credential serves as evidence of membership in a group.

6.3.2 The Ideal Functionality

We tweak the Password-Authenticated Key Exchange functionality $\mathcal{F}_{\text{PAKE}}$ (first provided in [CHK+05]) to handle more complex languages: the players now agree on a common secret key if and only if they own words that lie in the languages the partners have in mind. Using notations from Section 5, page 93 this can be said more precisely: after an agreement on pub between P_i and P_j, player P_i uses a word W_i belonging to $\mathcal{L}_i = \mathcal{L}_{\mathcal{R}_i}(\mathsf{pub}, \mathsf{priv}_i)$ and it expects its partner P_j to use a word W_j belonging to the language $\mathcal{L}'_j = \mathcal{L}_{\mathcal{R}_j}(\mathsf{pub}, \mathsf{priv}'_j)$. We assume relations $\mathcal{R}_i$ and $\mathcal{R}_j$ to be specified by the kind of protocol we study (PAKE, verifier-based PAKE, secret handshakes, ...) and so the languages are defined by the parameters pub, priv_i and priv_j: they both agree on the public part pub, to be possibly parsed in a different way by each player for each language according to the relations, player P_i owns $W_i \in \mathcal{L}_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_i) \subseteq \mathcal{S}_i$, for $\mathsf{priv}_i \in \mathcal{P}_i$, and expects player P_j to use the language $L'_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j) \subseteq \mathcal{S}_j$, for $\mathsf{priv}'_j \in \mathcal{P}_j$. Symmetrically, player P_j owns $W_j \in \mathcal{L}_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_j) \subseteq \mathcal{S}_j$ and expects player P_i to use the language $L'_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_i) \subseteq \mathcal{S}_i$. The subsets $\mathcal{S}_i, \mathcal{S}_j$ and $\mathcal{P}_i, \mathcal{P}_j$ are assumed public and determined by $\mathcal{R}_i$ and $\mathcal{R}_j$, and thus by the kind of protocol.

Note however that the respective languages do not need to be the same or to use similar relations: authentication means could be totally different for the 2 players. The key exchange should succeed if and only if the two following pairs of equations hold: ($\mathcal{L}'_i = \mathcal{L}_i$ and $W_i \in \mathcal{L}_i$) and ($\mathcal{L}'_j = \mathcal{L}_j$ and $W_j \in \mathcal{L}_j$).

The functionality $\mathcal{F}_{\text{LAKE}}$ is parametrized by a security parameter $\mathfrak{K}$. It interacts with an adversary $\mathcal{S}$ and a set of parties $P_1,\ldots,P_n$ via the following queries, where they already agree on the public parameter pub of the languages:

- New Session: Upon receiving a query $(\textit{NewSession} : \mathsf{sid}, P_i, P_j, W_i, \mathcal{L}_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_i), \mathcal{L}'_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j))$ from P_i or $\mathcal{S}$ (thus impersonating P_i [a]),
  - If this is the first *NewSession*-query with identifier sid, record the tuple $(P_i, P_j, W_i, \mathcal{L}_i, \mathcal{L}'_j, \mathsf{initiator})$. Send $(\textit{NewSession}; \mathsf{sid}, P_i, P_j, \mathsf{pub}, \mathsf{initiator})$ to $\mathcal{S}$ and P_j.
  - If this is the second *NewSession*-query with identifier sid and there is a record $(P_j, P_i, W_j, \mathcal{L}_j, \mathcal{L}'_i, \mathsf{initiator})$, record the tuple $(P_j, P_i, W_j, \mathcal{L}_j, \mathcal{L}'_i, \mathsf{initiator}, W_i, \mathcal{L}_i, \mathcal{L}'_j, \mathsf{receiver})$. Send $(\textit{NewSession}; \mathsf{sid}, P_i, P_j, \mathsf{pub}, \mathsf{receiver})$ to $\mathcal{S}$ and P_j.
- Key Computation: Upon receiving a query $(\textit{NewKey} : \mathsf{sid})$ from $\mathcal{S}$, if there is a record of the form $(P_i, P_j, W_i, \mathcal{L}_i, \mathcal{L}'_j, \mathsf{initiator}, W_j, \mathcal{L}_j, \mathcal{L}'_i, \mathsf{receiver})$ and this is the first *NewKey*-query for session sid, then
  - If $(\mathcal{L}'_i = \mathcal{L}_i$ and $W_i \in \mathcal{L}_i)$ and $(\mathcal{L}'_j = \mathcal{L}_j$ and $W_j \in \mathcal{L}_j)$, then pick a random key sk of length k and store $(\mathsf{sid}, \mathsf{sk})$. Send $(\mathsf{sid}, \mathsf{success})$ to $\mathcal{S}$.
  - Else, store $(\mathsf{sid}, \perp)$, and $(\mathsf{sid}, \mathsf{fail})$ is sent to $\mathcal{S}$.
- Key Delivery: Upon receiving a query $(\textit{SendKey} : \mathsf{sid}, P_i, \mathsf{sk})$ from $\mathcal{S}$, then
  - if there is a record of the form $(\mathsf{sid}, \mathsf{sk}')$, then, if both players are uncorrupted, output $(\mathsf{sid}, \mathsf{sk}')$ to P_i. Otherwise, output $(\mathsf{sid}, \mathsf{sk})$ to P_i.
  - if there is a record of the form $(\mathsf{sid}, \perp)$, then, if both players are uncorrupted, pick a random key sk' of length k and output $(\mathsf{sid}, \mathsf{sk}')$ to P_i. Otherwise, output $(\mathsf{sid}, \mathsf{sk})$ to P_i.

[a] This is possible when P_i is corrupted (controlled by the adversary), and then any message to P_i is sent to $\mathcal{S}$

Figure 6.3: Ideal Functionality $\mathcal{F}_{\text{LAKE}}$

Description

In [CHK+05], the initial $\mathcal{F}_{\text{PAKE}}$ functionality gave the adversary access to a TestPwd-query, which modeled the on-line dictionary attack. But it is known since [BCL+05] and [ACGP11] that it is equivalent to use the split functionality model [BCL+05], generate the *NewSession*-queries corresponding to the corrupted players and tell the adversary whether the protocol succeeds or not and see how the protocol terminates. Both methods enable the adversary to try a credential for a player (on-line dictionary attack). The second method (that we use here) implies allowing $\mathcal{S}$ to ask *NewSession*-queries on behalf of the corrupted player (hence the slight change in the *NewSession*-query on Figure 6.3). It also implies letting the adversary become aware of the success or failure of the protocol [KS05], but this does not change from the view of a real-life adversary since this success/failure information is available to the adversary in practice by simply eavesdropping the conversation between P_i and P_j and checking whether it continues (the protocol succeeded) or not (it failed). To this aim, the *NewKey*-query informs the adversary whether the credentials are consistent with the languages or not. In addition, the split functionality model guarantees from the beginning which player is honest and which one is controlled by the adversary. This finally allows us to get rid of the TestPwd-query.

The security goal is to show that the best attack for the adversary is a basic trial execution with a credential of its guess or choice: the proof will thus consist in emulating any real-life attack by either a trial execution by the adversary, playing as an honest player would do, but with a credential chosen by the adversary or obtained in any way; or a denial of service, where the adversary is clearly aware that its behavior will make the execution fail.

6.3.3 A First Generic Construction

Using smooth projective hash functions on commitments, one can generically define a LAKE protocol as done in [ACP09]. The basic idea is to make the player commit to their private information (for the expected languages and the owned words), and eventually the smooth projective hash functions will be

used to make implicit validity checks of the global relation.

To this aim, we use the commitments and associated smooth projective hash function as described in Section 5.3, page 99. More precisely, all examples of the previous SPHF in chapter II, page 92 can be used on commitments divided into one or two parts (the non-equivocable LCSCom or the equivocable $\mathsf{DLCSCom}'$ commitments), the first part being denoted as $\mathcal{C}$, and the second part (if needed) being denoted as $\mathcal{C}'$. In the case of a two-part commitment, the first step of the commitment will consist in sending $\mathcal{C}$ along with a Pedersen commitment $\mathcal{C}''$ on $\mathcal{C}'$ and the second step of the commitment will consist in sending $\mathcal{C}'$ and t to check the Pedersen commitment. The relations on the committed values will not be explicitly checked, since the values will never be revealed, but will be implicitly checked using SPHF. It is interesting to note that in both cases (one-part or two-part commitment), the projected hash key will only depend on the first part $\mathcal{C}$ of the commitment.

As it is often the case in the UC setting, we need the initiator to use stronger primitives than the receiver. They both have to use non-malleable and extractable commitments, but the initiator will use a commitment that is additionally equivocable, the $\mathsf{DLCSCom}'$ in two parts ($(\mathcal{C}_i, \mathcal{C}'_i)$ and $\mathsf{Com}_i = \mathcal{C}_i \cdot \mathcal{C}_i'^{\vec{\varepsilon}}$), while the receiver will only need the basic LCSCom commitment in one part ($\mathsf{Com}_j = \mathcal{C}_j$).

As already explained, SPHF will be used to implicitly check whether ($\mathcal{L}'_i = \mathcal{L}_i$ and $W_i \in \mathcal{L}_i$) and ($\mathcal{L}'_j = \mathcal{L}_j$ and $W_j \in \mathcal{L}_j$). But since in our instantiations private parameters priv and words W will have to be committed, the structure of these commitments will thus be publicly known in advance: commitments of $\mathcal{P}$-elements and $\mathcal{S}$-elements. Chapter 5, page 93 discusses on the languages captured by our definition, and the next sections will illustrate with some AKE protocols. However, while these $\mathcal{P}$ and $\mathcal{S}$ sets are embedded in $\mathbb{G}^n$ from some n, it might be important to prove that the committed values are actually in $\mathcal{P}$ and $\mathcal{S}$ (e.g., one can want to prove it commits bits, whereas they are first embedded as group elements in $\mathbb{G}$ of large order p). This will be an additional language to check on the commitments, but it will be possible to check it on the $\mathcal{C}$ part only, whatever the kind of commitment used, since equivocability will not be required for this sub-language.

Security Analysis

This leads to a very simple protocol described on Figure 6.4. Since we have to assume common pub, we make a first round (with flows in each direction) where the players send their contribution, to come up with pub. These flows will also be used to know which player is honest and which player is controlled by the adversary (as with the Split Functionality [BCL$^+$05]). In case the languages have empty pub, these additional flows are not required, since the Split Functionality can be applied on the committed values, and thus the signing key for the receiver will not be required anymore. This LAKE protocol is secure against static corruptions. The proof is provided in the next section, and is in the same vein as the one in [ACP09]. However, it is a bit more intricate:

- in PAKE, when one is simulating a player, and knows the adversary used the correct password, one simply uses this password for the simulated player. In LAKE, when one knows the language expected by the adversary for the simulated player and has to simulate a successful execution (because of success announced by the *NewKey*-query), one has to actually include a correct word in the commitment: smooth projective hash functions do not allow the simulator to cheat, equivocability of the commitment is the unique trapdoor, but with a valid word. The languages must allow the simulator to produce a valid word W in $\mathcal{L}(\mathsf{pub}, \mathsf{priv})$, for any pub and $\mathsf{priv} \in \mathcal{P}$ provided by the adversary or the environment. This will be the case in all the interesting applications of our protocol (see Section 6.4, page 120): if priv defines a Waters' verification key vk, with the master key s such that $h = g^s$, the signing key is $\mathsf{sk} = \mathsf{vk}^s$, and thus the simulator can sign any message; if such a master key does not exist, one can restrict $\mathcal{P}$, and implicitly check it with the SPHF (the additional language check, as said above).
- In addition, as already noted, our commitment $\mathsf{DLCSCom}'$ is not formally binding (contrarily to the much less efficient one used in [ACP09]). The adversary can indeed make the extraction give $\vec{M}$ from $\mathcal{C}_i$, whereas Com_i will eventually contain $\vec{M}'$ if $\mathcal{C}'_i$ does not encrypt $(1_{\mathbb{G}})^n$. However, since the actual value $\vec{M}'$ depends on the random challenge $\vec{\varepsilon}$, and the language is assumed sparse (otherwise authentication is easy), the protocol will fail: this can be seen as a denial of service from the adversary.

Theorem 6.3.1 *Our LAKE scheme from Figure 6.4 realizes the $\mathcal{F}_{\text{LAKE}}$ functionality in the $\mathcal{F}_{\text{CRS}}$-hybrid model, in the presence of static adversaries, under the* DLin *assumption and the security of the One-Time Signature.*

Execution between P_i and P_j, with session identifier sid.

- Preliminary Round: each user generates a pair of signing/verification keys $(\mathsf{SK}, \mathsf{VK})$ and sends VK together with its contribution to the public part of the language.

We denote by $\ell_i = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{VK}_i, \mathsf{VK}_j)$ and by $\ell_j = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{VK}_j, \mathsf{VK}_i)$, where pub is the combination of the contributions of the two players. The initiator now uses a word W_i in the language $\mathcal{L}(\mathsf{pub}, \mathsf{priv}_i)$, and the receiver uses a word W_j in the language $\mathcal{L}(\mathsf{pub}, \mathsf{priv}_j)$ [a]. We assume commitments and associated smooth projective hash functions exist for these languages.

- First Round: user P_i (with random tape ω_i) generates a multi-$\mathsf{DLCSCom}'$ commitment on $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ in $(\mathcal{C}_i, \mathcal{C}'_i)$, under the label ℓ_i. It also computes a Pedersen commitment on $\mathcal{C}'_i$ in $\mathcal{C}''_i$ (with random exponent t). It then sends $(\mathcal{C}_i, \mathcal{C}''_i)$ to P_j;
- Second Round: user P_j (with random tape ω_j) computes a multi-LCS commitment on $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$ in $\mathsf{Com}_j = \mathcal{C}_j$, with witness $\vec{r}$, under the label ℓ_j. It then generates a challenge $\vec{\varepsilon}$ on $\mathcal{C}_i$ and hashing/projected keys [b] hk_i and hp_i associated to $\mathcal{C}_i$ (which will be associated to the future Com_i). It finally signs all the flows using SK_j in σ_j, and sends $(\mathcal{C}_j, \vec{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ to P_i;
- Third Round: user P_i first checks the signature σ_j, computes $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}_i'^{\vec{\varepsilon}}$ and witness $\mathbf{z}$ (from $\vec{\varepsilon}$ and ω_i), it generates hashing/projected keys hk_j and hp_j associated to Com_j. It finally signs all the flows using SK_i in σ_i, and sends $(\mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ to P_j;
- Hashing: P_j first checks the signature σ_i and the correct opening of $\mathcal{C}''_i$ into $\mathcal{C}'_i$, it computes $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}_i'^{\vec{\varepsilon}}$.

  P_i computes K_i and P_j computes K_j as follows:

$$\begin{aligned} K_i &= \mathsf{Hash}(\mathsf{hk}_j, \{(\mathsf{priv}'_j, \mathsf{priv}_i)\} \times \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j), \ell_j, \mathsf{Com}_j) \\ &\quad \cdot \mathsf{ProjHash}(\mathsf{hp}_i, \{(\mathsf{priv}_i, \mathsf{priv}'_j)\} \times \mathcal{L}(\mathsf{pub}, \mathsf{priv}_i), \ell_i, \mathsf{Com}_i; \mathbf{z}) \\ K_j &= \mathsf{ProjHash}(\mathsf{hp}_j, \{(\mathsf{priv}_j, \mathsf{priv}'_i)\} \times \mathcal{L}(\mathsf{pub}, \mathsf{priv}_j), \ell_j, \mathsf{Com}_j; \vec{r}) \\ &\quad \cdot \mathsf{Hash}(\mathsf{hk}_i, \{(\mathsf{priv}'_i, \mathsf{priv}_j)\} \times \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_i), \ell_i, \mathsf{Com}_i) \end{aligned}$$

[a] The languages considered depend on two possibly different relations, namely $\mathcal{L}_i = \mathcal{L}_{\mathcal{R}_i}(\mathsf{pub}, \mathsf{priv}_i)$ and $\mathcal{L}_j = \mathcal{L}_{\mathcal{R}_j}(\mathsf{pub}, \mathsf{priv}_j)$, but we omit them for the sake of clarity.

[b] Recall that the SPHF is constructed in such a way that this projected key does not depend on $\mathcal{C}'_i$ and is indeed associated to the future whole Com_i.

Figure 6.4: Language-based Authenticated Key Exchange from a Smooth Projective Hash Function on Commitments

For the sake of simplicity, we give in Figure 6.5 an explicit version of the protocol described in Figure 6.4. Recall that Com_i is the combination $\mathcal{C}_i \cdot \mathcal{C}_i'^{\vec{\varepsilon}}$ of $\mathcal{C}_i$ and $\mathcal{C}'_i$ when the challenge $\vec{\varepsilon}$ is known, while $\mathcal{C}''_i$ is the Pedersen commitment of $\mathcal{C}'_i$ with randomness t, and $\mathsf{Com}_j = \mathcal{C}_j$. We omit the additional verification that all the committed values are in the correct subsets $\mathcal{P}$ and $\mathcal{S}$, since in the proof below we will always easily guarantee this membership. The proof heavily relies on the properties of the commitments and smooth projective hash functions given in section 2.6.5, page 46 and 5.3, page 99.

6.3.4 Notations

The proof follows that of [CHK+05] and [ACP09], but with a different approach since we want to prove that the best attack the adversary can perform is to play as an honest player would do with a chosen credential $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ —when trying to impersonate P_i— or $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$ —when trying to impersonate P_j—. In order to prove Theorem 6.3.1, page 114, we need to construct, for any real-world adversary $\mathcal{A}$ (controlling some dishonest parties), an ideal-world adversary $\mathcal{S}$ (interacting with dummy parties and the split functionality $s\mathcal{F}_{\text{LAKE}}$) such that no environment $\mathcal{Z}$ can distinguish between an execution with $\mathcal{A}$ in the real world and $\mathcal{S}$ in the ideal world with non-negligible probability.

We do not formally define the functionality $s\mathcal{F}_{\text{LAKE}}$, but it is straightforward from the definition of $\mathcal{F}_{\text{LAKE}}$ (see [BCL+05]). In particular, we assume that at the beginning of the protocol, $\mathcal{S}$ receives from it the contribution pub_i of P_i to the public language pub as answer to the Init query sent by the environment on behalf of this player. After the preflows, it will receive the public language pub (as

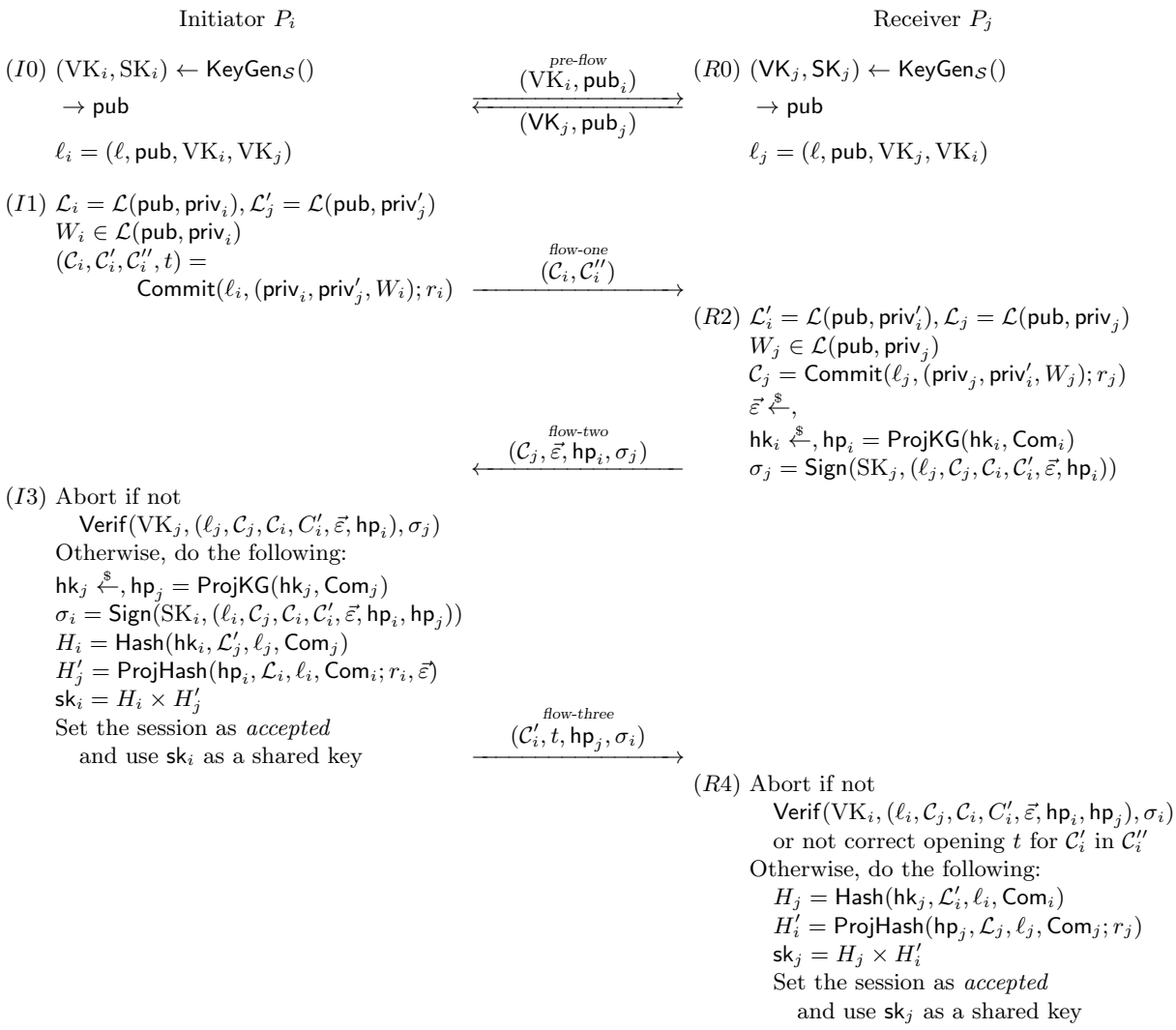


Figure 6.5: Description of the language authenticated key exchange protocol for players (P_i, ssid), with index i, message $W_i \in L_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_i)$ and expected language for P_j $L'_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j)$ and (P_j, ssid), with index j, message $W_j \in L_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_j)$ and expected language for P_i $L'_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_i)$. The label is $\ell = (\mathrm{sid}, \mathrm{ssid}, P_i, P_j)$. The random values used in the commitments (witnesses) are all included in r_i and r_j, and we note $\mathsf{Com}_j = \mathcal{C}_j$ and $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}_i^{\vec{\varepsilon}}$.

answer to the NewSession query transmitted by $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$, as seen on Figure 6.3) as determined during this preflow phase.

When initialized with security parameter $\mathfrak{K}$, the simulator begins by generating the CRS for the commitment (public parameters but also extraction and equivocation trapdoors), as well as the possibly required trapdoors to be able to generate, for any pub, a word in the language $\mathcal{L}(\mathsf{pub}, \mathsf{priv})$ when priv is known. It then initializes the real-world adversary $\mathcal{A}$, giving it these values. The simulator then starts its interaction with the environment $\mathcal{Z}$, the functionality $s\mathcal{F}_{\text{LAKE}}$ and its subroutine $\mathcal{A}$.

Since we are in the static-corruption model, the adversary can only corrupt players before the execution of the protocol. We assume players to be honest or not at the beginning, and they cannot be corrupted afterwards. However, this does not prevent the adversary from modifying flows coming from the players. Indeed, since we are in a weak authenticated setting, when a player acts dishonestly (even without being aware of it), it is either corrupted, hence the adversary knows its private values and acts on its behalf; or the adversary tries to impersonate it with chosen/guessed inputs. In both cases, we say the player is $\mathcal{A}$-controlled. Following [CHK+05], we say that a flow is *oracle-generated* if it was sent by an honest player and arrives without any alteration to the player it was meant to. We say it is *non-oracle-generated* otherwise, that is if it was sent by a $\mathcal{A}$-controlled player (which means corrupted, or which flows have been modified by the adversary). The one-time signatures are aimed at

avoiding changes of players during a session: if *pre-flow* is oracle-generated for P_i, then *flow-one* and *flow-three* cannot be non-oracle-generated without causing the protocol to fail because of the signature, for which the adversary does not know the signing key. Similarly, for P_j. On the other hand, if *pre-flow* is non-oracle-generated for P_i, then *flow-one* and *flow-three* cannot be oracle-generated without causing the protocol to fail, since the honest player would sign wrong flows (the flows the player sent before the adversary alters them). In both cases, the verifications of the signatures will fail at Steps ($I3$) or ($R4$) and P_i or P_j will abort. One can note that if there is one flow only in the protocol for one player, its signature is not required, which is the case for P_j when there is no pub to agree on at the beginning. But this is just an optimization that can be occasionally applied, as for the PAKE protocol. We do not consider it here.

To deal with both cases of $\mathcal{A}$-controlled players (either corrupted or impersonated by the adversary), we use the Split Functionality model. We thus add a *pre-flow* which will help us know which players are honest and which ones are $\mathcal{A}$-controlled. If one player is honest and the other one corrupted, the adversary will send the *pre-flow* on behalf of the latter, and the simulator will have to send the *pre-flow* on behalf of the former. But in the case where both players are honest at the beginning of the protocol, both *pre-flow* will have to be sent by $\mathcal{S}$ on behalf of these players and the adversary can then decide to modify one of these flows. This models the fact that the adversary can decide to split a session between P_i and P_j by answering itself to P_i, and thus trying to impersonate P_j with respect to P_i, and doing the same with P_j. Then, the Split Functionality model ensures that two independent sessions are created (with sub-session identifiers). We can thus study these sessions independently, which means that we can assume, right after the *pre-flow*, that either a player is honest if its *pre-flow* is oracle-generated, or $\mathcal{A}$-controlled if the *pre-flow* is non-oracle-generated. Since we want to show that the best possible attack for the adversary (by controlling a player) consists in playing honestly with a trial credential, we have to show that the view of the environment is unchanged if we simulate this dishonest player as an honest player. The simulator then has to transform its flows into queries to the Ideal Functionality $s\mathcal{F}_{\text{LAKE}}$, and namely the $\mathsf{NewSession}$-query. Still, the $\mathcal{A}$-controlled player is not honest, and can have a bad behavior when sending the real-life flows, but then either it has no strong impact, and it is similar to an honest behavior, or it will make the protocol to fail: we cannot avoid the adversary to make denial of service attack, and the adversary will learn nothing.

As explained in [BCL$^+$05] and [ACGP11], where the simulator actually had access to a $\mathsf{TestPwd}$ query to the functionality, it is equivalent to grant the adversary the right to test a password for P_i while trying to play on behalf of P_j (i.e. , use a $\mathsf{TestPwd}$ query) or to use the split functionality model and generate the $\mathsf{NewSession}$ queries corresponding to the $\mathcal{A}$-controlled players and see how the protocol terminates, since it corresponds to a trial of one credential by the adversary (one-line dictionary attack).

The proof will thus consist in generating ideal queries (and namely the $\mathsf{NewSession}$) when receiving non-oracle-generated flows from $\mathcal{A}$-controlled players, and generating real messages for the honest players (whose $\mathsf{NewSession}$ queries will be received from the environment). This will be done in a indistinguishable way for the environment.

We assume from now on that we know in which case we are, and the pub part is fixed. We then describe the simulator for each of these cases, while it has generated the *pre-flow* for the honest players by generating $(\text{VK}, \text{SK}) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}()$, and thus knows the signing keys. We denote by $\mathcal{L}_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_i)$ the language used by P_i, and by $L'_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j)$ the language that P_i expects P_j to use. We use the same notations in the reverse direction. The languages considered depend on two possibly different relations: $\mathcal{L}_i = \mathcal{L}_{\mathcal{R}_i}(\mathsf{pub}, \mathsf{priv}_i)$ and $\mathcal{L}_j = \mathcal{L}_{\mathcal{R}_j}(\mathsf{pub}, \mathsf{priv}_j)$, but we omit them for the sake of clarity. Note that the simulator will use the NewKey query to learn whether the protocol is a success or a failure. This will enable it to check whether the LAKE should fulfill, that is, whether the two users players compatible words and languages, i.e. . $\mathsf{priv}'_i = \mathsf{priv}_i$, $\mathsf{priv}'_j = \mathsf{priv}_j$, $W_i \in \mathcal{L}_i$ and $W_j \in \mathcal{L}_j$. For the most part, the interaction is implemented by the simulator $\mathcal{S}$ just following the protocol on behalf of all the honest players.

6.3.5 Description of the Simulators

Initialization and Simulation of *pre-flow*

This is the beginning of the simulation of the protocol, where $\mathcal{S}$ has to send the message *pre-flow* on behalf of each non-corrupted player.

Step ($I0$). When receiving the $(\mathsf{Init} : \text{ssid}, P_i, P_j, \mathsf{pub}_i, \mathsf{initiator})$ from $s\mathcal{F}_{\text{LAKE}}$ as answer to the Init query sent by the environment on behalf of P_i, $\mathcal{S}$ starts simulating the new session of the protocol for party

P_i, peer P_j, session identifier ssid. $\mathcal{S}$ chooses a key pair $(\mathrm{SK}_i, \mathrm{VK}_i)$ for a one-time signature scheme and generates a *pre-flow* message with the values $(\mathrm{VK}_i, \mathsf{pub}_i)$. It gives this message to $\mathcal{A}$ on behalf of (P_i, ssid).

Step ($R0$). When receiving the $(\mathsf{Init} : \mathrm{ssid}, P_j, P_i, \mathsf{pub}_j, \mathsf{receiver})$ from $s\mathcal{F}_{\mathrm{LAKE}}$ as answer to the Init query sent by the environment on behalf of P_j, $\mathcal{S}$ starts simulating the new session of the protocol for party P_j, peer P_i, session identifier ssid. $\mathcal{S}$ chooses a key pair $(\mathrm{SK}_j, \mathrm{VK}_j)$ for a one-time signature scheme and generates a *pre-flow* message with the values $(\mathrm{VK}_j, \mathsf{pub}_j)$. It gives this message to $\mathcal{A}$ on behalf of (P_j, ssid).

Splitting the Players

As just said, thanks to the Split Functionality model, according to which flows were transmitted or altered by $\mathcal{A}$, we know from the *pre-flow* which player(s) is (are) honest and which player(s) is (are) $\mathcal{A}$-controlled, and the public part pub. We can consider each case independently after the initial split, during which $\mathcal{S}$ generated the signing keys of the honest players. Thanks to the signature in the last flows for each player, if the adversary tries to take control on behalf of a honest user for some part of the execution (without learning the internal states, since we exclude adaptive corruptions), the verification will fail. Then we can assume that the sent flows are the received flows.

One can note that the prior agreement on pub allows to simulate P_i before any information from P_j and also whether it should be a success or not. Without such an agreement, the simulator would not know which value to use for pub whereas it cannot change its mind later, since it is sent in clear. Everything else is committed: either in an equivocable way on behalf of P_i so that we can change it later when we know the real status of the session; or in a non-equivocable way on behalf of P_j since we can check the status of the session before making this commitment. Of course, both commitments are extractable.

We come back again to the case of our equivocable commitment with SPHF that is not a really extractable/binding commitment since the player can open it in a different way one would extract it: if extraction leads to an inconsistent tuple, there is little change that with the random $\vec{\varepsilon}$ it becomes consistent; if extraction leads to a consistent tuple, there is little change that with the random $\vec{\varepsilon}$ it remains consistent, and then the real-life protocol will fail, whereas the ideal-one was successful at the NewKey-time. But then, because of the positive NewKey-answer, the $\mathsf{SendKey}$-query takes the key-input into consideration, that is random on the initiator side because of the SPHF on an invalid word, and thus indistinguishable from the environment point of view from a failed session: this is a denial of service of which the adversary is aware.

Hence, the three simulations presented below exploit the properties of our commitments and SPHF to make the view of the environment indistinguishable from a real-life attack, just using the simulator $\mathcal{S}$ that is allowed to interact with the ideal functionality on behalf of players, but in an honest way only, since the functionality is perfect and does not know bad behavior.

Case 1: P_i is $\mathcal{A}$-controlled and P_j is honest

In this case, $\mathcal{S}$ has to simulate the concrete messages in the real-life from the honest player P_j, for which it has simulated the *pre-flow* and thus knows the signing key, and has to simulate the queries to the functionality as if the $\mathcal{A}$-controlled player P_i was honest.

Step ($I1$). This step is taken care of by the adversary, who sends its *flow-one*, from which $\mathcal{S}$ extracts $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$. $\mathcal{S}$ then sends the query $(\mathsf{NewSession} : \mathrm{ssid}', P_i, P_j, W_i, \mathcal{L}_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_i), \mathcal{L}'_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j))$ to $\mathcal{F}_{\mathrm{LAKE}}$ on behalf of P_i.

Step ($R2$). The $\mathsf{NewSession}$ query for this player (P_j, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\mathrm{LAKE}}$ to $\mathcal{F}_{\mathrm{LAKE}}$ (transforming the session identifier from ssid to ssid$'$). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \mathrm{ssid}, P_j, P_i, \mathsf{pub}, \mathsf{receiver})$ and makes a call NewKey to the functionality to check the success of the protocol. In case of a success, $\mathcal{S}$ generates a word $W_j \in \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j)$ and uses $\mathsf{priv}_j = \mathsf{priv}'_j$ and $\mathsf{priv}'_i = \mathsf{priv}_i$ for this receiver session (we have assumed the existence of a trapdoor in the CRS to generate such valid words) and produces a commitment $\mathcal{C}_j$ on the tuple $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$. Otherwise, $\mathcal{S}$ produces a commitment $\mathcal{C}_j$ on a tuple $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$, where (priv_j, W_j) is consistent with pub, and priv'_i is a dummy value in $\mathcal{P}_i$.

It then generates a challenge value $\vec{\varepsilon}$ and the hash keys $(\mathsf{hk}_i, \mathsf{hp}_i)$ on $\mathcal{C}_i$. It sends the *flow-two* message $(C_j, \vec{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ to $\mathcal{A}$ on behalf of P_j, where σ_j is the signature on all the previous information.

STEP ($I3$). This step is taken care of by the adversary, who sends its *flow-three*.

STEP ($R4$). Upon receiving $m = (\textit{flow-three}, C'_i, t, \mathsf{hp}_j, \sigma_i)$, $\mathcal{S}$ makes the verification checks, and possibly aborts. In case of correct checks, $\mathcal{S}$ already knows whether the protocol should succeed, thanks to the NewKey query. If the protocol is a success, then $\mathcal{S}$ computes receiver session key honestly, and makes a SendKey to P_j. Otherwise, $\mathcal{S}$ makes a SendKey to P_j with a random key that will anyway not be used.

Case 2: P_i is honest and P_j is $\mathcal{A}$-controlled

In this case, $\mathcal{S}$ has to simulate the concrete messages in the real-life from the honest player P_i, for which it has simulated the *pre-flow* and thus knows the signing key, and has to simulate the queries to the functionality as if the $\mathcal{A}$-controlled player P_j was honest.

STEP ($I1$). The NewSession query for this player (P_i, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$ (transforming the session identifier from ssid to ssid$'$). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \text{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{initiator})$ and generates a *flow-one* message by committing to the tuple $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ where (priv_i, W_i) is a pair consistent with pub, and priv'_j is a dummy parameter in $\mathcal{P}_j$. It gives this commitment $(\mathcal{C}_i, \mathcal{C}''_i)$ to $\mathcal{A}$ on behalf of (P_i, ssid').

STEP ($R2$). This step is taken care of by the adversary, who sends its $\textit{flow-two} = (\textit{flow-two}, \mathcal{C}_j, \vec{\varepsilon}, \mathsf{hp}_i, \sigma_i)$, from which $\mathcal{S}$ first checks the signature, and thereafter extracts the committed triple $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$. $\mathcal{S}$ then sends the query $(\mathsf{NewSession} : \text{ssid}', P_j, P_i, W_j, \mathcal{L}_j = \mathcal{L}(\mathsf{pub}, \mathsf{priv}_j), \mathcal{L}'_i = \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_i))$ to $\mathcal{F}_{\text{LAKE}}$ on behalf of P_j.

STEP ($I3$). $\mathcal{S}$ makes a NewKey query to the functionality to know whether the protocol should succeed. In case of a success, $\mathcal{S}$ generates a word $W_i \in \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_i)$ and uses $\mathsf{priv}_i = \mathsf{priv}'_i$ for this initiator session (we have assumed the existence of a trapdoor in the CRS to generate such valid words) and then uses the equivocability trapdoor to update $\mathcal{C}'_i$ and t in order to contain the new consistent tuple $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ with respect to the challenge $\vec{\varepsilon}$. If the protocol should be a success, then $\mathcal{S}$ computes initiator session key honestly, and makes a SendKey to P_i. Otherwise, $\mathcal{S}$ makes a SendKey to P_i with a random key that will anyway not be used.

$\mathcal{S}$ sends the *flow-three* message $(\mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ to $\mathcal{A}$ on behalf of P_i, where σ_i is the signature on all the previous information.

STEP ($R4$). This step is taken care of by the adversary.

Case 3: P_i and P_j are honest

In this case, $\mathcal{S}$ has to simulate the concrete messages in the real-life from the two honest players P_i and P_j, for which it has simulated the *pre-flow* and thus knows the signing keys.

STEP ($I1$). The NewSession query for this player (P_i, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$ (transforming the session identifier from ssid to ssid$'$). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \text{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{initiator})$ and generates a *flow-one* message by committing to the tuple $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ where (priv_i, W_i) is a pair consistent with pub, and priv'_j is a dummy parameter in $\mathcal{P}_j$. It gives this commitment $(\mathcal{C}_i, \mathcal{C}''_i)$ to $\mathcal{A}$ on behalf of (P_i, ssid').

STEP ($R2$). The NewSession query for this player (P_i, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$ (transforming the session identifier from ssid to ssid$'$). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \text{ssid}, P_j, P_i, \mathsf{pub}, \mathsf{receiver})$ and makes a call NewKey to the functionality to check the success of the protocol. In case of a success, $\mathcal{S}$ generates a word $W_j \in \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_j)$ and uses $\mathsf{priv}_j = \mathsf{priv}'_j$ and $\mathsf{priv}'_i = \mathsf{priv}_i$ for this receiver session (we have assumed the existence of a trapdoor in the CRS to generate such valid words) and produces a commitment $\mathcal{C}_j$ on the tuple $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$. Otherwise, $\mathcal{S}$ produces a commitment $\mathcal{C}_j$ on a tuple $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$, where (priv_j, W_j) is consistent with pub, and priv'_i is a dummy value in $\mathcal{P}_i$.

It then generates a challenge value $\vec{\varepsilon}$ and the hash keys $(\mathsf{hk}_i, \mathsf{hp}_i)$ on $\mathcal{C}_i$. It sends the *flow-two* message $(C_j, \vec{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ to $\mathcal{A}$ on behalf of P_j, where σ_j is the signature on all the previous information.

STEP ($I3$). When the session $(P_i; \text{ssid}')$ receives the message $m = (\textit{flow-two}, \mathcal{C}_j, \vec{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ from its peer session $(P_j; \text{ssid}')$, the signature is necessarily correct. If the session should be a success (answer of the previous NewKey-query), $\mathcal{S}$ updates its commitment with a word $W_i \in \mathcal{L}(\mathsf{pub}, \mathsf{priv}'_i)$ and uses $\mathsf{priv}_i = \mathsf{priv}'_i$ for this initiator session (we have assumed the existence of a trapdoor in the CRS to generate such valid words) and then uses the equivocability trapdoor to update $\mathcal{C}'_i$ and t in order to contain the new consistent

tuple $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ with respect to the challenge $\vec{\varepsilon}$. Otherwise, it does not change anything about this commitment and uses the initial value that is likely inconsistent.

In any case, $\mathcal{S}$ makes a $\mathsf{SendKey}$ to P_i with a random key that will anyway not be used, since no player is corrupted.

Step ($R4$). When the session $(P_j; \mathrm{ssid}')$ receives the message $m = (\textit{flow-three}, \mathsf{hp}_j, C'_i, \sigma_i)$ from its peer session $(P_i; \mathrm{ssid}')$, the signature will necessarily correct. $\mathcal{S}$ makes a $\mathsf{SendKey}$ to P_j with a random key that will anyway not be used, since no player is corrupted.

6.4 Efficient Instantiation of AKE protocols

In this section, we first recall two useful languages, and then give some concrete instantiations of several AKE protocols, using our generic protocol of LAKE, and compare their efficiencies to the existing instantiations.

6.4.1 Useful Languages

This section recalls two relations to be thereafter used in concrete instantiations. The same way as before, we will underline elements the prover wants to keep private, and that will be committed either with the LCSCom non-equivocable commitment or the $\mathsf{DLCSCom}'$ equivocable commitment. As equality tests and linear pairing product equations have been detailed earlier throughout chapter 5, page 93, we are mainly going to explain here how we can additionally restrict the space of the committed values.

Bit Commitment: $\underline{m} \in \{0, 1\}$**.** Player P_i owns a word W_i which is either 0 or 1, and wants to prove it. The apparent problem with disjunction of languages from [ACP09] comes from the fact the projection key relies on the final commitment. This makes no difference for the LCSCom but can be problematic in the case of the $\mathsf{DLCSCom}'$. While in a regular context this does not create extra problems, this contradicts the creation of our projection key before the end of the equivocable commitment (in the third round of the protocol). But in our case, the verifier will build the projection key on the sole pre-commit $\mathcal{C}$, and we make sure that in our simulation for the proof the simulator always pre-commits to $m_i = 0$: we do not need the equivocability on that property for the commitment. However, this bit-language will often be in combination with some other languages (like "a valid signature"): the computation of the hash values for the bit-language will be on the commitment $\mathcal{C}$ only, while the contribution for the other languages will be on the final commitment (and the hash values will be multiplied).

Finite Subset: $\underline{m} \in \mathcal{S} = \{A, B, \ldots\}$**.** Player P_i owns a word W_i in a (polynomial-sized) set $\mathcal{S}$. This follows the same idea as before, except that in this case, the simulator in the proof will always pre-commit to A, which does not require equivocability on the $\mathcal{C}$ part of the commitment since it is in this language. Equivocation can be required to make it in a more specific language, but then this will be checked with another SPHF on the full commitment.

6.4.2 Password Authenticated Key Exchange

Using our generic construction, we can easily obtain a PAKE protocol, as described on Figure 6.6, where we optimize from the generic construction, since $\mathsf{pub} = \emptyset$, removing the agreement on pub, but still keeping the signing key VK_i to avoid man-in-the-middle attacks since it has another later flow: P_i uses a password W_i and expects P_j to own the same word, and thus in the language $\mathcal{L}'_j = \mathcal{L}_i = \{W_i\}$; P_j uses a password W_j and expects P_i to own the same word, and thus in the language $\mathcal{L}'_i = \mathcal{L}_j = \{W_j\}$; The relation is the equality test between priv_i and priv_j, which have no restriction in $\mathbb{G}$ (hence $\mathcal{P} = \mathbb{G}$). As the word W_i, the language private parameters priv_i of a user and priv'_j of the expected language for the other user are the same, each user can commit in the protocol to only one value: its password W_i.

It is quite efficient, as discussed in the next session, and relies on the DLin assumption. We can of course instantiate it with the Cramer-Shoup variant, under the DDH assumption, and it becomes even more efficient.

6.4.3 Verifier-based PAKE

The above scheme can be modified into an efficient PAKE protocol that is additionally secure against *server compromise*: the also so-called verifier-based PAKE, where the client owns a password pw, while the server knows a verifier only, such as g^{pw}, so that in case of break-in to the server, the adversary will not immediately get all the passwords.

P_i uses a password W_i and P_j uses a password W_j. We denote $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.

- First Round: P_i (with random tape ω_i) first generates a pair of signing/verification keys $(\mathsf{SK}_i, \mathsf{VK}_i)$ and a $\mathsf{DLCSCom}'$ commitment on W_i in $(\mathcal{C}_i, \mathcal{C}'_i)$, under $\ell_i = (\ell, \mathsf{VK}_i)$. It also computes a Pedersen commitment on $\mathcal{C}'_i$ in $\mathcal{C}''_i$ (with random exponent t). It then sends $(\mathsf{VK}_i, \mathcal{C}_i, \mathcal{C}''_i)$ to P_j;
- Second Round: P_j (with random tape ω_j) computes a LCSCom commitment on W_j in $\mathsf{Com}_j = \mathcal{C}_j$, with witness $\vec{r}$, under the label ℓ. It then generates a challenge ε on $\mathcal{C}_i$ and hashing/projected keys hk_i and the corresponding hp_i for the equality test on Com_i ("Com_i is a valid commitment of W_j", this only requires the value ξ_i computable thanks to $\mathcal{C}_i$). It then sends $(\mathcal{C}_j, \varepsilon, \mathsf{hp}_i)$ to P_i;
- Third Round: user P_i can compute $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}'^{\varepsilon}_i$ and witness $\mathbf{z}$ (from ε and ω_i), it generates hashing/projected keys hk_j and hp_j for the equality test on Com_j. It finally signs all the flows using SK_i in σ_i and send $(\mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ to P_j;
- Hashing: P_j first checks the signature and the validity of the Pedersen commitment (thanks to t), it computes $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}'^{\varepsilon}_i$. P_i computes K_i and P_j computes K_j as follows:

$$\begin{aligned} K_i &= \mathsf{Hash}(\mathsf{hk}_j, \mathcal{L}'_j, \ell, \mathsf{Com}_j) \cdot \mathsf{ProjHash}(\mathsf{hp}_i, \mathcal{L}_i, \ell_i, \mathsf{Com}_i; \mathbf{z}) \\ K_j &= \mathsf{ProjHash}(\mathsf{hp}_j, \mathcal{L}_j, \ell, \mathsf{Com}_j; \vec{r}) \cdot \mathsf{Hash}(\mathsf{hk}_i, \mathcal{L}'_i, \ell_i, \mathsf{Com}_i) \end{aligned}$$

Figure 6.6: Password-based Authenticated Key Exchange

To this aim, as usually done, one first does a PAKE with g^{pw} as common password, then asks the client to additionally prove it can compute the Diffie-Hellman value h^{pw} for a basis h chosen by the server. Ideally, we could implement this trick, where the client P_j just considers the equality test between the g^{pw} and the value committed by the server for the language $\mathcal{L}'_i = \mathcal{L}_j$, while the server P_i considers the equality test with $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$, where h is sent as its contribution to the public part of the language by the server $\mathcal{L}_i = \mathcal{L}'_j$. Since the server chooses h itself, it chooses it as $h = g^{\alpha}$, for an ephemeral random α, and can thus compute $h^{\mathsf{pw}} = (g^{\mathsf{pw}})^{\alpha}$. On its side, the client can compute this value since it knows pw. The client could thus commit to $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$, in order to prove its knowledge of pw, whereas the server could just commit to g^{pw}. Unfortunately, from the extractability of the server commitment, one would just get g^{pw}, which is not enough to simulate the client.

To make it in a provable way, the server chooses an ephemeral h as above, and they both run the previous PAKE protocol with $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$ as common password, and mutually checked: h is seen as a the pub part, hence the preliminary flows are required.

Credential-Authenticated Key Exchange

In [CCGS10], the authors proposed instantiations of the CAKE primitive for conjunctions[4] of atomic policies that are defined algebraically by relations of the form $\prod_{j=1}^{k} g_j^{F_j} = 1$ where the g_j's are elements of an abelian group and F_j's are integer polynomials in the variables committed by the users. We can construct LAKE for languages with the same expressibility using the constructions described in chapter II, page 92. The resulting schemes are faster and more bandwidth-efficient than the proposals from [CCGS10]. Our schemes require two interleaved executions of the commitment scheme where the CAKE schemes need at least three.

In [CCGS10], the core of their constructions relies on their practical UC zero-knowledge proof. There is no precise instantiation of such proof, but it is very likely to be inefficient. Their proof technique indeed requires to transform the underlying Σ-protocols into corresponding Ω-protocols [GMY06] by verifiably encrypting the witness. An Ω-protocol is a Σ-protocol with the additional property that it admits a polynomial-time straight-line extractor. Since the witnesses are scalars in their algebraic relations, their approach requires either inefficient bit-per-bit encryption of these witnesses or Paillier encryption in which case the problem of using group with different orders in the representation and in the encryption requires additional overhead.

Even when used with Σ-protocols, their PAKE scheme without UC-security, requires at least two

[4] Camenisch *et al.* [CCGS10] claim that their CAKE protocol can also handle disjunctions of such languages but the technique relies on known techniques for circuit evaluation (based on homomorphic encryption) and are even less efficient.

proofs of knowledge of representations that involve at least 30 group elements (if we assume the encryption to be linear Cramer-Shoup), and some extra for the last proof of existence (*cf.* [CKS11]), where our PAKE requires at most 22 group elements (see next section). Anyway they say, their PAKE scheme is less efficient than [CHK+05], which needed 6 rounds and around 30 modular exponentiations per user, while our efficient PAKE requires overall 42 exponentiations in only 3 rounds. Our scheme is therefore more efficient than the scheme from [CHK+05] for the same security level (*i.e.* UC-security with static corruptions).

Secret-handshakes

We can also instantiate a (linkable) Secret Handshakes protocol, using our scheme with two different languages: P_i will commit to a valid signature σ_i on a message m_i (his identity for example), under a private verification key vk_i, and expects P_j to commit to a valid signature on a message m'_j under a private verification key vk'_j; but P_j will do analogously with a signature σ_j on m_j under vk_j, while expecting a signature on m'_i under vk'_i. The public parts of the signature (the second component) are sent in clear with the commitments.

In a regular Secret Handshakes both users should use the same languages. But here, we have a more general situation[5]: the two participants will have the same final value if and only if they both belong to the organization the other expects. If one lies, our protocol guarantees no information leakage. Furthermore, the semantic security of the session is even guaranteed with respect to the authorities, in a forward-secure way (this property is also achieved in [JL09] but in a weaker security model). Finally, our scheme supports revocation and can handle roles as in [AKB07].

Standard secret handshakes, like [AKB07], usually work with credentials delivered by a unique authority, this would remove our need for a hidden verification key, and private part of the language. Both users would only need to commit to signatures on their identity/credential, and show that they are valid. This would require 24 group elements with our approach. Their construction requires only 4 elements under BDH, however it relies on the asymmetric Waters IBE with only two elements, whereas the only security proof known for such IBE [Duc10] requires an extra term in $\mathbb{G}_2$ which would render their technique far less efficient, as several extra terms would be needed to expect a provably secure scheme. While sometimes less effective, our LAKE approach can manage Secret Handshakes, and provide additional functionalities, like more granular control on the credentials as part of them can be expressly hidden by both the users.

Unlinkable Secret-handshakes

Moving the users' identity from the public pub part to individual private priv part or even to the word W, and combining our technique with [BPV12b], it is also possible to design an *unlinkable* Secret Handshakes protocol [JL09] with practical efficiency. It illustrates the case where committed values have to be proven in a strict subset of $\mathbb{G}$, as one has to commit to bits: the signed message M is now committed and not in clear, it thus has to be done bit-by-bit since the encoding $\mathcal{G}$ does not allow algebraic operations with the content to apply the Waters function on the message. It is thus possible to prove the knowledge of a Waters signature on a private message (identity) valid under a private verification key. Additional relations can be required on the latter to make authentication even stronger.

6.4.4 Complexity

In the Table 6.1, we give the number of elements to be sent (group elements or scalars) and the number of exponentiations required for each operation (commitment and SPHF), where we consider the Equality Test, the Linear Pairing Product Equations and the OR languages. One has to commit all the private inputs, and then the cost for relations is just the additional overhead due to the projected keys and hashing computations once the elements are already committed: an LCSCom commitment is 5 group elements, and a $\mathsf{DLCSCom}'$ is twice more, plus the Pedersen commitment, the challenge ε and the opening t, and thus 13 elements. If the global language is a conjunction of several languages, one should simply add all the costs, and consider the product of all the sub-hashes as the final hash value from the SPHF.

[5] [AKB07] call it a *dynamic matching*

DLin	$\mathbb{G}$	$\mathbb{Z}_p$	Exp.
LCSCom	$5\,n$	0	$7\,n+2$
DLCSCom	$10\,n+1$	2	$18\,n+6$
Equality	2	0	14
LPPE	$2\,n+1$	0	$10\,n+11$
OR	1	0	12

CSCom	$\mathbb{G}$	$\mathbb{Z}_p$	Exp.
CSCom	$4\,n$	0	$4\,n+1$
DCSCom	$8\,n+1$	2	$12\,n+5$
Equality	1	0	10
LPPE	$n+1$	0	$7\,n+9$
OR	1	0	9

Table 6.1: Computational and Communication Costs

PAKE

Two users want to prove to each other they possess the same password. In this case $W_i = \mathsf{priv}'_j = \mathsf{priv}_i = \mathsf{priv}_j = \mathsf{priv}'_i = W_j$. So P_i will commit to his password, and thus a unique DLCSCom commitment for W_i, priv_i and priv'_i. This is the same for P_j, but with an LCSCom. They then send projected keys for equality tests: 13 group elements and 2 scalars for Com_i and 7 group elements for Com_j, plus VK_i and sk_i. This leads to 20 group elements and two scalars our PAKE scheme. The DDH-based variant would use 15 group elements and 2 scalars only in total, which is far more efficient than existing solutions, and namely [ACP09] that uses a bit-per-bit commitment to provide equivocability.

CRS: a group $\mathbb{G}$ of prime order p, with six independent generators $(g_1, g_2, h, c, d, \zeta) \xleftarrow{\$} \mathbb{G}^6$, a collision-resistant hash function $\mathfrak{H}_K$, a reversible mapping $\mathcal{G}$ from $\{0,1\}^n$ to $\mathbb{G}$, a signature scheme $(\mathsf{KeyGen}_{\mathcal{S}}, \mathsf{Sign}, \mathsf{Verif})$. We note $M = \mathcal{G}(m)$, and $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.

$(\mathsf{SK}_i, \mathsf{VK}_i) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}()$
$\ell_i = (\ell, \mathsf{VK}_i)$
$a_i, r_i \xleftarrow{\$} \mathbb{Z}_p$
$(\mathcal{C}_i, \mathcal{C}'_i) = \mathsf{DCSCom}(\ell_i, M_i, 1_{\mathbb{G}}; r_i, a_i)$,
$t_i \xleftarrow{\$} \mathbb{Z}_p, \chi_i = \mathfrak{H}_K(\mathcal{C}_i, \mathcal{C}'_i), \mathcal{C}''_i = g_1^{t_i}\zeta^{\chi_i}$,

$\xrightarrow{\mathsf{VK}_i, \mathcal{C}_i, \mathcal{C}''_i}$

$r_j \xleftarrow{\$} \mathbb{Z}_p$
$\mathcal{C}_j = \mathsf{CS}(\ell, M_j; r_j)$,
$\xi_i = \mathfrak{H}_K(\ell_i, \vec{u_i}, e_i)$,
$\mathsf{hk}_i = (\eta_i, \theta_i, \lambda_i, \mu_i) \xleftarrow{\$} \mathbb{Z}_p^4$,
$\mathsf{hp}_i = h^{\lambda_i} g_1^{\eta_i} g_2^{\theta_i} (cd^{\xi_i})^{\mu_i}$

$\varepsilon \xleftarrow{\$} \mathbb{Z}_p^*$

$\xleftarrow{\mathcal{C}_j, \varepsilon, \mathsf{hp}_i}$

$\varepsilon \overset{?}{\neq} 0, z_i = r_i + \varepsilon a_i$,
$\xi_j = \mathfrak{H}_K(\ell, \vec{u_j}, e_j)$,
$\mathsf{hk}_j = (\eta_j, \theta_j, \lambda_j, \mu_j) \xleftarrow{\$} \mathbb{Z}_p^4$,
$\mathsf{hp}_j = h^{\lambda_j} g_1^{\eta_j} g_2^{\theta_j} (cd^{\xi_j})^{\mu_j}$,
$H'_i = \mathsf{hp}_i^{z_i}$
$H_j = \mathsf{Hash}(\mathsf{hk}_j, \mathcal{L}_{M_i}, \ell'_j, \mathcal{C}_j)$
$K_i = H'_i \cdot H_j$
$\sigma_i = \mathsf{Sign}(\mathsf{SK}_i, (\ell_i, \ell, \mathcal{C}_i, \mathcal{C}'_i, \mathcal{C}_j, \varepsilon, \mathsf{hp}_i, \mathsf{hp}_j))$

$\xrightarrow{\mathsf{hp}_j, \mathcal{C}'_i, t_i, \sigma_i}$

$\chi_i = \mathfrak{H}_K(\mathcal{C}_i, \mathcal{C}'_i), \pi_i \overset{?}{=} g_1^{t_i}\zeta^{\chi_i}$
$H_i = \mathsf{Hash}(\mathsf{hk}_i, \mathcal{L}_{M_j}, \ell'_i, \mathcal{C}_i\mathcal{C}_i'^{\varepsilon_i})$
$H'_j = \mathsf{hp}_j^{r_j} K_j = H_i \cdot H'_j$,

If $\mathsf{Verif}(\mathsf{VK}_i, (\ell_i, \ell, \mathcal{C}_i, \mathcal{C}'_i, \mathcal{C}_j, \varepsilon, \mathsf{hp}_i, \mathsf{hp}_j), S_i)$ sets the session as *accepted*

Figure 6.7: Password Authenticated Key Exchange based on Cramer-Shoup

Verifier-based PAKE

As explained earlier, we do a PAKE with the common password $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$, where h has been chosen by the server: the commitment Com_i needs 21 group elements plus 2 scalars, and 4 additional group elements to check it; the commitment Com_j needs 10 group elements, and 4 additional elements to check it. Because of the ephemeral h, one has to send in total 40 group elements and 2 scalars, plus the one-time signatures. The DDH-based variant would use 29 group elements and 2 scalars only in total.

Secret Handshake

The users want to check their partner possesses a valid signature on their public identity or pseudonym (in pub) under some valid but private verification key. More precisely, P_i wants to prove he possesses a valid signature σ on the public message m (his identity or a pseudonym) under a private verification key vk: we thus have m in the pub part, $\mathsf{priv}_i = \mathsf{vk}$ and $W = \sigma$. This is the same for P_j. Using Waters

CRS: a group $\mathbb{G}$ of prime order p, with eleven independent generators $(g, g_1, g_2, g_3, h_1, h_2, c_1, c_2, d_1, d_2, \zeta) \xleftarrow{\$} \mathbb{G}^{11}$, $n+1$ independent generators $f_i \xleftarrow{\$} \mathbb{G}^{n+1}$ for the Waters function $\mathcal{F}$, an extra generator $h = g^s$, (which defines signing/verification keys $(\mathsf{sk}, \mathsf{vk}) = (h^x, g^x)$), a collision-resistant hash function $\mathfrak{H}_K$, a reversible mapping $\mathcal{G}$ from $\{0,1\}^n$ to $\mathbb{G}$, and a signature scheme S. In the following we note $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ and $\mathfrak{C}_* = \mathcal{C}_* \mathcal{C}_*'^{\varepsilon}$. We assume each user possesses a valid signature Σ on their identity valid under vk_*. The language $\mathcal{L}_*$ is composed of valid signatures Σ_* under the private $\mathfrak{p}_*$ verification key vk_*.

$(\mathsf{SK}_i, \mathsf{VK}_i) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}()$

$\xleftarrow[\mathsf{VK}_j, \mathsf{pub}_j]{\mathsf{VK}_i, \mathsf{pub}_i}$

$(\mathsf{SK}_j, \mathsf{VK}_j) \leftarrow \mathsf{KeyGen}_{\mathcal{S}}()$

$\ell_i' = (\ell, \mathsf{pub}, \mathsf{VK}_i)$ $\quad$ $\ell_j' = (\ell, \mathsf{pub}, \mathsf{VK}_j)$

$\omega_i = (\vec{a}_i, \vec{b}_i, \vec{r}_i, \vec{s}_i) \xleftarrow{\$} \mathbb{Z}_p^2, t_i \xleftarrow{\$} \mathbb{Z}_p$
$\mathfrak{L}_{\mathfrak{i}} = (\mathsf{vk}_i, \Sigma_i)$
$(\mathcal{C}_i, \mathcal{C}_i') = 2 - \mathsf{DLCSCom}(\ell_i', \mathfrak{L}_{\mathfrak{i}}, 1_{\mathbb{G}}^2; \omega_i)$
$\chi_i = \mathfrak{H}_K(\mathcal{C}_i, \mathcal{C}_i'), \mathcal{C}_i'' = g_1^{t_i} \zeta^{\chi_i}$

$\xrightarrow{\mathcal{C}_i, \mathcal{C}_i''}$

$\omega_j = (\vec{r}_j, \vec{s}_j) \xleftarrow{\$} \mathbb{Z}_p^2$
$\mathfrak{L}_{\mathfrak{j}} = (\mathsf{vk}_j, \Sigma_j)$
$\mathcal{C}_j = 2 - \mathsf{LCSCom}(\ell_j', \mathfrak{L}_{\mathfrak{j}}; \omega_j)$
$\mathsf{hk}_i \xleftarrow{\$} \mathbb{Z}_p^{5\times 2}$
$\mathsf{hp}_i = \mathsf{ProjKG}(\mathsf{hk}_i, (\mathfrak{p}'_i \wedge \mathcal{L}_i'), \mathcal{C}_i)$
$\varepsilon \xleftarrow{\$} \mathbb{Z}_p^*$
$\sigma_j = \mathsf{Sign}(\mathsf{SK}_j, (\ell_i', \ell_j', \mathcal{C}_i, \mathcal{C}_j, \varepsilon)$

$\xleftarrow{\mathcal{C}_j, \varepsilon, \sigma_i}$

$\vec{z}_i = (z_{i,r} = \vec{r}_i + \varepsilon \vec{a}_i, z_{i,s} = \vec{s} + \varepsilon \vec{b}_i)$
$\mathsf{hk}_j \xleftarrow{\$} \mathbb{Z}_p^{5\times 2}$
$\mathsf{hp}_j = \mathsf{ProjKG}(\mathsf{hk}_j, (\mathfrak{p}'_j \wedge \mathcal{L}_j'), \mathcal{C}_j)$
$\sigma_i = \mathsf{Sign}(\mathsf{SK}_i, (\ell_i', \ell_j', \mathcal{C}_i, \mathcal{C}_i'', \mathcal{C}_j, \varepsilon)$

$\xrightarrow{\mathsf{hp}_j, \mathcal{C}_i', t_i, \sigma_i}$

$H'_{\mathfrak{p}_{\mathfrak{i}}} = \mathsf{ProjHash}(\mathsf{hp}_{\mathfrak{p}_{\mathfrak{i}}}, \mathsf{vk}_i, \ell_i', \mathfrak{C}_{\mathfrak{p}_{\mathfrak{i}}}; \vec{z}_i)$
$H'_{\mathcal{L}_i} = \mathsf{ProjHash}(\mathsf{hp}_{\mathcal{L}_i}, \mathcal{L}_i, \ell_i', \mathfrak{C}_{\mathcal{L}_i}; \vec{z}_i)$
$H_{\mathfrak{p}_{\mathfrak{j}}} = \mathsf{Hash}(\mathsf{hk}_{\mathfrak{p}_{\mathfrak{j}}}, \mathsf{vk}_j', \ell_j', \mathcal{C}_{\mathfrak{p}_{\mathfrak{j}}})$
$H_{\mathcal{L}_j} = \mathsf{Hash}(\mathsf{hp}_{\mathcal{L}_j}, \mathcal{L}_j', \ell_j', \mathcal{C}_{\mathcal{L}_j})$
$H_j' = e(H'_{\mathfrak{p}_{\mathfrak{i}}}, g) \cdot H'_{\mathcal{L}_i}$
$H_i = e(H_{\mathfrak{p}_{\mathfrak{j}}}, g) \cdot H_{\mathcal{L}_j}$
$K_i = H_i \cdot H_j'$

If $\mathsf{Verif}(\mathsf{VK}_j, (\ell_i', \ell_j', \mathcal{C}_i, \mathcal{C}_j, \varepsilon), \sigma_j)$ sets the session as *accepted*.

$H_{\mathfrak{p}_{\mathfrak{i}}} = \mathsf{Hash}(\mathsf{hk}_{\mathfrak{p}_{\mathfrak{i}}}, \mathsf{vk}_i', \ell_i', \mathfrak{C}_{\mathfrak{p}_{\mathfrak{i}}})$
$H_{\mathcal{L}_i} = \mathsf{Hash}(\mathsf{hp}_{\mathcal{L}_i}, \mathcal{L}_i', \ell_i', \mathfrak{C}_{\mathcal{L}_i})$
$H'_{\mathfrak{p}_{\mathfrak{j}}} = \mathsf{ProjHash}(\mathsf{hp}_{\mathfrak{p}_{\mathfrak{j}}}, \mathsf{vk}_j, \ell_j', \mathcal{C}_{\mathfrak{p}_{\mathfrak{j}}}; \omega_j)$
$H'_{\mathcal{L}_j} = \mathsf{ProjHash}(\mathsf{hp}_{\mathcal{L}_j}, \mathcal{L}_j, \ell_j', \mathcal{C}_{\mathcal{L}_j}; \omega_j)$
$H_j = e(H_{\mathfrak{p}_{\mathfrak{i}}}, g) \cdot H_{\mathcal{L}_i}$
$H_i' = e(H'_{\mathfrak{p}_{\mathfrak{j}}}, g) \cdot H'_{\mathcal{L}_j}$
$K_j = H_i' \cdot H_j$

If $\mathsf{Verif}(\mathsf{VK}_i, (\ell_i', \ell_j', \mathcal{C}_i, \mathcal{C}_i', \mathcal{C}_j, \varepsilon), \sigma_i)$ sets the session as *accepted*.

Figure 6.8: Secret Handshake

signature, $\sigma = (\sigma_1, \sigma_2)$, where σ_1 only has to be encrypted, because σ_2 does not contain any information, it can thus be sent in clear. To achieve unlinkability, one can rerandomize this signature σ to make the σ_2 values different and independent each time.

As a consequence, the committed values are: vk that can be any group element, since with the master secret key s such that $h = g^s$ for the global parameters of the Waters signature one can derive the signing key associated to any verification key, and thus generate a valid word in the language; and σ_1. One additionally sends σ_2 in clear, and so 22 group elements plus 2 scalars for Com_i, and 11 group elements for Com_j. The languages to be verified are $\mathsf{priv}_i = \mathsf{priv}_i'$, on the committed $\mathsf{priv}_i = \mathsf{vk}_i$ with the expected $\mathsf{priv}_i' = \mathsf{vk}_i'$, and the Linear Pairing Product Equation for the committed signature σ_i, but under the expected vk_i': 5 group elements for the projected keys in both directions: 43 group elements plus 2 scalars are sent in total (plus the one-time signatures).

$\star$

Conclusion

This thesis can be divided into two main parts, in the first one we focused on how to use non-interactive zero-knowledge or witness-indistinguishable proofs to combine existing primitives to build efficient new protocols while in the second we built implicit proofs of knowledge.

In the first part, we followed the Groth-Sahai methodology for non-interactive proofs of knowledge which led us to two constructions, one allowing to build the first list signature in the standard model, and one to present a new primitive we called Signatures on Randomizable Ciphertexts.

We first began by enhancing group signatures, and produced new efficient schemes for *Traceable Signatures*, and *List Signatures*.

We then provided a new primitive allowing some commutative properties between signature and encryption where one can "decrypt" a signature on a ciphertext to obtain a signature on the associated plaintext. We called this primitive *Signature on Randomizable Ciphertexts*. We then instantiated our primitive under the standard DLin assumption and showed how to use it to propose the first efficient instantiation of Fischlin's round-optimal blind signatures which results in a standard signature. We also gave several other examples on how to use this primitive in other cases, like how to obtain perfectly blind signatures, or how to allow a part of the message to be public. Thanks to a side result on Waters Hash Function programmability, we even show how to use some malleability on the ciphertexts to blindly sign at once multiple messages coming from different sources.

Then we considered interactive protocols and tried to see if there was a way to exploit the interaction to produce better proofs. We used the *Smooth Projective Hash Functions* as a way to provide *Implicit Proofs of Knowledge*. We developed a complete methodology for these proofs to allow our protocols to handle many different new languages. We then followed it in various primitives, first *Oblivious Signature-Based Envelope* which allows a sender to send a message which can be read only if the recipient has the appropriate credentials, and then further developed the construction into *Language Authenticated Key Exchange*. This new primitive can supersede most of the existing AKE protocols by allowing two users to establish a shared key, only if both possessed the credentials (viewed as a word in a language) expected by the other, otherwise they learn nothing on the other credential. The two languages involved do not necessarily need to be the same as long as the other expects it.

We proved the security of this protocol in the UC framework with joint state and static corruptions. Once again we instantiated those primitives under classical assumption, DLin, and shows various applications that lead to a very-efficient *Password Authenticated Key Exchange protocol*, an efficient verifier-based PAKE so that a server corruption does not compromise the user's password directly. We also presented a protocol for *Secret Handshakes*, with different variations, the authority who signed may or may not be the same, the user identity may or may not be secret, ... We also compared our construction to existing CAKE, and showed that we can handle more general kinds of languages, and that our instantiations were way more efficient when compared to the cases they can handle.

This new methodology of implicit proofs based on languages have opened new directions of study, and may prove quite useful in several cases to supplant existing methodology in interactive protocol. We believe this can lead to better constructions, and may avoid pairing requirements in existing ones when they rely too deeply on Groth-Sahai. The servers may now require more granular authentication policies while simultaneously preserving even more the user privacy.

Bibliography

[ABC+05] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, August 2005. ix, 4, 59

[ACGP11] Michel Abdalla, Céline Chevalier, Louis Granboulan, and David Pointcheval. Contributory password-authenticated group key exchange with join capability. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 142–160. Springer, February 2011. 113, 117

[ACHdM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. 17

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 671–689. Springer, August 2009. xiv, xv, xviii, 8, 9, 10, 12, 24, 93, 110, 112, 113, 114, 115, 120, 123

[AF96] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 244–251. Springer, November 1996. 82

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer, August 2010. viii, x, 4, 5, 23, 82

[AKB07] Giuseppe Ateniese, Jonathan Kirsch, and Marina Blanton. Secret handshakes with dynamic and fuzzy matching. In *ISOC Network and Distributed System Security Symposium – NDSS 2007*. The Internet Society, February / March 2007. xiv, 9, 112, 122

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 271–286. Springer, August 2000. 82

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, May 2004. ix, 4, 61

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008. 39, 40

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, August 2004. xii, 7, 17, 27, 55, 61, 73, 109

[BCC+08] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Delegatable anonymous credentials. Cryptology ePrint Archive, Report 2008/428, 2008. 61

[BCF+11] Olivier Blazy, Sébastien Canard, Georg Fuchsbauer, Aline Gouget, Hervé Sibert, and Jacques Traoré. Achieving optimal anonymity in transferable e-cash with a judge. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11: 4th International Conference on Cryptology in Africa*, volume 6737 of *Lecture Notes in Computer Science*, pages 206–223. Springer, July 2011. xxii

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 356–374. Springer, March 2008. ix, xx, xxii, 4, 13, 38, 40, 41

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 361–377. Springer, August 2005. 19, 113, 114, 115, 117

[BCPV12] Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Efficient uc-secure authenticated key-exchange for algebraic languages. Cryptology ePrint Archive, Report 2012/284, 2012. xxiii

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE Computer Society Press, October 1997. 107

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana K. Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *IEEE Symposium on Security and Privacy*, pages 180–196. IEEE Computer Society, 2003. xiv, 9, 103, 112

[BFI+10] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch Groth-Sahai. In Jianying Zhou and Moti Yung, editors, *ACNS 10: 8th International Conference on Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 218–235. Springer, June 2010. xx, xxi, 13, 35

[BFM90] Manuel Blum, Paul Feldman, and Silvio Micali. Proving security against chosen cyphertext attacks. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 256–268. Springer, August 1990. 29

[BFP+01] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *20th ACM Symposium Annual on Principles of Distributed Computing*, pages 274–283. ACM Press, August 2001. 80

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 403–422. Springer, March 2011. viii, xii, xvi, xviii, xxii, 4, 7, 11, 12, 18, 23, 26, 70, 89, 110

[BFPW07] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio, and Bogdan Warinschi. A closer look at PKI: Security and efficiency. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 458–475. Springer, April 2007. 38

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, May / June 1998. 35

[BHS04] Robert W. Bradshaw, Jason E. Holt, and Kent E. Seamons. Concealing complex policies with hidden credentials. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 04: 11th Conference on Computer and Communications Security*, pages 146–157. ACM Press, October 2004. xiv, 9, 103

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 1–35. Springer, April 2009. 21

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992. xv, 9, 112

[BM93] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 244–250. ACM Press, November 1993. xiv, 9, 103

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, May 2003. 55

[BNPS01] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme. In Paul F. Syverson, editor, *FC 2001: 5th International Conference on Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 319–338. Springer, February 2001. 80

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *Lecture Notes in Computer Science*. Springer, 1998. Invited paper. 17, 26

[BP12] Olivier Blazy and David Pointcheval. Traceable signature with stepping capabilities. In David Naccache, editor, *Quisquater Festschrift*, Lecture Notes in Computer Science. Springer, 2012. Full version available from the web page of the authors. xxii, 89

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, May 2000. xiv, 9, 103

[BPV12a] Olivier Blazy, David Pointcheval, and Damien Vergnaud. Compact Round-Optimal Partially-Blind Signatures. In Roberto De Prisco and Ivan Visconti, editors, *The 8th Conference on Security in Communication Networks (SCN '12)*, volume 7485 of *Lecture Notes in Computer Science*, pages 95–113, Amalfi, Italy, 2012. Springer. xxiii

[BPV12b] Olivier Blazy, David Pointcheval, and Damien Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In Ronald Cramer, editor, *Proceedings of TCC 2012*, Lecture Notes in Computer Science. Springer, 2012. Full version available from the web page of the authors. xxiii, 122

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, November 1993. viii, 80

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, February 2005. vii, 3, 55, 56, 58, 66

[BW06] Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 427–444. Springer, May / June 2006. viii, ix, xii, 4, 7

[BW07] Xavier Boyen and Brent Waters. Full-domain subgroup hiding and constant-size group signatures. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 1–15. Springer, April 2007. viii, 4

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society Press, October 2001. 18

[CCGS10] Jan Camenisch, Nathalie Casati, Thomas Groß, and Victor Shoup. Credential authenticated identification and key exchange. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 255–276. Springer, August 2010. xv, xviii, 9, 12, 97, 111, 112, 121

[CG08] Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS 08: 6th International Conference on Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 207–223. Springer, June 2008. xxii

[Cha83] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1983. v, x, 2, 5, 22, 54, 80

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, May 2005. 112, 113, 115, 116, 122

[CHL05a] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, May 2005. 61

[CHL+05b] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 422–439. Springer, May 2005. 20

[CJT04] Claude Castelluccia, Stanislaw Jarecki, and Gene Tsudik. Secret handshakes from CA-oblivious encryption. In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 293–307. Springer, December 2004. xiv, 9, 103

[CKP07] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore-Penrose pseudoinverse and its application to secure linear algebra. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 613–630. Springer, August 2007. xxi, 13, 28

[CKS11] Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In *Advances in Cryptology – ASIACRYPT 2011*, Lecture Notes in Computer Science, pages 449–467. Springer, December 2011. 122

[CKW04] Jan Camenisch, Maciej Koprowski, and Bogdan Warinschi. Efficient blind signatures without random oracles. In Carlo Blundo and Stelvio Cimato, editors, *SCN 04: 4th International Conference on Security in Communication Networks*, volume 3352 of *Lecture Notes in Computer Science*, pages 134–148. Springer, September 2004. x, 5

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, August 2003. 19

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, August 1998. xx, 13, 26

[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, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64. Springer, April / May 2002. xiv, 8, 24, 92, 109

[CSST06] Sébastien Canard, Berry Schoenmakers, Martijn Stam, and Jacques Traoré. List signature schemes. *Discrete Appl. Math.*, 154(2):189–201, 2006. ix, 5, 54, 66

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, April 1991. v, 2, 54, 55, 58

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, August 1992. xi, 61

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598. Springer, August 2001. 29

[DFN06] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 41–59. Springer, March 2006. 80

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. vii, 2, 18, 21

[DM95] Burgess Davis and David McDonald. An elementary proof of the local central limit theorem. *Journal of Theoretical Probability*, 8(3), jul 1995. xviii, 12, 44

[DN07] Cynthia Dwork and Moni Naor. Zaps and their applications. *SIAM J. Comput.*, 36(6):1513–1543, 2007. 111

[DP06] Cécile Delerablée and David Pointcheval. Dynamic fully anonymous short group signatures. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06: 1st International Conference on Cryptology in Vietnam*, volume 4341 of *Lecture Notes in Computer Science*, pages 193–210. Springer, September 2006. x, 5, 61, 62

[Duc10] Léo Ducas. Anonymity from asymmetry: New constructions for anonymous HIBE. In Josef Pieprzyk, editor, *Topics in Cryptology – CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 148–164. Springer, March 2010. 122

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, January 2005. x, 5, 61

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, August 1985. 26

[FGHP09] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 309–324. Springer, April 2009. xx, 13, 35

[Fia90] Amos Fiat. Batch RSA. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 175–185. Springer, August 1990. 35

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer, August 2006. x, 5, 23

[FP09] Georg Fuchsbauer and David Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009: 3rd International Conference on Pairing-based Cryptography*, volume 5671 of *Lecture Notes in Computer Science*, pages 132–149. Springer, August 2009. 74

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, August 1987. viii, 3, 29

[FS90] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 526–544. Springer, August 1990. viii, 4, 29

[Fuc09] Georg Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. Cryptology ePrint Archive, Report 2009/320, 2009. x, 5, 82

[Fuc11] Georg Fuchsbauer. Commuting signatures and verifiable encryption. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 224–245. Springer, May 2011. xxii

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479. IEEE Computer Society Press, October 1984. 23

[GK08] Kristian Gjøsteen and Lillian Kråkmo. Round-optimal blind signatures from Waters signatures. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec*, volume 5324 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2008. 79, 80

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 524–543. Springer, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`. xiv, 8, 24

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. 19

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988. 21, 22, 71

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. vii, 2, 29

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, April 2006. 121

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 97–111. Springer, August 2006. 111

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, May / June 2006. ix, 4

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113 – 3121, 2008. xvi, 11, 17, 35

[Gro07] Jens Groth. Fully anonymous group signatures without random oracles. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 164–180. Springer, December 2007. viii, xx, xxii, 4, 13, 38, 39

[GRS+11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In *Crypto 2011*, volume 6841 of *LNCS*, pages 630–648. Springer, 2011. 111

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, April 2008. xii, 7, 8, 29, 74

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. 105

[HK08] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38. Springer, August 2008. xiii, xviii, 8, 12, 25, 35, 42, 43, 44, 88

[JG02] Ari Juels and Jorge Guajardo. RSA key generation with verifiable randomness. In David Naccache and Pascal Paillier, editors, *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 357–374. Springer, February 2002. 105

[JL09] Stanislaw Jarecki and Xiaomin Liu. Private mutual authentication and conditional oblivious transfer. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 90–107. Springer, August 2009. xiv, 9, 112, 122

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer, August 1997. x, 5

[Kal05] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 78–95. Springer, May 2005. xiv, 8, 24

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, January 1883. vii, 2

[KS05] Jonathan Katz and Ji Sun Shin. Modeling insider attacks on group key-exchange protocols. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05: 12th Conference on Computer and Communications Security*, pages 180–189. ACM Press, November 2005. 113

[KTY04] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 571–589. Springer, May 2004. ix, 4, 54, 57, 58

[LDB03] Ninghui Li, Wenliang Du, and Dan Boneh. Oblivious signature-based envelope. In *22nd ACM Symposium Annual on Principles of Distributed Computing*, pages 182–189. ACM Press, July 2003. xiv, 9, 92, 102, 103, 104, 109, 110

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 446–466. Springer, May 2011. xx, 13, 35, 52

[LR98] Anna Lysyanskaya and Zulfikar Ramzan. Group blind digital signatures: A scalable solution to electronic cash. In Rafael Hirschfeld, editor, *FC'98: 2nd International Conference on Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, pages 184–197. Springer, February 1998. 90

[LY09] Benoît Libert and Moti Yung. Efficient traceable signatures in the standard model. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009: 3rd International Conference on Pairing-based Cryptography*, volume 5671 of *Lecture Notes in Computer Science*, pages 187–205. Springer, August 2009. ix, 4, 58, 59

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130. IEEE Computer Society Press, October 1999. 23

[MSF10] Sarah Meiklejohn, Hovav Shacham, and David Mandell Freeman. Limitations on transformations from composite-order to prime-order groups: The case of round-optimal blind signatures. In Masayuki Abe, editor, *Proceedings of Asiacrypt 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 519–538. Springer, 2010. 79

[Nac05] David Naccache. Secure and *practical* identity-based encryption. Cryptology ePrint Archive, Report 2005/369, 2005. xviii, 12, 42

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, pages 33–43. ACM Press, May 1989. 23

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*. ACM Press, May 1990. 19

[Oka06] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 80–99. Springer, March 2006. x, 5

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, August 1992. 26, 52

[PS96] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer, November 1996. 80

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000. x, 5, 80

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, August 1992. 19

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978. 80

[Sha07] Hovav Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074, 2007. `http://eprint.iacr.org/2007/074.pdf`. xxi, 13, 28, 109

[Sho02] Victor Shoup. OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249, 2002. 108

[SPC95] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology – EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 209–219. Springer, May 1995. 81

[SPMLS02] Jacques Stern, David Pointcheval, John Malone-Lee, and Nigel P. Smart. Flaws in applying proof methodologies to signature schemes. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 93–110. Springer, August 2002. 22

[Sue21] Caius T. Suetonius. Julius caesar. In *De vita Caesarum*, volume I, page 72. 121. vi, 2

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, May 2005. xii, 7, 25, 42, 43, 73, 109

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91. IEEE Computer Society Press, November 1982. 18

Résumé

Dans cette thèse, nous proposons et utilisons de nouvelles briques conduisant à des protocoles efficaces dans le cadre d'une approche modulaire de la cryptographie. Tout d'abord dans un contexte non-interactif en s'appuyant sur les preuves Groth-Sahai, puis avec des interactions en permettant aux *Smooth Projective Hash Functions* de gérer de nouveaux langages.
Dans un premier temps cette thèse s'appuie sur la méthodologie introduite par Groth et Sahai pour des preuves de connaissance non-interactives pour développer divers protocoles de signatures de groupe dans le modèle standard, puis de signatures en blanc. Pour cela, nous proposons un système permettant de signer un chiffré de manière telle que, sous réserve de connaissance d'un secret indépendant du protocole de signature, il soit possible de retrouver une signature sur un clair. Cette approche nous permet entre autre de proposer un protocole de signatures en blanc avec un nombre optimal d'interactions à la fin duquel le demandeur peut utiliser une signature usuelle et ce sous des hypothèses classiques dans le modèle standard.
Ensuite nous proposons une nouvelle méthodologie pour faire des preuves implicites de connaissance dans un contexte interactif sans oracle aléatoire. Pour cela nous utilisons les *smooth projective hash functions*, dans un premier temps pour faire des *Oblivious Signature-Based Envelopes*, puis dans des protocoles d'authentification et de mise en accord de clés. Ce faisant nous précisons la notion de langage, et élargissons grandement le spectre des langages pouvant être traités à l'aide de ces SPHF. Grâce à ce résultat nous introduisons le concept de LAKE (*Language Authenticated Key Exchange*) ou encore Échange de clés authentifié par un langage : un moyen pour deux utilisateurs de se mettre d'accord sur une clé si chacun possède un secret vérifiant une contrainte espérée par l'autre. Nous montrons alors comment instancier plusieurs protocoles d'échange de clé sous ce regard plus efficacement qu'avec les techniques actuelles, et nous prouvons la sécurité de nos instanciations dans le modèle UC sous des hypothèses usuelles.

Abstract

In this thesis, we create new building blocks and use them to present new efficient protocols via a modular design. We first begin by using the Groth-Sahai methodology for non-interactive proofs to design various group signature protocols in the standard model. We also present a new approach allowing to sign ciphertext and then under the knowledge of a secret independent from the signature protocol we show how a user can recover the signature on the plaintext, creating this way some sort of commutative property between signature and encryption where a decryption of a signature on a ciphertext provides a signature on the associated plaintext. This approach allows us to build a *Round-Optimal Blind Signature* scheme where the user can ultimately exploit a regular signature. We prove the security of this construction under classical hypotheses in the standard model.
We then present a new methodology for implicit proofs of knowledge in an interactive environment without random oracle. For that we use *Smooth Projective Hash Functions*, first to instantiate *Oblivious Signature-Based Envelope* schemes, and then to create Authenticated Key Exchange scheme. Throughout this process we further refine the notion of language, and greatly widen the set of languages manageable via SPHF. This last result allows us to introduce the concept of LAKE (*Language Authenticated Key Exchange*), a new AKE design where two users will be able to share a common key if they both possess a secret word in a language expected by the other. We then show how to build standard AKE schemes (like *Password Authenticated Key Exchange*) using our framework, and show that our design leads to an increment in efficiency from pre existing solutions. We prove the security of our design in the UC framework under regular hypotheses.