Signatures pour l'anonymat fondées sur les couplages et applications

Emeline Hufschmitt

UNIVERSITÉ de CAEN/BASSE-NORMANDIE

U.F.R. de Sciences

ÉCOLE DOCTORALE SIMEM

THÈSE

présentée par

Mlle Émeline HUFSCHMITT

et soutenue

le jeudi 29 novembre 2007

en vue de l'obtention du

DOCTORAT de l'UNIVERSITÉ de CAEN

Spécialité : informatique

(Arrêté du 07 août 2006)

TITRE

Signatures pour l'anonymat fondées sur les couplages et applications

MEMBRES du JURY

Louis GOUBIN, professeur, Université de Versailles Saint-Quentin-en-Yvelines (rapporteur)
David POINTCHEVAL, directeur de recherche CNRS, École normale supérieure (rapporteur)
Jacques TRAORÉ, expert senior, Orange Labs R&D
Brigitte VALLÉE, directrice de recherche CNRS, Université de Caen Basse-Normandie
Gilles ZÉMOR, professeur, Université de Bordeaux
Marc GIRAULT, HDR, expert émérite, Orange Labs R&D (directeur de thèse)

Décor

La scène se passe dans la salle des thèses de l'université de Caen. Sur le mur du fond, posé au milieu, un écran blanc. Sur le côté, une thésarde, toute tremblante de peur devant son ordinateur.

Au premier rang, directement devant elle, son jury, composé de la manière suivante :
Marc Girault*, le directeur, qui par son soutien pendant trois ans a permis à la thésarde d'être là ;*
Jacques Traoré*, l'encadrant, qui a fait en sorte que la thésarde reste dans le droit chemin et progresse dans ses recherches ;*
Louis Goubin *et* David Pointcheval*, les rapporteurs, qui ont accepté de rapporter son mémoire dans un délai un peu court (!) et qui par leurs remarques ont permis d'en améliorer la qualité ;*
et enfin Brigitte Vallée *et* Gilles Zémor *dont la présence est un honneur.*

Dans la salle, répartis selon leur bon vouloir, on trouvera :
Fabrice*, le chef du laboratoire MPS/NSS dans lequel la thésarde a travaillé pendant trois ans et qui l'a accueillie avec beaucoup d'humour ;*
les cobureaux et tout particulièrement Sébastien*, qui a supporté, dans tous les sens du terme, la thésarde ;*
les collègues, trop nombreux pour être cités, mais qui ont participé à l'aboutissement de cette thèse ;
Aline, David *et* Hervé*, avec qui la thésarde s'est initiée aux joies des soumissions et des deadlines.*

On trouvera également des amis venus soutenir la thésarde :
Marc*, celui qui a partagé la vie de la thésarde et qui l'a soutenue dans tous les moments bons ou mauvais ;*
Pascal*, grâce à qui la thésarde a compris enfin le sens de l'expression « Appel à un ami » ;*
Axelle*, qui a compris que les silences n'étaient pas un manque d'attention ;*
Fabien*, qui pour être passé par le même chemin a su trouver les bons mots pour rassurer la thésarde ;*
Fanny*, qui grâce à des discussions sur les problèmes relationnels des jeunes d'aujourd'hui a su remonter le moral de la thésarde dans ses moments difficiles ;*
Ion, Manu *et les autres partenaires de poker, qui ont sans aucun scrupule plumé la thé-*

sarde ;
Iwen*, qui en plus d'être un collègue est devenu véritablement un ami ;*
Léo*, qui par sa vision de la vie a toujours été une bouffée de bonheur ;*
Yohann et Cécile*, qui par leurs expériences culinaires un peu spéciales ont développé le palais de la thésarde.*

Et puis mes parents*, encore plus fiers que* moi *d'être là.*

Le rideau se lève.

Table des matières

Notations

$\mathbb{N}$	Ensemble des entiers naturels
$\mathbb{Z}$	Ensemble des entiers relatifs
$\mathbb{Z}_n$	Anneau des classes de résidus modulo n
$\mathbb{Z}_n^*$	Groupe des éléments inversibles de $\mathbb{Z}_n$ selon la loi de multiplication
$\langle g \rangle$	Groupe engendré par l'élément g
$\mathrm{ord}(g)$	Ordre de l'élément g d'un groupe dans ce groupe
$\phi(n)$	Indicatrice d'Euler d'un entier $n \geq 2$
$\lambda(n)$	Indicatrice de Carmichaël d'un entier $n \geq 2$
$\mathrm{pgcd}(a, b)$	Plus grand commun diviseur des deux entiers non nuls a et b
$\mathrm{ppcm}(a, b)$	Plus petit commun multiple des deux entiers non nuls a et b
$[a, b]$	Intervalle des entiers entre a et b
$\lvert \mathrm{X} \rvert$	Cardinal de l'ensemble X
$\{0,1\}^*$	Ensemble des mots de longueur quelconque constitués des caractères 0 ou 1
$\{0,1\}^k$	Ensemble des mots de longueur k constitués des caractères 0 ou 1
$\perp$	Chaîne de caractère vide
$a \lvert\lvert b$	Concaténation des suites de bits de a et b
$\cup\cap \wedge\vee$	Opérateurs ensemblistes et logiques « et » et « ou »
$x \in_{\mathrm{R}} \mathrm{E}$ ou $x \xleftarrow{\mathrm{R}} \mathrm{E}$	L'élément x est tiré au hasard dans l'ensemble fini E selon une distribution uniforme
$a \leftarrow b$	la valeur de b est affectée à a
$x \leftarrow \mathcal{A}(\ldots)$	L'algorithme $\mathcal{A}$ produit l'élément de sortie x
$\Pr[\mathrm{R}_1, \ldots, \mathrm{R}_n \lvert \mathrm{E}]$	probabilité de l'événement E après les exécutions successives des événements $\mathrm{R}_1, \ldots \mathrm{R}_n$
$pok = pok(\alpha : \mathrm{T}(\alpha, \ldots))$	Preuve de connaissance de la valeur α vérifiant le prédicat T
$f_{g,s}^{\mathsf{DY}}(x)$	Fonction pseudo-aléatoire de Dodis-Yampolskyi, de générateur g et de graine x, évaluée en x
$\mathsf{PedCom}(x; r)$	Engagement de Pedersen de la valeur x calculé avec l'aléa r

$\mathsf{Sign}_{sk}^{\mathsf{Schéma}}$	Schéma de signature Schéma utilisant la clé privée sk
$\mathsf{Encrypt}_{pk}^{\mathsf{Schéma}}$	Schéma de chiffrement Schéma utilisant la clé publique pk
BBS	Signature de groupe de Boneh, Boyen et Shacham
ACJT	Signature de groupe d'Ateniese, Camenisch, Tsudik et Joye
FBS	Schéma de signatures aveugles à anonymat révocable
MC	Schéma de multi-coupons
ME	Schéma de monnaie électronique

Nous utilisons la description suivante des expériences aléatoires modélisant une attaque sur un problème algorithmique :

Expérience $\mathbf{Exp}_{\mathsf{G},\mathcal{A}}^{\mathsf{problème}}(k)$
E_1
$\vdots$
E_n
Sortie

Cette expérience aléatoire décrit une attaque de l'adversaire $\mathcal{A}$ sur le générateur G d'instances de problème par rapport à un paramètre de sécurité k. Si l'attaquant est de type décisionnel, on ajoute également un bit b. Des événements $\mathrm{E}_1, \ldots, \mathrm{E}_n$ se succèdent jusqu'à la fin de l'expérience qui se termine par une sortie $\mathsf{Sortie} \in \{0, 1\}$.

Pour les attaques sur les cryptosystèmes nous utilisons des expériences se déroulant sur le modèle suivant :

Expérience $\mathbf{Exp}_{\mathsf{Schéma},\mathcal{A}}^{\mathsf{attaque}}(k)$
E_1
$\vdots$
E_n
Sortie

Cette expérience aléatoire décrit une attaque de $\mathcal{A}$ sur le schéma Schéma dans une attaque de type attaque.

Si l'adversaire est de type calculatoire, on définit son avantage de la façon suivante

$$\mathbf{Adv}_{\mathrm{YY},\mathcal{ZZ}}^{\mathsf{XX}}(k) = \Pr[\mathbf{Exp}_{\mathrm{YY},\mathcal{ZZ}}^{\mathsf{XX}}(k) = 1].$$

S'il est décisionnel, son avantage est donné par

$$\mathbf{Adv}_{\mathrm{YY},\mathcal{ZZ}}^{\mathsf{XX}}(k) = |Pr[\mathbf{Exp}_{\mathrm{YY},\mathcal{ZZ}}^{\mathsf{XX}-1}(k) = 1] - Pr[\mathbf{Exp}_{\mathrm{YY},\mathcal{ZZ}}^{\mathsf{XX}-0}(k) = 1]|.$$

Les probabilités sont considérées sur les aléas internes des algorithmes intervenant dans les événements $\mathrm{E}_1, \ldots, \mathrm{E}_n$ et sur les différents tirages aléatoires.

Introduction

La cryptologie, souvent définie comme la *science du secret*, est une discipline dont les origines remonteraient au XVI^{e} siècle avant Jésus-Christ. Cette science est divisée en deux branches étroitement liées : la *cryptographie*, qui vise à spécifier de nouveaux procédés, et la *cryptanalyse*, qui vise à mettre ces derniers en défaut. Ces deux notions ne peuvent exister l'une sans l'autre et leur conjugaison permet de les développer mutuellement. Ce mémoire s'intéresse principalement à la construction de procédés cryptographiques, ou *cryptosystèmes*, et relève donc plutôt de la cryptographie. Cependant, afin de s'assurer de la solidité de ces procédés, il est nécessaire de vérifier qu'ils résistent aux méthodes de cryptanalyse connues, une manière de faire étant d'exhiber des preuves de sécurité.

Pendant longtemps la cryptographie fut essentiellement utilisée à des fins militaires. L'exemple le plus connu est certainement le *système de César*, dont ce dernier se servit pendant la guerre des Gaules. Ce procédé masque le contenu d'un message à l'aide d'une substitution. Plus précisément, les lettres de l'alphabet sont décalées pour *chiffrer* le message et seules les personnes ayant connaissance du décalage utilisé sont à même de *déchiffrer* le contenu. Cette méthode nécessite donc que les deux parties en présence, à savoir l'émetteur et le destinataire, se mettent d'accord au préalable, sur le « décalage » utilisé. Elle relève de ce qu'on appelle la *cryptographie symétrique* ou *à clé secrète*, ainsi nommée car les opérations de chiffrement et de déchiffrement reposent sur le même *secret*. Ce partage d'un secret commun a été pendant longtemps la seule façon connue d'échanger des messages secrets. Le *chiffrement de Vigenère* en constitue un autre exemple. Il fonctionne comme le système de César, à ceci près que le décalage, au lieu d'être constant, varie avec la position de la lettre destinée à être chiffrée. La clé est cette fois constituée d'un mot ou d'une phrase, dont chaque caractère indique quel doit être ce décalage.

Mais ce sont les deux guerres mondiales qui établiront véritablement la cryptographie comme une arme. La machine Enigma en est un parfait exemple. Son « cassage » par les autorités françaises, polonaises et britanniques permettra d'ailleurs aux armées alliées de déjouer bon nombre de plans allemands.

La sécurité de ces systèmes reposait sur le secret des procédés employés ou sur l'impossibilité de recourir à des méthodes exhaustives, consistant à essayer toutes les valeurs de clés possibles jusqu'à déchiffrer un message cohérent. De nos jours, les procédés utilisés sont presque toujours publiés et les capacités croissantes des ordinateurs ont obligé à reconsidérer la taille des clés. C'est dans ce but que le *Data Encryption Standard* (DES) a été conçu dans les années 70. Cependant son emploi n'est plus re-

commandé aujourd'hui, du fait de sa (relative) lenteur d'exécution et de sa taille de clés devenue trop petite. Son successeur est l'*Advanced Encryption Standard* (AES), choisi par le gouvernement des États-Unis en octobre 2000 pour être le nouveau standard de chiffrement.

En 1976, Diffie et Hellman ont été à l'origine d'un tournant majeur en cryptographie. Leur article *« New directions in Cryptography »* [DH76] présente une nouvelle manière de concevoir le rôle des clés. Cette fois, chaque utilisateur possède deux clés, l'une qu'il garde secrète, appelée *clé secrète* ou *clé privée*, et une deuxième connue de tous, appelée *clé publique*. Ce principe est à la base de la *cryptographie asymétrique* ou *cryptographie à clé publique*. Le premier schéma de chiffrement basé sur ce principe est dû à Rivest, Shamir et Adleman [RSA78] et est connu sous le nom de RSA. Depuis lors, la cryptographie à clé publique n'a cessé de se développer.

Si l'engouement des chercheurs pour cette *cryptographie moderne* fut immédiat, il ne signifia pas pour autant la fin de la cryptographie symétrique. En effet, même si la cryptographie asymétrique offre l'avantage de ne pas nécessiter d'échange de clé secrète avant toute opération, les systèmes à clé secrète sont la plupart du temps beaucoup plus rapides. Le mieux est de combiner les deux systèmes afin de tirer partie des avantages des deux. Lorsque deux interlocuteurs souhaitent échanger des données en toute confidentialité, ils commencent à utiliser un système à clé publique afin de se mettre d'accord sur une clé secrète qui sera ensuite utilisée pour chiffrer les messages avec un système symétrique.

Qu'elle soit symétrique ou asymétrique, la cryptographie peut poursuivre quatre buts principaux.

La *confidentialité* est, historiquement, le premier de ces buts. Elle garantit que seules les personnes autorisées peuvent accéder au contenu d'un message transmis ou stocké. Cette propriété est obtenue en chiffrant le message, de façon à ce que seules ces personnes puissent le rétablir en clair.

L'*intégrité* est complémentaire de la confidentialité. Elle garantit que le message n'a pas été altéré pendant la transmission, c'est-à-dire que la version reçue est identique à celle envoyée. On peut pour cela accompagner le message d'une signature numérique.

L'*authentification d'entité ou identification* permet de s'assurer de l'identité d'une entité et donc de détecter ceux qui tentent de l'usurper. Elle peut se faire soit en révélant un secret (tel qu'un mot de passe), soit en prouvant qu'on connaît bien un secret, mais sans le dévoiler.

L'*authentification de message* permet d'*authentifier* la provenance d'un message, c'est-à-dire prouver qu'il a bien été envoyé par la personne prétendue. Une manière d'authentifier un message est de le signer numériquement.

Les deux buts suivants sont moins courants en cryptographie mais d'un intérêt tout particulier pour nos travaux.

La *non-répudiation* empêche qu'on puisse revenir sur sa parole ou sur ses actes. Elle est généralement obtenue grâce à l'usage d'une signature numérique, laquelle ne peut être reniée.

L'*anonymat* permet d'engager une identité dans une procédure cryptographique tout en gardant son identité secrète. Il peut par exemple servir à prouver que l'on connaît

certaines données sans pour autant dévoiler son identité.

Cette thèse s'intéresse à la combinaison de l'authentification et de l'anonymat, et plus particulièrement à ce que nous appellerons *signatures pour l'anonymat*. Ces signatures permettent soit de masquer l'identité du signataire au sein d'un ensemble de signataires possibles (signatures de groupe, signatures d'anneau), soit de masquer le contenu du message auprès d'un signataire bien identifié (signatures aveugles).

Par ailleurs, comme il est tout à fait possible d'imaginer qu'une telle garantie d'anonymat puisse être utilisée à des fins malhonnêtes, il existe, pour certaines de ces signatures, des procédures permettant de faire « marche arrière » c'est-à-dire de lever l'anonymat. Ceci est généralement confié à une autorité compétente, qui est la seule à détenir ce pouvoir de *révocation d'anonymat*. Dans ce contexte-là, nous nous sommes plus particulièrement intéressés aux *signatures aveugles à anonymat révocable*, pour lesquelles nous avons spécifié un nouveau modèle de sécurité et un nouveau protocole. Nous nous sommes ensuite penchés sur les applications de ces signatures, et plus particulièrement la monnaie et les coupons électronique.

Le présent mémoire, qui regroupe les résultats obtenus, est structuré de la manière suivante.

Dans une première partie, nous fournissons les outils (notamment mathématiques) nécessaires à nos constructions futures. Nous définissons aussi les principales primitives cryptographiques sur lesquelles elles reposeront.

La deuxième partie donne l'état de l'art des signatures pour l'anonymat existantes. Le chapitre 4 présente les signatures de groupe qui permettent à un utilisateur de signer au sein d'un groupe sans révéler son identité, sauf lorsqu'une procédure de révocation est mise en oeuvre. Le chapitre 5 décrit les signatures aveugles qui permettent à un utilisateur de faire signer un message par un signataire, sans que ce dernier n'ait connaissance de son contenu. Le chapitre 6 est relatif aux signatures d'anneau et à leurs variantes. C'est l'occasion pour nous d'y présenter notre première contribution. Il s'agit d'un schéma d'authentification fondé sur l'utilisation de couplages, que l'on peut transformer en signature d'anneau de deux personnes. Ce résultat a fait l'objet d'une publication au workshop *WEWorc* [HLS05].

La troisième partie constitue le cœur de ce mémoire. Nous y présentons une variante des signatures aveugles introduite en 1995 par Camenisch, Piveteau et Stadler [SPC95] : les signatures aveugles à anonymat révocable. Ces signatures offrent les mêmes propriétés d'anonymat qu'une signature aveugle, à la différence près qu'une autorité de révocation est capable de lever cet anonymat et ce de deux manières différentes. Le chapitre 7 présente un état de l'art de ces signatures et plus particulièrement la construction de [AO01], dont nous montrons qu'elle ne satisfait pas les propriétés de sécurité annoncées par ses auteurs. Nous présentons ensuite un nouveau modèle de sécurité pour les signatures aveugles à anonymat révocable en partant des définitions de [AO01], puis en les reprécisant et en les complétant. Ceci nous permet de construire un nouveau schéma et de prouver sa sécurité dans le chapitre 9, le dernier chapitre de cette partie. Ces résultats ont été présentés à la conférence *Pairing 07* et publiés dans [HT07].

La dernière partie s'intéresse à deux applications des signatures pour l'anonymat : la monnaie électronique et les coupons électroniques. Ces deux systèmes reposent sur des concepts très proches. Nous partons des travaux de Camenisch, Hohenberger et Lysysankaya [CHL05] et spécifions un nouveau système de multi-coupons et de monnaie électronique, doté de nouvelles propriétés qui les rendent très pratiques. En particulier un utilisateur peut retirer le nombre de coupons ou de pièces de son choix afin de composer son porte-monnaie. De plus, il peut décider de la valeur des coupons ou des pièces qu'il retire. Chemin faisant, nous décrivons un nouveau modèle de sécurité pour les systèmes de multi-coupons et de monnaie et prouvons la sécurité de nos schémas dans ces modèles. Ces résultats ont été publiés aux conférences ACNS'06 [CGH06] et SAR-SSI'07 [CGH07].

Première partie

Généralités

Chapitre 1

Cryptographie à clé publique

Le but de ce premier chapitre est d'introduire les outils essentiels à la cryptographie à clé publique.

Nous commençons par rappeler quelques notions de mathématiques et de théorie de la complexité importantes pour la compréhension de notre mémoire. En particulier nous présentons en détail les applications bilinéaires admissibles que nous utiliserons abondamment par la suite.

Nous définissons ensuite quelques fonctions particulières avant de présenter les primitives principales de la cryptographie à clé publique. Ces primitives nous servirons plus tard à construire de nouveaux cryptosystèmes.

Sommaire

1.1 Outils mathématiques

Cette section présente les outils mathématiques dont nous aurons besoin par la suite. Nous commençons par quelques rappels en théorie des groupes et en théorie des nombres.

Puis nous définissons les courbes elliptiques et les applications bilinéaires admissibles dont nous nous servirons intensément dans les parties III et IV.

1.1.1 Rappels mathématiques

1.1.1.1 Théorie des groupes

Définition 1. *Le nombre d'éléments d'un groupe fini* $\mathbb{G}$*, noté* $|\mathbb{G}|$*, est appelé* ordre *de* $\mathbb{G}$*.*

Dans toute la suite de ce mémoire, $\mathbb{G}$ désignera un groupe fini dont la loi sera notée multiplicativement et l'élément neutre noté 1.

Définition 2 (Ordre d'un élément)**.** *L'ordre d'un élément* g *de* $\mathbb{G}$ *est le plus petit entier positif* m *tel que* $g^m = 1$*.*

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

Définition 3. *Un groupe* $\mathbb{G}$ *est dit* cyclique *s'il existe un élément* g *de* $\mathbb{G}$ *tel que, pour tout élément* h *de* G*, il existe un entier* x *vérifiant* $h = g^x$*. Un tel élément* g *est appelé* générateur *de* $\mathbb{G}$*. On écrit alors* $\mathbb{G} = \langle g \rangle$*.*

1.1.1.2 Théorie des nombres

La cryptographie à clé publique fait très largement appel à la théorie des nombres. C'est pourquoi nous en rappelons ici quelques notions essentielles.

Dans ce mémoire, un entier naturel sera indifféremment appelé entier ou nombre. Un *nombre premier* est un nombre qui n'est divisible que par lui même et par 1. Deux nombres a et n sont *premiers entre eux* si leur plus grand dénominateur commun, noté $\text{pgcd}(a, n)$, est égal à 1.

Définition 4 (Indicatrice d'Euler)**.** *Pour tout entier* n *on note* $\phi(n)$ *le nombre d'entiers inférieurs à* n *qui sont premiers avec* n*. La fonction* ϕ *est appelée* indicatrice d'Euler*.*

Proposition 1. *L'indicatrice d'Euler possède les propriétés suivantes :*
- *si* p *est premier, alors on a* $\phi(p) = p - 1$*,*
- *si* p *et* q *sont premiers entre eux, alors on a* $\phi(pq) = \phi(p)\phi(q)$*,*
- *si* p *et* q *sont des premiers distincts, alors* $n = pq$ *implique* $\phi(n) = (p-1)(q-1)$*.*

La notion de primalité est souvent essentielle dans la construction de cryptosystèmes et il est important de savoir trouver des nombres premiers ou déterminer si des nombres sont premiers ou non. Il existe pour cela plusieurs résultats dont les théorèmes suivants :

Théorème 2 (Petit Théorème de Fermat)**.** *Soient* p *un nombre premier et* a *un entier vérifiant* $pgcd(a, p) = 1$*. Alors on a* $a^{p-1} \equiv 1 \mod p$*.*

Théorème 3 (Théorème d'Euler-Fermat)**.** *Soient* a *et* n *deux entiers vérifiant* $\text{pgcd}(a, n) = 1$*. Alors on a* $a^{\phi(n)} \equiv 1 \mod n$*.*

Un autre résultat important concernant les nombres premiers est le Théorème des Restes Chinois.

Théorème 4 (Théorème des Restes Chinois)**.** *Soient $n_1, \ldots, n_k$ des entiers deux à deux premiers entre eux et de produit n. Alors pour tous entiers $a_1, \ldots, a_k$, il existe un entier x, unique modulo n et tel que*

$$\begin{cases} x \equiv a_1 \mod n_1 \\ \quad \ldots \\ x \equiv a_k \mod n_k \end{cases}$$

Groupe $\mathbb{Z}_n^*$. Dans tout le mémoire, $\mathbb{Z}_n$ représentera l'anneau des classes de résidus modulo n et $\mathbb{Z}_n^*$ le groupe des éléments inversibles pour la multiplication de $\mathbb{Z}_n$. D'après la définition de l'indicatrice d'Euler, on a donc $|\mathbb{Z}_n^*| = \phi(n)$.

Dans le cas particulier où p est premier, alors $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$ est un corps cyclique et $|\mathbb{Z}_p^*| = p - 1$.

Définition 5. *Soit a dans $\mathbb{Z}_n^*$. L'*ordre *de a, noté* $\mathrm{ord}(a)$*, est le plus petit entier positif r satisfaisant $a^r \equiv 1 \mod n$.*

Pour construire l'ensemble $\mathbb{Z}_n^*$ à partir des éléments de $\mathbb{Z}_n$ il faut être capable de trouver les inversibles de $\mathbb{Z}_n$, c'est-à-dire les éléments premiers avec n. Une manière efficace de le faire est d'utiliser le théorème de Bézout.

Théorème 5 (Théorème de Bézout)**.** *Soient a et b deux nombres entiers. Soit d le pgcd de a et b. Alors, il existe α et β dans $\mathbb{Z}$ tels que*

$$a\alpha + b\beta = d.$$

Pour inverser a modulo n il nous faut donc trouver l'entier a tel qu'il existe α et β vérifiant : $a\alpha + n\beta = 1$.

Pour calculer le pgcd de deux nombres, la méthode naïve consiste à commencer par factoriser a et b. Or, lorsqu'on travaille avec de grands entiers (comme c'est le cas en cryptographie), la factorisation s'avère vite très difficile. L'*algorithme d'Euclide* permet de calculer de manière efficace le pgcd de deux entiers. L'*algorithme d'Euclide étendu* permet ensuite de trouver les entiers α et β. Dans le cas $\mathrm{pgcd}(a, n) = 1$, l'inverse de a est donné par la valeur α. Nous renvoyons le lecteur au *Handbook of applied cryptography* [MvOV01] pour une description de ces algorithmes.

Nous définissons maintenant certaines classes d'entiers dont les constructions particulières permettent, à condition de choisir les paramètres adéquats, de définir des cryptosystèmes sûrs.

Modules RSA.

Définition 6 (Module RSA)**.** *Soient p et q deux nombres premiers distincts impairs. L'entier $n = pq$ est appelé* module RSA.

L'appellation « module RSA » fait référence au cryptosystème RSA que nous présentons dans la section 1.4.1.

Pour des raisons de sécurité, il est parfois nécessaire de se restreindre à un ensemble spécifique de modules RSA. Pour cela, nous définissons les nombres premiers dits « sûrs » qui permettent ensuite de construire les modules RSA dits « sûrs ».

Définition 7 (Nombre premier sûr)**.** *Un nombre premier p est dit* sûr *si on a $p = 2p'+1$, où p' est lui-même un nombre premier.*

Définition 8 (Module RSA sûr)**.** *Soient p et q deux nombres premiers sûrs distincts. L'entier $n = pq$ est appelé* module RSA sûr*.*

Résidus quadratiques. Nous définissons maintenant les résidus quadratiques, qui sont les carrés inversible de $\mathbb{Z}_n$.

Définition 9 (Résidus quadratiques)**.** *Un entier $a \in \mathbb{Z}_n^*$ est un* résidu quadratique modulo n *s'il existe un $x \in \mathbb{Z}_n^*$ vérifiant :*

$$x^2 \equiv a \mod n.$$

(Dans le cas contraire, on dit que a est un non-résidu quadratique modulo n*.)*

On notera par la suite $\mathrm{QR}(n)$ le sous-groupe de $\mathbb{Z}_n^*$ des résidus quadratiques modulo n.

Pour savoir si un nombre est un résidu quadratique ou non modulo un nombre premier p, on utilise le symbole de Legendre.

Définition 10 (Symbole de Legendre)**.** *Soit p un nombre premier impair et a un entier. Le symbole de Legendre noté $\left(\frac{a}{p}\right)$ est défini par*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{si } p|a \\ a^{\frac{p-1}{2}} \mod p & \text{sinon} \end{cases}$$

Théorème 6. *Soit p un nombre premier impair et a un entier. On a alors*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{si } a \in \mathrm{QR}(p) \\ -1 & \text{si } a \notin \mathrm{QR}(p). \end{cases}$$

Pour les entiers impairs, on peut utiliser le symbole de Jacobi, une généralisation du symbole de Legendre.

Définition 11 (Symbole de Jacobi)**.** *Soit $n \geq 3$ un nombre entier impair dont la factorisation est $n = p_1^{a_1} \cdots p_k^{a_k}$, avec $p_1, \ldots, p_k$ des premiers distincts et $a_1, \ldots, a_k$ des entiers. Alors le symbole de Jacobi $\left(\frac{a}{n}\right)$ est défini par*

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{a_1} \cdots \left(\frac{a}{p_k}\right)^{a_k}.$$

On notera qu'il existe un algorithme efficace permettant de calculer $\left(\frac{a}{p}\right)$ sans connaître la factorisation de n [MvOV01].

Il est alors possible d'établir le lemme suivant :

Lemme 1. *Soit $n = pq$ un module RSA sûr, alors* $\mathrm{QR}(n)$ *est cyclique et pour tout entier $a \in \mathbb{Z}_n^*$, pgcd$(a \pm 1, n) = 1$ implique $\langle a^2 \rangle = \mathrm{QR}(n)$.*

Nombres de Carmichaël. Nous présentons rapidement les nombres de Carmichaël ainsi que la fonction de Carmichaël et ses propriétés sur les éléments de $\mathbb{Z}_{n^2}^*$.

Définition 12 (Nombres de Carmichaël)**.** *Un* nombre de Carmichaël *est un entier n qui, pour tout entier b premier à n, satisfait l'équivalence suivante :*

$$b^{n-1} \equiv 1 \pmod{n}.$$

Définition 13 (Fonction de Carmichaël)**.** *La* fonction de Carmichaël λ *est définie de la manière suivante. Pour $n = pq$, avec p et q premiers, $\lambda(n)$ est donné par :*

$$\lambda(n) = \mathrm{ppcm}(p-1, q-1),$$

où $\mathrm{ppcm}(p-1, q-1)$ *est le plus petit multiple commun à $(p-1)$ et $(q-1)$.*

Pour faciliter la lecture, nous utiliserons la notation λ pour désigner $\lambda(n)$.

Proposition 2 (Propriétés de la fonction de Carmichaël)**.** *Pour tout $w \in \mathbb{Z}_{n^2}^*$ on a*

$$\begin{cases} w^{\lambda} = 1 \mod n, \\ w^{n\lambda} = 1 \mod n^2. \end{cases}$$

Corps finis. Nous rappelons ici quelques définitions et propositions sur les corps, plus particulièrement les corps finis.

Définition 14 (Corps finis)**.** *Un* corps fini K *est un corps contenant un nombre fini d'éléments. L'*ordre *de* K *est alors le nombre d'éléments du corps.*

Définition 15 (Caractéristique d'un corps)**.** *La* caractéristique *d'un corps est le plus petit entier n non nul tel que $1 + 1 + \cdots + 1$ (avec n termes) est nul. S'il n'existe pas d'entier non nul vérifiant cette propriété, on dit que le corps est de caractéristique nulle.*

Proposition 3 (Existence et unicité des corps finis)**.** *Les deux propriétés suivantes sont vérifiées :*

1. *tout corps fini contient q^m éléments où q est un nombre premier et $m \geq 1$,*
2. *pour tout entier premier q et tout entier m, il existe un unique corps fini (à isomorphisme de corps près) d'ordre q^m.*

Proposition 4. *Soit* K *un corps fini d'ordre q^m, avec q premier, alors la caractéristique de* K *est q.*

Définition 16 (Extension et clôture algébrique)**.** *Soit* K *un corps. Les nombres racines d'un polynôme non nul à coefficients dans* K *sont appelés* nombres algébriques*. Le corps engendré par* K *et un nombre algébrique est appelé* extension algébrique de K*. Tous ses éléments sont algébriques sur* K*. Le corps formé de toutes les extensions algébriques de* K *est appelé la* clôture algébrique *de* K *et est noté* $\bar{\mathrm{K}}$*.*

Proposition 5. *Tout corps* K *possède une clôture algébrique et cette clôture est unique à isomorphisme près.*

1.1.2 Courbes elliptiques

Les courbes elliptiques ont été utilisées pour la première fois en cryptographie par Miller en 1985 [Mil85] puis par Koblitz en 1987 [Kob87].

L'objet de cette section est de définir les courbes elliptiques (plus particulièrement sur les corps finis) et les applications bilinéaires admissibles. Les définitions données ici proviennent en grande partie du livre référence de Silverman [Sil86] auquel on pourra se reporter pour plus de détails.

1.1.2.1 Définition

Les courbes elliptiques peuvent être définies en coordonnées projectives ou affines. Pour plus de simplicité, nous utilisons la version affine donnée par l'équation de Weierstrass :

Définition 17 (Courbe elliptique)**.** *Soit* K *un corps fini. Une* courbe elliptique E(K) définie sur K *est l'ensemble des solutions dans* K^2 *d'une équation de Weierstrass*

$$\mathrm{E} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

avec a_1*,* a_2*,* a_3*,* a_4*,* $a_6 \in \mathrm{K}$*, complété par le point à l'infini* $\mathcal{O}$*.*

On a donc

$$\mathrm{E}(\mathrm{K}) = \{(x, y) \in \mathrm{K}^2, y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\}.$$

Dans les cas où la caractéristique de K n'est ni 2 ni 3 on peut aussi écrire l'équation sous une forme plus courte de l'équation de Weierstrass :

$$\mathrm{E} : y^2 = x^3 + ax + b \text{ pour } a, b \in \mathrm{K}.$$

1.1.2.2 Structure de groupe

Les courbes elliptiques ont la propriété remarquable de bénéficier d'une structure de groupe abélien.

On peut en effet définir une opération d'addition de points (notée +), l'élément neutre du groupe étant donné par le point à l'infini $\mathcal{O}$.

La loi dite *corde et tangente* permet d'exprimer de manière géométrique l'addition de deux points sur une courbe elliptique E(K).

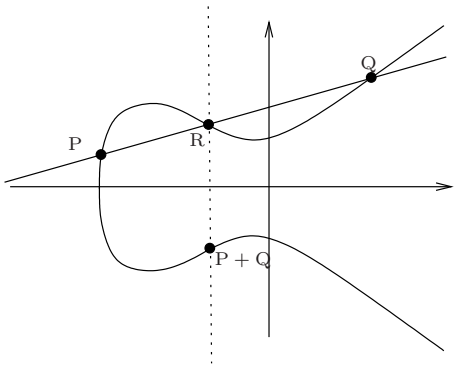

FIG. 1.1 – Addition de deux points sur une courbe elliptique

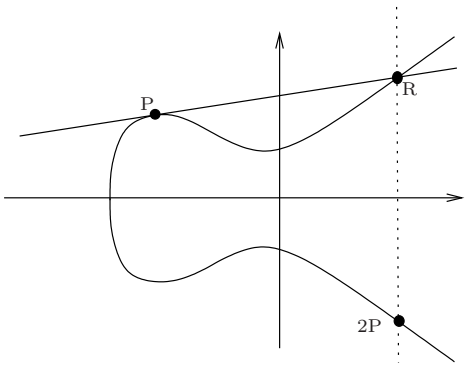

FIG. 1.2 – Doublement d'un point sur une courbe elliptique

Définition 18 (Loi d'addition - loi corde et tangente)**.** *Pour tous points* P *et* Q *situés sur la courbe elliptique* E(K)*, l'addition de* P *et* Q *est donnée par les règles suivantes :*

- $-\mathcal{O} = \mathcal{O}$*,*
- $\mathrm{Q} = -\mathrm{P}$ *implique* $\mathrm{Q} + \mathrm{P} = \mathcal{O}$*,*
- $\mathcal{O} + \mathrm{P} = \mathrm{P}$ *et* $\mathrm{P} + \mathcal{O} = \mathrm{P}$*,*
- $\mathrm{P} = (x_1, y_1) \neq \mathcal{O}$ *implique* $-\mathrm{P} = (x_1, -y_1 - a_1 x_1 - a_3)$*,*
- *si* $\mathrm{P}, \mathrm{Q} \neq \mathcal{O}$ *et* $\mathrm{Q} \neq -\mathrm{P}$*, la loi* corde et tangente *permet de définir le point* $\mathrm{P} + \mathrm{Q}$ *de la manière suivante :*
  - *si* $\mathrm{P} \neq \mathrm{Q}$*, on note* R *le troisième point d'intersection (en comptant la multiplicité) de la droite* (PQ) *avec* E(K)*,*
  - *si* $\mathrm{P} = \mathrm{Q}$ *on note* R *le deuxième point d'intersection de la tangente à* E(K) *en* P*.*

  Alors $\mathrm{P} + \mathrm{Q} = -\mathrm{R}$*. (voir figure 1.1 et 1.2 pour une interprétation visuelle).*

On a alors le théorème suivant :

Théorème 7. $(\mathrm{E}(\mathrm{K}), +)$ *est un groupe abélien d'élément neutre* $\mathcal{O}$*.*

La partie la plus délicate de la démonstration est la propriété d'associativité. La démonstration provient essentiellement du théorème de Riemann-Roch (voir [Har77] ou [Sil86] pour une démonstration complète).

Dans le cas particulier où la caractéristique de K est différente de 2 ou 3, l'addition de deux points $\mathrm{P} = (x_1, y_1)$ et $\mathrm{Q} = (x_2, y_2)$ est donnée par le point $\mathrm{R} = (x_3, y_3)$ défini de la manière suivante lorsqu'on utilise l'équation courte de Weierstrass :

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \end{aligned}$$

où la valeur λ est définie comme suit :

- pour $\mathrm{P} \neq \pm\mathrm{Q}$ et $\mathrm{P}, \mathrm{Q} \neq \mathcal{O}$, on pose

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1};$$

- pour P = Q, on pose

$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

- pour P = −Q, on a

$$P + Q = \mathcal{O}$$

Courbes elliptiques sur les corps finis. On considère dans la suite de cette section que K désigne un corps fini. En cryptographie, il est en général indispensable de connaître le *nombre de points*, noté #E(K), de la courbe E(K) que l'on utilise.

Proposition 6 (Frobenius)**.** *Soit* K *un corps de caractéristique* q *et* E(K) *une courbe elliptique. Soit* φ *la fonction définie comme suit :*

$$\varphi : \begin{cases} E(\bar{K}) & \to & E(\bar{K}) \\ (x,y) & \mapsto & (x^q, y^q), \text{pour } (x,y) \neq \mathcal{O} \\ \mathcal{O} & \mapsto & \mathcal{O}. \end{cases}$$

La fonction φ *est un endomorphisme de groupe appelé* endomorphisme de Frobenius.

Cet endomorphisme est utilisé pour calculer le nombre de points sur une courbe elliptique. Le nombre t satisfaisant $\#E(K) = q + 1 - t$ est appelé *trace du Frobenius de* q.

Une première estimation de l'ordre de E(K) est donnée par le théorème de Hasse.

Théorème 8 (Théorème de Hasse)**.** *Soit* E(K) *une courbe elliptique définie sur un corps* K *à* $p = q^m$ *éléments, alors on a*

$$\#E(K) = p + 1 - t \text{ avec } |t| \leq 2\sqrt{p}.$$

Une preuve de ce théorème est donnée dans [Sil86] (Théorème V.1.1).

Pour les courbes sur K de caractéristique q et pour tout entier a dans $[q + 1 - 2\sqrt{p}, q + 1 + 2\sqrt{p}]$, il existe une courbe E(K) d'ordre a.

Le problème du comptage des points sur une courbe elliptique est assez bien résolu d'un point de vue algorithmique. Il existe deux grandes familles d'algorithmes (polynomiaux) pour compter les points sur une courbe :

- les algorithmes de type Schoof [Sch95]. La complexité de l'algorithme de Schoof est en $O(log^{4+\varepsilon} q)$, en supposant l'utilisation de la multiplication rapide et en admettant certaines heuristiques. L'algorithme a ensuite été amélioré par Atkin et Elkies pour donner l'algorithme SEA puis par Morain *et al.* [CM94], [LM95].
- les algorithmes de type Satoh [Sat02]. Dans un corps à q^m éléments, l'algorithme déterministe de Satoh a un temps de calcul en $O(n^3)$ et (pour q fixé) une complexité en espace de $O(n^3)$. Des améliorations, dues entre autres à Preneel, Vandewalle, Skjernaa et Taguchi, ont permis de réduire considérablement la complexité du calcul en temps et en espace. Ces algorithmes fonctionnent bien sur des corps finis de petite caractéristique et sont, dans ce cas, plus rapides que les algorithmes précédents.

Courbes supersingulières. Les courbes supersingulières constituent un cas particulier très intéressant. En effet, dans les premières applications cryptographiques utilisant des courbes elliptiques, les courbes supersingulières ont été mises de côté en raison de l'existence d'un algorithme sous-exponentiel pour résoudre le problème du logarithme discret[1] [MOV91]. Cependant, c'est cette faiblesse même qui les a réhabilitées plus de dix ans plus tard, pour des raisons expliquées en section 1.1.2.3.

Théorème 9. *Soit* K *un corps de caractéristique* q *et* E(K) *une courbe elliptique sur* K*, alors les conditions suivantes sont équivalentes :*

- *l'anneau des endomorphismes de* End(E)*, c'est-à-dire l'anneau formé des endomorphismes de* E(K) *dans* E(K)*, est non commutatif et* $\mathrm{E}(\overline{\mathrm{K}})$ *n'a pas de point d'ordre* q*,*
- $p|t$*, où* t *est la trace du Frobenius,*
- *il existe* $k \in \mathbb{N}$ *vérifiant tel que* $\varphi^k = \pm q^{k/2}$*.*

Définition 19. *Soit* E(K) *une courbe elliptique sur* K *vérifiant l'une quelconque des propriétés énoncées au théorème 9 ; alors* E(K) *est dite* supersingulière.

Le calcul de la cardinalité de ces courbes est immédiat. D'après le théorème précédent, on a que $\#\mathrm{E}(\mathrm{K}) = p+1-t$, si E(K) est une courbe supersingulière. Il est possible, selon la valeur de t de caractériser plus précisément le groupe E(K). Par exemple, si $t^2 = p, 2p$ ou $3p$, alors E(K) est cyclique. Ceci permet alors de simplifier les opérations de groupe sur ces courbes.

1.1.2.3 Couplages et applications bilinéaires admissibles

Dès 1991, les applications bilinéaires ont été utilisées à des fins de cryptanalyse [MOV91], [MOV93]. L'optique de ces travaux était la construction d'algorithmes sous-exponentiels pour résoudre le problème du logarithme discret. Mais le premier protocole cryptographique reposant sur de telles applications n'a été proposé qu'en 2000 par Joux [Jou00]. Ce dernier a montré comment réaliser efficacement un échange de clé dans un groupe tripartite en un seul échange de communication. Depuis, la cryptographie basée sur les courbes elliptiques n'a cessé de se développer, notamment sur les courbes supersingulières, où les applications bilinéaires sont plus facilement calculables. En effet, les propriétés particulières de ces courbes autorisent des choix de clés plus courtes que dans les cryptosystèmes classiques tout en offrant les mêmes critères de sécurité. Les implémentations basées sur les courbes elliptiques nécessitent moins de puissance de calcul, ce qui les rend tout à fait adaptées à des supports comme les cartes à puces. De plus, elles permettent de spécifier un grand nombre de nouveaux cryptosystèmes jouissant de propriétés que l'on n'avait pu atteindre jusqu'alors, ou bien de manière inefficace.

Tout d'abord, donnons la définition exacte d'une telle application.

Définition 20 (Application bilinéaire admissible)**.** *Soient* $\mathbb{G}_1$ *et* $\mathbb{G}_2$ *deux groupes cycliques de même ordre premier* q*, engendrés respectivement par* P_1 *et* P_2*, et* $\mathbb{G}_\mathrm{T}$ *un troisième groupe d'ordre* q*. L'application* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ *est dite* application bilinéaire admissible *si elle vérifie les propriétés suivantes :*

[1] Le problème du logarithme discret sera détaillé dans la section 2.1.

- *bilinéarité : pour tout* $a, b \in \mathbb{Z}^2$ *et* $(\mathrm{P}, \mathrm{Q}) \in \mathbb{G}_1 \times \mathbb{G}_2$, $e(\mathrm{P}^a, \mathrm{Q}^b) = e(\mathrm{P}, \mathrm{Q})^{ab}$,
- *non-dégénérescence :* $e(\mathrm{P}_1, \mathrm{P}_2) \neq 1$,
- *calculabilité : pour tout couple* $(\mathrm{P}, \mathrm{Q}) \in \mathbb{G}_1 \times \mathbb{G}_2$, *il existe un algorithme efficace pour calculer* $e(\mathrm{P}, \mathrm{Q})$.

Note : la notion d'algorithme efficace sera détaillée dans la section suivante. Pour l'heure, nous supposons que cela signifie que l'application peut être évaluée en un temps satisfaisant pour une utilisation pratique.

Il n'existe, à notre connaissance, que deux types d'applications bilinéaires admissibles. Un seul, basé sur les couplages (et souvent appelé couplage par abus de langage), peut réellement être utilisé en cryptographie et c'est celui que nous allons présenter ici.

Dans la suite de cette section, nous définissons de manière formelle les deux couplages utilisés en cryptographie pour la construction d'applications bilinéaires admissibles. Ceci fait appel à des notions de géométrie algébrique et à des constructions particulières basées sur les courbes elliptiques. Cette section, plus théorique, peut être omise sans compromettre la compréhension de la suite du mémoire.

Nos définitions s'inspirent du livre de Menezes [Men94] sur l'utilisation des courbes elliptiques en cryptographie à clé publique ainsi que de la thèse de Laguillaumie [Lag05].

Groupe de torsion. Soit n un entier et $\mathbb{F}_q$ un corps fini de caractéristique p. On note $[n]$ la multiplication scalaire par n dans K et $\mathrm{E}[n]$ le noyau de $[n]$. L'application $[n]$ est un endomorphisme de $\mathrm{E}(\bar{\mathbb{F}}_q)$ et $\mathrm{E}[n]$ est un sous-groupe de $\mathrm{E}(\bar{\mathrm{K}})$ appelé le *groupe des points de n-torsion*. On définit alors $\mathrm{E}(\mathrm{K})[n]$ comme étant $\mathrm{E}[n] \cap \mathrm{E}(\mathrm{K})$ pour tout $\mathbb{F}_q \subset \mathrm{K} \subset \overline{\mathbb{F}_q}$.

Proposition 7 (Structure du groupe de torsion)**.** *Soient* $\mathrm{E}(\mathrm{K})$ *une courbe elliptique sur* $\mathbb{F}_q$ *de caractéristique* p *et* m *un entier non nul.*

- $\mathrm{pgcd}(p, m) = 1$ *implique* $\mathrm{E}[m] \simeq \mathbb{Z}_m \oplus \mathbb{Z}_m$.
- *s'il existe un entier* l *satisfaisant* $m = p^l$ *alors on a :*
  - $\mathrm{E}[m] \simeq \{\mathcal{O}\}$, *si* $\mathrm{E}(\mathrm{K})$ *est supersingulière,*
  - $\mathrm{E}[m] \simeq \mathbb{Z}_{p^l}$, *sinon.*

Diviseurs. La notion de diviseur est essentielle pour la construction des couplages. Nous en rappelons ici la définition et les propriétés principales.

Définition 21 (Diviseur sur une courbe)**.** *On appelle* diviseur *de* $\mathrm{E}(\mathrm{K})$, *et on note* D, *la somme formelle de points sur la courbe* $\mathrm{E}(\mathrm{K})$ *:*

$$\mathrm{D} = \sum_{\mathrm{P} \in \mathrm{E}(\mathrm{K})} n_{\mathrm{P}}(\mathrm{P}),$$

où les n_{P} *sont des entiers tous nuls sauf pour un nombre fini.*

On note $\mathrm{Div}_{\mathrm{K}}(\mathrm{E})$ l'ensemble des diviseurs de $\mathrm{E}(\mathrm{K})$.

Proposition 8. *L'ensemble* $Div_{\mathrm{K}}(\mathrm{E})$ *muni de l'addition*

$$\sum_{\mathrm{P} \in \mathrm{E}(\mathrm{K})} n_{\mathrm{P}}(\mathrm{P}) + \sum_{\mathrm{P} \in \mathrm{E}(\mathrm{K})} m_{\mathrm{P}}(\mathrm{P}) = \sum_{\mathrm{P} \in \mathrm{E}(\mathrm{K})} (n_{\mathrm{P}} + m_{\mathrm{P}})(\mathrm{P}),$$

forme un groupe. C'est le groupe libre engendré par les points de la courbe.

Définition 22. *Le* degré d'un diviseur *est égal à* $\deg(\mathrm{D}) = \sum_{p\in \mathrm{E(K)}} n_\mathrm{P}$. *On note* $\mathrm{D}iv^d(\mathrm{E})$ *les sous-groupes des diviseurs de degré d.*

Le support *d'un diviseur est* $\mathrm{supp(D)} = \{\mathrm{P}|n_\mathrm{P} \neq 0\}$.

Dans la suite, on s'intéresse plus particulièrement à $\mathrm{Div}^0(\mathrm{E})$, le sous groupe de $\mathrm{Div_K(E)}$ des diviseurs de degré 0.

On appelle F le corps des fonctions de la courbe (c'est-à-dire le corps des fractions de l'anneau $\mathrm{K}[x,y]/(f(x,y))$, où $f(x,y)$ est une équation affine de E(K)), il est noté F(E).

Définition 23 (Diviseur d'une fonction)**.** *Soit* f *une fonction de* F*, le* diviseur de la fonction f*, noté div(*f*), est défini par :*

$$div(f) = \sum_{\mathrm{P}\in \mathrm{E(K)}} \mathrm{ord_P}(f)(\mathrm{P}),$$

où $\mathrm{ord_P}(f)$ *est l'ordre de* f *au point* P*.* $ord_\mathrm{P}(f)$ *est la multiplicité de* P *si* P *est un zéro, négatif si* P *est un pôle de* f*.*

Un diviseur D *pouvant s'écrire sous la forme* $\mathrm{D} = \mathrm{div}(f)$ *est dit* principal*, c'est-à-dire que* D *est le diviseur d'une fonction de* E(K)*. On le note alors* (f)*.*

On note $\mathrm{Princ_K(E)}$ le sous groupe de $\mathrm{Div^0_K(E)}$ formé par les diviseurs principaux.

Soit (f) un diviseur principal. Soit D un diviseur $\mathrm{D} = \sum_{\mathrm{P}\in \mathrm{E(K)}} n_\mathrm{P}(\mathrm{P})$ et $f \in \bar{\mathrm{K}}(\mathrm{E})^*$ une fonction tels que (f) et D ont des supports disjoints. On définit l'évaluation de f en D et on note $f(\mathrm{D})$:

$$f(\mathrm{D}) = \prod_{\mathrm{P}\in \mathrm{supp(D)}} f(\mathrm{P})^{n_\mathrm{P}}.$$

Définition 24. *Soient* D_1 *et* D_2 *deux diviseurs. Ils sont dit* équivalents *et on note* $\mathrm{D}_1 \sim \mathrm{D}_2$ *si on a* $\mathrm{D}_1 - \mathrm{D}_2 \in \mathrm{Princ_K(E)}$ *c'est-à-dire* $\mathrm{D}_1 = \mathrm{D}_2 + \mathrm{D}iv(f)$ *qu'on a pour une certaine fonction* f*.*

Couplages. Nous avons maintenant tous les outils nécessaires pour construire les couplages à partir desquels nous pourrons définir les applications bilinéaires admissibles.

On définit d'abord le groupe des racines m-ièmes de l'unité.

Définition 25. *Pour un corps* K *quelconque et un entier* $m \in \mathbb{Z}$ *positif, on définit le groupe des* racines m-ième de l'unité $\mu_m(\mathrm{K})$ *par* $\mu_m(\mathrm{K}) = \{x \in \mathrm{K}; x^m = 1\}$*. Une clôture algébrique* $\bar{\mathrm{K}}$ *de* K *étant donnée, on note* $\mu_m = \mu_m(\bar{\mathrm{K}})$*.*

Le premier couplage que nous définissons est dû à Weil [Wei40] et a été construit en 1940 sans aucune intention cryptographique.

Soit $\mathbb{F}_q$ le corps fini à q éléments de caractéristique p et soit $m \in \mathbb{N}$ premier à p.

Définition 26 (Couplage de Weil)**.** *Soient* P *et* Q *dans* $\mathrm{E}[m]$ *et* A *et* B *deux diviseurs de degré* 0 *vérifiant* $\mathrm{A} \sim (\mathrm{P}) - (\mathcal{O})$ *et* $\mathrm{B} \sim (\mathrm{Q}) - (\mathcal{O})$*, et* $\mathrm{supp}(\mathrm{A}) \cap \mathrm{supp}(\mathrm{B}) = \emptyset$*. Soient* f_{A} *et* f_{B} *dans* $\overline{\mathbb{F}_q}(\mathrm{E})$ *satisfaisant* $\mathrm{div} f_{\mathrm{A}} = m\mathrm{A}$ *et* $\mathrm{div}(f_{\mathrm{B}}) = m\mathrm{B}$*. Le* couplage de Weil *est la fonction*

$$\begin{aligned} e_m : \mathrm{E}[m] \times \mathrm{E}[m] &\to \mu_m \\ (\mathrm{P}, \mathrm{Q}) &\mapsto e_m(\mathrm{P}, \mathrm{Q}) = f_{\mathrm{A}}(\mathrm{B}) / f_{\mathrm{B}}(\mathrm{A}). \end{aligned}$$

Proposition 9. *La fonction* e_m *possède les propriétés suivantes :*

1. *bilinéarité :*

$$\begin{aligned} \forall \mathrm{P}, \mathrm{P}_1, \mathrm{P}_2, \mathrm{Q}, \mathrm{Q}_1, \mathrm{Q}_2 \in \mathrm{E}[m], \quad & e_m(\mathrm{P}_1 + \mathrm{P}_2, \mathrm{Q}) = e_m(\mathrm{P}_1, \mathrm{Q}) e_m(\mathrm{P}_2, \mathrm{Q}), \\ & e_m(\mathrm{P}, \mathrm{Q}_1 + \mathrm{Q}_2) = e_m(\mathrm{P}, \mathrm{Q}_1) e_m(\mathrm{P}, \mathrm{Q}_2); \end{aligned}$$

2. *identité :*

$$\forall \mathrm{P} \in \mathrm{E}[m], e_m(\mathrm{P}, \mathrm{P}) = 1,$$

3. e_m *est alternée :*

$$\forall \mathrm{P}, \mathrm{Q} \in \mathrm{E}[m], e_m(\mathrm{P}, \mathrm{Q}) = e_m(\mathrm{Q}, \mathrm{P})^{-1},$$

4. *non dégénérée :*

$$\begin{aligned} (\forall \mathrm{P} \in \mathrm{E}[m], e_m(\mathrm{P}, \mathrm{Q}) = 1) &\iff \mathrm{Q} = \mathcal{O} \\ (\forall \mathrm{Q} \in \mathrm{E}[m], e_m(\mathrm{P}, \mathrm{Q}) = 1) &\iff \mathrm{P} = \mathcal{O} \end{aligned}$$

5. $\mathrm{E}[m] \subset \mathrm{E}(\mathrm{K})$ *implique* $e_m(\mathrm{P}, \mathrm{Q}) \in \mathrm{K}, \forall \mathrm{P}, \mathrm{Q} \in \mathrm{E}[m]$ *;*
6. *compatibilité : avoir* $\mathrm{P} \in \mathrm{E}[m]$ *et* $\mathrm{Q} \in \mathrm{E}[mm']$ *implique* $e_{mm'}(\mathrm{P}, \mathrm{Q}) = e_m(\mathrm{P}, m'\mathrm{Q})$*.*

Les propriétés sont démontrées dans [Sil86].

Il existe aussi un autre couplage sur les courbes elliptiques : le couplage de Tate. On note l l'ordre de q modulo m et l'on suppose de plus que m divise $\#\mathrm{E}(\mathbb{F}_{q^l})$, de telle sorte que $\mathrm{E}(\mathbb{F}_{q^l})$ contienne au moins m points de m-torsions.

Définition 27 (Couplage de Tate)**.** *Soit* $\mathrm{P} \in \mathrm{E}(\mathbb{F}_{q^l})[m]$ *et* $\mathrm{Q} \in \mathrm{E}(\mathbb{F}_{q^l})$*. Soit* $f_{\mathrm{P}} \in \mathbb{F}_{q^l}(\mathrm{E})$ *telle que* $\mathrm{div}(f_{\mathrm{P}}) = m(\mathrm{P}) - m(\mathcal{O})$ *et* $\mathrm{D} \in \mathrm{D}iv_{\mathbb{F}_{q^l}}(\mathrm{E})$ *tel que* $\mathrm{D} \sim (\mathrm{Q}) - (\mathcal{O})$*. Le* couplage de Tate t_m *est la fonction*

$$\begin{aligned} t_m : \mathrm{E}(\mathbb{F}_{q^l})[m] \times \mathrm{E}(\mathbb{F}_{q^l}) / [m]\mathrm{E}(\mathbb{F}_{q^l}) &\to \mathcal{F}_{q^l}^* / (\mathcal{F}_{q^l}^*)^m \\ (\mathrm{P}, \mathrm{Q}) &\mapsto f_m(\mathrm{P}, \mathrm{Q}) = f_{\mathrm{P}}(\mathrm{D}) \end{aligned}$$

Proposition 10. *Le couplage de Tate est bilinéaire et non dégénéré.*

Cette proposition a été démontrée dans [FR94].

Le couplage de Weil comme nous l'avons défini n'est pas une application bilinéaire admissible. En effet, pour tout $\mathrm{P} \in \mathrm{E}[m], e_m(\mathrm{P},\mathrm{P}) = 1$. De même, dans le cas du couplage de Tate, si P et Q sont dépendants, $t_m(\mathrm{P},\mathrm{Q})$ peut valoir 1 ou pas, selon la courbe. Afin de contourner cette propriété, il est possible de modifier le couplage de Weil, tout du moins dans le cas des courbes supersingulières. Pour cela, définissons d'abord le degré MOV d'une courbe.

Définition 28 (Degré MOV)**.** *Soit un entier r. On appelle* degré MOV *d'une courbe elliptique* $\mathrm{E}(\mathrm{K})$ *relativement à r le plus petit entier k vérifiant* $\mathrm{E}[r] = \mathrm{E}(\mathrm{K})[r]$.

Menezes, Okamoto et Vanstone ont montré que les courbes supersingulières ont la particularité d'avoir un degré MOV petit [MOV93]. Considérons maintenant une courbe supersingulière $\mathrm{E}(\mathrm{K})$ de degré MOV k relativement à un diviseur premier m de $\#\mathrm{E}(\mathbb{F}_q)$. Dans le cas où il existe un morphisme

$$\psi : \mathrm{E}(\mathbb{F}_{q^k}) \leftrightarrow \mathrm{E}(\mathbb{F}_{q^k})$$

non $\mathbb{F}_q$-rationnel, si P est un générateur de $\mathrm{E}(\mathrm{K})[m]$ et $\psi(p)$ engendre $\mathrm{E}[m]$, alors on peut définir le couplage de Weil modifié :

$$\begin{array}{rccl} \hat{e}[m] : & \mathrm{E}[m] \times \mathrm{E}[m] & \longrightarrow & \mathbb{F}_{q^k}^* \\ & (\mathrm{R},\mathrm{S}) & \longmapsto & \hat{e}_m(\mathrm{R},\mathrm{S}) = e_m(\mathrm{R},\psi(\mathrm{S})). \end{array}$$

Pour que cette application soit une application bilinéaire admissible, il faut aussi vérifier la propriété de calculabilité. En 1986, Miller a proposé dans [Mil86] un algorithme polynomial pour calculer les couplages de Weil dont la complexité est liée au degré MOV. C'est pourquoi il est important de travailler sur des courbes de degré MOV petit, comme les courbes supersingulières.

Un bref aperçu du calcul du couplage de Weil $\hat{e}_m(\mathrm{P},\mathrm{Q})$ est donné dans l'algorithme suivant. On note $d_{\mathrm{P},\mathrm{Q}}$ la fonction associée à la droite passant par P et Q, t_P la fonction associée à la tangente à P et v_P la fonction associée à la verticale en P.

1. `initialisation` : $n \leftarrow \mathrm{A}$, $d \leftarrow 1$, $\mathrm{R} \leftarrow \mathrm{P}$, $\mathrm{S} \leftarrow \mathrm{Q}$;
2. `Pour` i de $|m| - 2$ à 0 `faire`
   (a) $\mathrm{R}' \leftarrow 2\mathrm{R}$, $\mathrm{S}' \leftarrow 2\mathrm{S}$;
   (b) $n \leftarrow n^2.t_\mathrm{R}(\mathrm{Q}).v_{\mathrm{S}'}(\mathrm{P}).y(\mathrm{S})$;
   (c) $d \leftarrow d^2.v_{\mathrm{R}'}(\mathrm{Q}).t_\mathrm{S}(\mathrm{P}).y(\mathrm{R})$;
   (d) `Si` $m_i = 1$ et $i > 0$ `alors`
      i. $\mathrm{R} \leftarrow \mathrm{R}' + \mathrm{P}$, $\mathrm{S} \leftarrow \mathrm{S}' + q$;
      ii. $n \leftarrow n.d_{\mathrm{R}',\mathrm{P}}(\mathrm{Q}).v_\mathrm{S}(\mathrm{P}).(x(\mathrm{Q}) - d(\mathrm{S}'))$;
      iii. $d \leftarrow d.v_\mathrm{R}(\mathrm{Q}).d_{\mathrm{S},\mathrm{Q}}(\mathrm{P}).(x(\mathrm{P}) - s(\mathrm{S}'))$;
   (e) `Sinon`
      i. $\mathrm{R} \leftarrow \mathrm{R}'$, $s \leftarrow \mathrm{S}'$;

(f) `si` $n = 0$ ou $d = 0$ `alors` renvoyer 1 ;

3. renvoyer n/d.

Le couplage de Tate, quant à lui, définit directement une application bilinéaire et il existe des algorithmes efficaces pour l'évaluer (*cf.* [Mil86], [GHS02]). Le calcul du couplage de Tate est à peu près deux fois plus rapide que celui du couplage de Weil. De plus, la rapidité du calcul dépend de la taille de l'extension du corps de base. Plus celle-ci est petite, plus le calcul est rapide. Entre les deux couplages, l'extension correspondante au couplage de Tate est plus petite que celle du couplage de Weil. Un autre point est que le couplage de Tate peut être utilisé sur plus de courbes que le couplage de Weil et même sur des courbes non elliptiques.

En règle générale, les groupes sont choisis de la manière suivante : $\mathbb{G}_1$ est un sous-groupe d'un groupe de points d'une courbe elliptique sur un corps fini. $\mathbb{G}_2$ est alors un sous-groupe d'un groupe multiplicatif d'un corps fini. A l'heure actuelle, les paramètres sont choisis tels que $\mathbb{G}_1$ contient environ 2^{256} éléments et $\mathbb{G}_2$ est un sous-groupe de $\mathbb{F}_{t^r}$ où r est le degré MOV de la courbe et t^r est d'environ 1024 bits.

Nous utiliserons dans la suite de ce mémoire les couplages comme des « boîtes noires ». Cependant, comme l'ont rappelé Galbraith, Paterson et Smart dans [GPS06a], il est important, lors de la spécification d'un cryptosystème utilisant des couplages, de vérifier si les courbes utilisées offrent bien les fonctionnalités voulues (l'existence d'un homomorphisme calculable entre $\mathbb{G}_2$ et $\mathbb{G}_1$, la possibilité d'évaluer rapidement le couplage, par exemple). En effet, il peut être très rapide d'émettre des hypothèses fausses et par conséquent de construire des schémas moins efficaces que prévus ou alors non implémentables.

1.2 Machines de Turing

Nous commençons tout d'abord par une description des machines de Turing et de certaines de leurs variantes. Ces machines nous serviront, par la suite, à décrire les attaquants contre nos cryptosystèmes.

1.2.1 Définition

Les machines de Turing sont des machines purement conceptuelles mais qui peuvent servir à simuler un ordinateur. Elles ont été introduites par Turing en 1937 [Tur37].

Une *machine de Turing* consiste en les cinq éléments ci-dessous :

- un ruban, avec une extrémité gauche, infini à droite, divisé en cases de même taille ;
- un ensemble fini de symboles, par exemple $\{0, 1, s, d, f\}$, avec 0 et 1 pour la numérotation binaire, s pour séparer deux expressions, d pour le début du ruban et f pour signifier que ce qui est à droite n'a plus d'importance. Ils seront inscrits et lus sur le ruban à raison d'un symbole par case ;
- une tête de lecture/écriture qui se déplace sur le ruban. À chaque impulsion, la tête peut soit rester sur place, soit reculer (si possible) ou avancer, d'une case à la fois. La tête a le droit de lire ou d'écrire dans la case qu'elle est en train de sonder. Pour rester concret, on peut penser que c'est la bande (et non la tête) qui bouge ;

- un ensemble fini d'états. Ces états permettent de distinguer plusieurs situations possibles. Notamment l'état D représente l'état de départ, fourni par les « données initiales » et l'état F marque la fin du "programme" et dont le résultat peut être lu par un observateur extérieur ;
- un ensemble fini d'instructions : à chaque impulsion, en fonction du symbole c lu par la tête, en fonction de l'état courant S, la tête écrit un nouveau symbole 'c', effectue un déplacement noté -1 (gauche), $+1$ (droite) ou 0 (immobile) et passe à l'état 'S'.

Plus formellement on définit une machine de Turing de la manière suivante :

Définition 29 (Machine de Turing)**.** *Une* Machine de Turing *est définie par un 7-uplet* $(\Sigma, \mathcal{Q}, \sigma, \delta, \Delta, q_0, \mathcal{F})$ *où :*
- Σ *est l'alphabet,*
- $\mathcal{Q}$ *est l'ensemble d'états,*
- $\sigma : \mathcal{Q} \times \Sigma \to \Sigma$ *est la fonction d'écriture,*
- $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ *est la fonction de changement d'état,*
- $\Delta : \mathcal{Q} \times \Sigma \to \{\mathrm{G}, \mathrm{D}\}$ *est la fonction de déplacement,*
- $\mathcal{F} \subset \mathcal{Q}$ *est l'ensemble d'états finaux.*

1.2.2 Complexités

On peut associer différentes complexité à une machine de Turing donnée ; relativement à une entrée $\mathcal{I}$. Nous ne nous intéressons dans ce mémoire qu'à la *complexité en temps*, notée $\mathrm{T}_{\mathrm{M}}(\mathcal{I})$. Elle est donnée par le nombre de pas élémentaires nécessaires pour passer de l'état initial à l'état final.

La complexité d'un calcul est dite *polynomiale* si elle est majorée par un polynôme en la taille n des données initiales. On parle de *machine de Turing fonctionnant en temps polynomial* et on la note $\mathrm{PTM}(n)$. Plus formellement on définit cette complexité de la manière suivante :

Définition 30 (Complexité en temps polynomiale)**.** *Soit* $\mathcal{M}$ *une machine de Turing probabiliste. Notons* $\mathrm{T}_{\mathcal{M}}(n) = \sup\{\mathrm{T}_{\mathcal{M}}(\mathcal{I}), |\mathcal{I}| = n\}$.

La complexité en temps de $\mathcal{M}$ *est dite* polynomiale *si*

$$\exists n_0 \in \mathbb{N}, \exists c \in \mathbb{N}^*, \forall n \geq n_0, \mathrm{T}_{\mathcal{M}}(n) \leq n^c.$$

En pratique, il est souhaitable qu'une machine de Turing fonctionne en temps polynomial, autrement dit que son temps d'exécution soit raisonnable à notre échelle.

1.2.3 Variantes des machines de Turing

Les machines de Turing peuvent être modifiées de diverses manières afin d'obtenir des machines plus puissantes et plus rapides.

Dans la définition 29, la machine est décrite de manière déterministe : elle est incapable de faire des choix lors de son exécution. Ainsi, pour une entrée donnée elle donnera toujours la même sortie. Il est possible de changer de manière plus radicale le fonctionnement d'une telle machine en brisant son déterminisme.

Machines de Turing probabilistes. Dans ce cas de figure, on suppose que la machine de Turing dispose d'une source de caractères aléatoires pouvant être utilisée durant le fonctionnement de la machine. On appelle ces machines, des *machines de Turing probabilistes*. Plus concrètement, une telle machine dispose d'un ruban qu'elle peut utiliser à chaque étape pour tirer au sort et adapter son comportement en fonction du résultat du tirage. Ce second ruban, dit *ruban aléatoire* et noté ω_{M}, est supposé infini à droite et contient une suite infinie de caractères notée s, chaque caractère étant choisi uniformément aléatoirement dans l'alphabet Σ. La complexité en temps d'une telle machine $\mathcal{M}$ relativement à une entrée $\mathcal{I}$ et la suite s est notée $\mathrm{T}_{\mathcal{M},s}(\mathcal{I})$. On pose alors $\mathrm{T}_{\mathcal{M},s}(n) = \sup\{\mathrm{T}_{\mathcal{M},s}(\mathcal{I}), |\mathcal{I}| = n\}$.

Considérons désormais une machine de Turing probabiliste qui s'arrête (c'est-à-dire que le nombre de pas élémentaires effectués est borné) pour toute entrée $\mathcal{I}$ de taille n et toute suite s écrite sur $\omega_{\mathcal{M}}$. Notons $k(n)$ le nombre maximum de caractères aléatoires utilisés pour une entrée de taille n et notons $\mathrm{S}_{k(n)}$ l'ensemble des séquences de $k(n)$ caractères uniformément distribués sur Σ^{kn}. La complexité en temps de $\mathcal{M}$, notée $\mathrm{T}_{\mathcal{M}}(n)$ relativement à une entrée de taille n est alors définie par

$$\mathrm{T}_{\mathcal{M}}(n) = \frac{1}{|\mathrm{S}_{k(n)}|} \sum_{s \in \mathrm{S}_{k(n)}} \mathrm{T}_{\mathcal{M},s}(n).$$

Définition 31 (Complexité en temps polynomiale)**.** *Soit $\mathcal{M}$ une machine de Turing probabiliste, munie d'un ruban aléatoire $\omega_{\mathcal{M}}$. La complexité en temps de $\mathcal{M}$ est dite* polynomiale *si*

$$\exists n_0 \in \mathbb{N}, \exists c \in \mathbb{N}^*, \forall n \geq n_0, \mathrm{T}_{\mathcal{M}}(n) \leq n^c.$$

Une machine de Turing probabiliste de complexité en temps polynomial est appelée *machine de Turing probabiliste polynomiale* et est notée PPTM (*Probabilistic Polynomial time Turing Machine*).

Les machines de Turing probabilistes permettent de définir la notion de réduction de problèmes.

Définition 32 (Réduction de problème)**.** *Soient $\mathbb{P}_1$ et $\mathbb{P}_2$ deux problèmes mathématiques. Le problème $\mathbb{P}_1$ se* réduit *au problème $\mathbb{P}_2$ si l'existence d'une PPTM résolvant le problème $\mathbb{P}_2$ permet de construire une PPTM résolvant le problème $\mathbb{P}_1$. Dans un tel cas, $\mathbb{P}_2$ est dit au moins aussi difficile que $\mathbb{P}_1$ ou encore $\mathbb{P}_1$ est dit au moins aussi facile que $\mathbb{P}_2$.*

Les problèmes $\mathbb{P}_1$ et $\mathbb{P}_2$ sont équivalents si $\mathbb{P}_1$ est à la fois au moins aussi facile et au moins aussi difficile que $\mathbb{P}_2$.

Machines de Turing interactives Les *machines de Turing interactives* sont des machines pourvues en plus d'un ruban de communication en lecture/écriture. Une machine de Turing interactive disposant d'un ruban aléatoire est appelée une *machine de Turing probabiliste interactive*.

Deux machines de Turing interactives partageant les mêmes données initiales et un même ruban de communication forment un *protocole interactif*.

Définition 33 (Protocole interactif)**.** *Un* protocole interactif *est un ensemble de deux machines de Turing interactives $\mathcal{P}$ et $\mathcal{V}$, appelées prouveur et vérificateur, partageant un même ruban de communication qui leur permet d'échanger des données. On note un tel protocole $(\mathcal{P}, \mathcal{V})$.*

Par la suite, la sortie d'un protocole interactif entre $\mathcal{P}$ et $\mathcal{V}$ sera noté $\langle \mathcal{P}; \mathcal{V} \rangle$.

Machines de Turing à oracle. Il existe aussi un autre type de machine : les *machines de Turing à oracle*. Ces machines peuvent interroger un « oracle » capable de répondre à une question de type donné, pour un coût constant. Si $\mathcal{A}$ est une telle machine ayant accès à l'oracle $\mathcal{O}$ nous la noterons $\mathcal{A}(\mathcal{O})$.

Une machine de Turing à oracle disposant d'un ruban aléatoire est appelée une *machine de Turing probabiliste à oracle*.

Dans la suite de ce mémoire, tous les algorithmes seront représentés par des machines de Turing. En particulier,
- un *algorithme déterministe* sera représenté par une PTM,
- un *algorithme probabiliste* sera représenté par une PPTM,
- un *algorithme efficace* sera représenté par une machine de Turing interactive (probabiliste) fonctionnant en temps polynomial,
- les *adversaires* (aussi appelés attaquants) seront représentés par des machines de Turing (probabilistes) (à oracle) fonctionnant en temps polynomial.

1.3 Fonctions particulières

Nous définissons dans cette section des fonctions particulières adaptées à nos besoins cryptographiques.

1.3.1 Fonctions à sens unique

La cryptographie moderne repose en grande partie sur la notion de fonction à sens unique, c'est-à-dire de fonction facilement calculable mais difficilement inversible.

La définition de fonction à sens unique repose sur celle des fonctions négligeables. Une fonction négligeable est une fonction qui est asymptotiquement plus petite que l'inverse de n'importe quel polynôme. Les fonctions négligeables servent aussi à quantifier la sécurité des cryptosystèmes, comme nous le verrons par la suite.

Définition 34 (Fonction négligeable)**.** *Une fonction $\nu : \mathbb{N} \to \mathbb{R}^+$ est dite* négligeable *si, pour tout entier positif c, il existe un entier k_c satisfaisant*

$$\nu(k) < k^{-c}$$

pour tout $k \geq k_c$.

Les fonctions négligeables permettent de définir les notions de probabilité négligeable et écrasante.

Définition 35 (Probabilité négligeable, écrasante)**.** *Une probabilité* P *dépendant d'un paramètre* k*, appelé* paramètre de sécurité *est dite* négligeable *si* P *est une fonction négligeable de* k*.*

Une probabilité P *dépendant d'un paramètre de sécurité* k *est dite* écrasante *si la probabilité* $1 - \mathrm{P}$ *est négligeable.*

Définition 36 (Fonction à sens unique)**.** *Une fonction* $f : \{0,1\}^* \to \{0,1\}^*$ *est dite à* sens unique *si elle vérifie les deux conditions suivantes :*

- *il existe un algorithme efficace qui prenant* $x \in \{0,1\}^*$ *en entrée renvoie* $f(x)$*,*
- *pour tout algorithme efficace* $\mathcal{A}$*, pour tout* k *suffisamment grand, la probabilité*

$$\Pr[x \xleftarrow{\mathrm{R}} \{0,1\}^k; y \leftarrow f(x); x' \leftarrow \mathcal{A}(1^k, y) : f(x') = f(x)]$$

est négligeable.

Autrement dit, il n'existe pas d'algorithme efficace pour inverser la fonction f*.*

Fonction à trappe. Bien souvent, on utilise en cryptographie une classe particulière de fonctions à sens unique : les fonctions à sens unique à trappe. Ces fonctions possèdent une donnée particulière dont la connaissance permet de les calculer efficacement dans le sens difficile.

Définition 37 (Fonction à trappe)**.** *Une fonction* $f : \{0,1\}^* \to \{0,1\}^*$ *est dite* à trappe *si elle est à sens unique et s'il existe un algorithme efficace* $\mathcal{B}$ *et un entier* $c \geq 0$ *tel que, pour tout* $k \in \mathbb{N}$ *il existe une suite de bits* $t_k \in \{0,1\}^*$ *avec* $|t_k| \leq k^c$ *et pour tout* $x \in \{0,1\}^*$*,*

$$\mathcal{B}(1^k, f(x), t_k) = x' \text{ et } f(x) = f(x').$$

Fonctions de hachage Les fonctions de hachage sont une autre classe de fonctions à sens unique très fréquemment utilisées en cryptographie. Initialement utilisées en informatique pour compresser les données, elles permettent de plus de plonger un message quelconque dans un corps ou un ensemble particulier (une courbe elliptique, le groupe multiplicatif d'un corps fini, ...).

On peut définir, de manière informelle, une fonction de hachage comme étant une application

$$\mathrm{H} : \{0,1\}^* \to \mathrm{F}$$

où F est un ensemble tel que $|\mathrm{F}| \leq 2^k$ avec k suffisamment petit. Comme nous l'avons dit, F peut être un corps fini ou plus simplement l'ensemble $\{0,1\}^k$.

En cryptographie il est nécessaire, pour des raisons de sécurité, de s'assurer que les fonctions de hachage utilisées respectent certaines propriétés. On parle alors de *fonction de hachage cryptographiquement sûre.* Toutes les fonctions de hachage utilisées dans ce mémoire seront de ce type. Nous en donnons ici une définition informelle.

Définition 38 (Fonction de hachage cryptographiquement sûre)**.** *Une fonction de hachage* H *est dite* cryptographiquement sûre *si elle respecte les propriétés suivantes :*

résistance à la préimage *:* H *est à sens unique,*

résistance à la seconde préimage *: il n'existe pas d'algorithme efficace permettant, étant donné un message* $m \in \{0,1\}^*$*, de trouver un message* $m' \neq m$ *tel que* $\mathrm{H}(m') = \mathrm{H}(m)$*,*

résistance aux collisions *: il n'existe pas d'algorithme efficace permettant de trouver deux éléments différents* m *et* m' *dans* $\{0,1\}^*$ *tels que* $\mathrm{H}(m) = \mathrm{H}(m')$*.*

1.3.2 Fonctions pseudo-aléatoires

Les fonctions pseudo-aléatoires ont été introduites par Goldreich, Goldwasser et Micali [GGM86] . Informellement, une fonction est dite *pseudo-aléatoire* s'il est difficile de distinguer sa sortie d'une valeur aléatoire.

De manière plus formelle, on définit une famille de fonctions pseudo-aléatoires de la manière suivante :

Définition 39 (Famille de fonctions pseudo-aléatoires)**.** *Soient* m *et* ℓ *deux fonctions polynomiales,* k *un paramètre de sécurité et soit* $\mathcal{R}_k$ *l'ensemble des fonctions de* $\{0,1\}^{m(k)} \to \{0,1\}^{\ell(k)}$*.*

Soit $\mathcal{F}_k = \{\mathrm{F}_s\}_{s\in\{0,1\}^k}$ *l'ensemble des fonctions indexées par une* graine s *(*$\mathcal{F}_k \subseteq \mathcal{R}_k$*). Alors* $\mathcal{F}_k$ *est une famille de fonctions pseudo-aléatoires si elle possède les propriétés suivantes :*

efficacité *: étant donnés* s *et* x*, il existe un algorithme efficace pour évaluer* $\mathrm{F}_s(x)$*,*

pseudo-aléatoire *: pour tout adversaire* $\mathcal{A}$ *dont le nombre de requêtes à* F *est polynômialement limité, pour tout* k *la probabilité*

$$\left| \Pr_{s\in\{0,1\}^k}[\mathcal{A}(\mathrm{F}_s) = 1] - \Pr_{\mathrm{R}\in\mathcal{R}_k}[\mathcal{A}(\mathrm{R}) = 1] \right|$$

est négligeable.

Par abus de langage, on dira que la fonction F, ou la fonction F_s pour s fixé, est pseudo-aléatoire.

1.3.3 Engagements

Un engagement, comme son nom l'indique, permet à un prouveur de s'engager auprès d'un vérificateur, lors d'un protocole interactif, sur une valeur sans la dévoiler de prime abord. Cette valeur peut être, si besoin est, dévoilée par la suite par le prouveur. Ainsi, le vérificateur a l'assurance qu'une fois l'engagement publié, le prouveur ne peut plus changer d'avis sur la valeur contenue dans l'engagement.

Plus formellement, un engagement se définit de la manière suivante.

Définition 40 (Engagement)**.** *Un protocole d'*engagement *se construit à l'aide des algorithmes suivants :*

- Setup*, algorithme probabiliste de génération des paramètres et des clés, qui prend en entrée un entier* k *et renvoie des paramètres publics* $params$*.*
  $(params) \leftarrow \mathsf{Setup}(1^k)$*.*

- Commit, *algorithme d'engagement qui prend en entrée les paramètres publics, un message m et la clé publique du schéma. Il renvoie un engagement* C *du message m et des informations r de désengagement.*
  $(r, \mathrm{C}) \leftarrow \mathsf{Commit}(params, m)$.
- Decommit, *algorithme de désengagement qui prend en entrée un engagement* C*, le message m associé et les informations associées r. Il renvoie 1 si* C *est bien un engagement sur m, sinon il renvoie 0.*
  $(\{0,1\} \leftarrow \mathsf{Decommit}(\mathrm{C}, m, r)$.

Un schéma d'engagement doit satisfaire deux notions de sécurité.

indistinguabilité (hiding) : le vérificateur ne doit apprendre aucune information sur le message m s'il n'a en sa possession que l'engagement c. Autrement dit, un ensemble d'engagements sur une valeur donnée x doit être indistinguable d'un ensemble d'engagements sur une autre valeur x'.

résistance aux collisions (binding) : le prouveur ne doit pas pouvoir changer la valeur de m après s'être engagé sur elle, c'est-à-dire que pour tout m, pour tout (r, C), il n'existe pas d'algorithme efficace capable de trouver $m' \neq m$ et r' satisfaisant $\mathsf{Decommit}(\mathrm{C}, m, r) = \mathsf{Decommit}(\mathrm{C}, m', r') = 1$.

1.4 Primitives cryptographiques

Cette section vise à définir les principales primitives cryptographiques dont nous nous servirons ensuite. Nous donnons la définition formelle de chaque primitive ainsi que ses propriétés de sécurité. Dans ce but, nous caractérisons aussi les attaques auxquelles la primitive peut être soumise.

Les définitions données se restreignent délibérément au contexte de la cryptographie à clé publique.

1.4.1 Chiffrement

Le chiffrement est un procédé cryptographique permettant à un utilisateur A de transmettre des données de manière confidentielle à un utilisateur B. Pour chiffrer son message, l'émetteur A utilise la clé publique du destinataire B. Pour déchiffrer le message reçu, B utilise sa clé privée.

Ainsi, n'importe qui est à même d'envoyer un message chiffré, à la seule condition de connaître la clé publique du destinataire. Mais seul ce dernier peut retrouver le contenu du message clair initial, à partir du message chiffré.

1.4.1.1 Définition

On définit formellement un schéma de chiffrement de la manière suivante :

Définition 41 (Schéma de chiffrement à clé publique)**.** *Un* schéma de chiffrement à clé publique *se construit à l'aide des algorithmes suivants :*

- Setup, *algorithme probabiliste de génération des paramètres du système, qui prend en entrée un entier k et renvoie les paramètres publics $params$ et une paire de clés publique/privée (pk, sk).*
  $(params, pk, sk) \leftarrow \mathsf{Setup}(1^k)$.
- Encrypt, *algorithme déterministe ou probabiliste de chiffrement, qui prend en entrée les paramètres publics, un message $m \in \{0,1\}^*$ et une clé publique pk, et qui retourne un chiffré c de m.*
  $c \leftarrow \mathsf{Encrypt}(params, m, pk)$.
- Decrypt, *algorithme déterministe de déchiffrement, qui prend en entrée les paramètres publics, un chiffré c et une clé secrète sk, et qui retourne le message m dont c est le chiffré.*
  $m \leftarrow \mathsf{Decrypt}(params, c, sk)$.

1.4.1.2 Sécurité

La première propriété de sécurité attendue d'un schéma de chiffrement est sa validité : un message chiffré avec une clé publique pk doit être correctement déchiffré avec la clé privée associée sk. La seconde est que la seule connaissance du message chiffré, sans la connaissance de la clé privée, ne révèle aucune information que ce soit sur le message initial.

De manière plus précise, pour étudier la sécurité d'un schéma de chiffrement (plus généralement de tout schéma cryptographique), nous avons besoin de préciser les buts que peut chercher à atteindre un attaquant, ainsi que les moyens dont il susceptible de disposer.

L'adversaire commence par cibler l'utilisateur qu'il veut attaquer et se procure sa clé publique. Il peut alors poursuivre plusieurs objectifs. S'il souhaite retrouver certains messages clairs correspondant à certains messages chiffrés reçus par cet utilisateur, alors, il existe deux attaques possibles :

cassage total : l'adversaire déduit la clé privée de l'utilisateur de sa clé publique.

cassage du sens unique de la fonction de chiffrement : l'adversaire retrouve le message clair à partir d'un chiffré mais sans connaître forcément la clé secrète. Il n'est alors pas forcément capable de déchiffrer tous les messages.

L'adversaire peut aussi être moins ambitieux et chercher à obtenir des informations sur le message clair sans pour autant parvenir à le reconstituer entièrement. Il s'attaque alors à l'une des propriétés suivantes :

non-malléabilité - NM : l'adversaire, connaissant un chiffré c d'un message m inconnu, est capable de produire un chiffré c' d'un message m' inconnu, tels que m et m' soient liés par certaine une relation.

sécurité sémantique : l'adversaire parvient à obtenir des informations sur le texte clair à partir du chiffré seul.

indistinguabilité - IND : l'adversaire est capable de distinguer les chiffrés de deux messages différents.

En fonction des ressources auxquelles il a accès, l'attaquant peut mener les attaques suivantes :

attaque à textes clairs choisis (Chosen Plaintext Attack - CPA) : l'attaquant peut obtenir les messages chiffrés correspondants aux messages clairs de son choix. En cryptographie à clé publique, l'attaquant est toujours en position de mener ce genre d'attaque puisque la clé de chiffrement est publique.

attaque à chiffrés choisis (Chosen Ciphertext Attack -CCA1) : l'attaquant a accès à un oracle de déchiffrement, pendant une période déterminée de son attaque, lui permettant d'obtenir les messages clairs correspondants aux chiffrés de son choix.

attaque à chiffrés choisis adaptative (Chosen Ciphertext Attack Adaptative - CCA2) : l'attaquant a accès à un oracle de déchiffrement pendant tout le temps de son attaque. On dit que cette attaque est adaptative car l'attaquant peut adapter ses requêtes à tous moments de son attaque.

Dans la suite de ce mémoire on dira qu'un schéma ayant la propriété de sécurité XX relativement à une attaque YY est XX-YY. Par exemple, un schéma de chiffrement indistinguable contre des attaques à messages choisis adaptatives sera dit IND-CCA2.

De nombreuses relations entre les différentes propriétés de sécurité ont été montrées. Par exemple, les propriétés d'indistinguabilité et de sécurité sémantique sont équivalentes. Nous renvoyons le lecteur à [BDPR98], [DDN00] et [Poi02] pour plus de précisions.

La construction de schémas de chiffrement repose souvent sur l'existence d'une fonction *à trappe*. Cependant, certaines de ces fonctions ne garantissent pas forcément l'indistinguabilité du schéma. C'est pourquoi, on utilisera plutôt la notion de prédicat *hard-core* pour des fonctions à trappe [BM84] ou de prédicats à trappe [GM82], [GM84]. Intuitivement, un prédicat s est dit *hard-core* s'il est difficile, étant donné $f(x)$, avec f une fonction à sens unique, de le trouver autrement qu'en le donnant au hasard. Un prédicat à trappe est un prédicat à sens unique pour lequel il existe, pour tout $k \in \mathbb{N}$, une information de taille t_k majorée par un polynôme, dont la connaissance permet le calcul en temps polynomial de $\mathrm{B}(x)$ pour tout x vérifiant $|x| \leq k$, où B est un prédicat à sens unique.

Chiffrement RSA. Afin d'illustrer la notion de chiffrement à clé publique, nous présentons maintenant l'exemple du chiffrement RSA. C'est le premier schéma de chiffrement à clé publique qui ait été inventé. Il tire son nom de celui de ses concepteurs Rivest, Shamir et Adleman, et a été publié en 1978 [RSA78].

Setup

Les paramètres sont définis comme suit :

- k est un paramètre de sécurité,
- p, $q \in \mathbb{N}$ sont deux entiers premiers sûrs, de taille k,
- $n = pq$,
- e est un entier premier avec $\phi(n) = (p-1)(q-1)$, où ϕ est l'indicatrice d'Euler,
- et $d = e^{-1} \bmod \phi(n)$.

La clé publique de chiffrement est donnée par (n, e), la clé privée par d.

Rappel : e étant premier avec $\phi(n)$, le nombre d existe bien et est calculable à l'aide de l'algorithme d'Euclide étendu.

Encrypt
Soit $m \in [0, n-1]$, le message que A souhaite chiffrer. Afin de chiffrer m, A calcule $c \equiv m^e \mod n$.

Decrypt
Pour déchiffrer c, B effectue le calcul suivant :

$$c^d \mod n \equiv (m^e)^d \mod n \equiv m^{ed} \mod n$$

Comme $ed \equiv 1 \mod \phi(n)$ il existe $l \in \mathbb{N}$ tel que :

$$ed = 1 + l\phi(n), \text{avec } l \in \mathbb{N}.$$

D'où :

$$m^{ed} \mod n \equiv m \cdot m^{l\phi(n)} \mod n \equiv m \cdot (m^{\phi(n)})^l \mod n$$

Dans le cas où m est premier avec n, on obtient d'après le théorème d'Euler :

$$\mathrm{C}^d \equiv m \mod n.$$

Dans le cas général la vérification vient du fait que

$$ed \equiv 1 \mod (p-1)(q-1) \text{ donc } ed \equiv 1 \mod (p-1) \text{ et } ed \equiv 1 \mod (q-1).$$

D'après le Petit Théorème de Fermat (*cf.* théorème 2), on obtient

$$m^{p-1} \equiv 1 \mod p \text{ et } m^{q-1} \equiv 1 \mod q,$$

d'où

$$m^{(p-1)(q-1)} \equiv 1 \mod p \text{ et } m^{(p-1)(q-1)} \equiv 1 \mod q.$$

Les entiers p et q étant premiers entre eux, le Théorème des Restes Chinois (*cf.* théorème 4) nous permet d'écrire $m^{\varphi(n)} = m^{(p-1)(q-1)} \equiv 1 \mod pq$.

Finalement, on obtient

$$\mathrm{C}^d \equiv m \cdot m^{k\varphi(n)} \equiv m \mod n.$$

La schéma tel que présenté ici n'est pas sémantiquement sûr et est malléable. De plus, il est vulnérable à une attaque adaptative à chiffrés choisis. En effet, considérons deux messages clairs m_1 et m_2 et soient c_1 et c_2 leurs chiffrés respectifs. On a alors $(m_1 m_2)^e = c_1 c_2 \mod n$, autrement dit le chiffré du message $m = m_1 m_2$ est donné par $c = c_1 c_2 \mod n$. L'adversaire peut utiliser cette multiplicativité de RSA pour attaquer le schéma comme suit. Soit c le message à déchiffrer. L'adversaire choisit $x \in \mathbb{Z}_n$ aléatoirement. Il demande à son oracle de déchiffrer le message $y \equiv cx^e \mod n$. Il reçoit en réponse $z \equiv y^d \mod n$. Comme $z \equiv (cx^e)^d \equiv c^d x^{ed} \equiv mx \mod n$, l'adversaire est capable de calculer $z/x \equiv m \mod n$.

Il est cependant possible de le transformer en schéma probabiliste, non malléable face à une attaque adaptative à chiffrés choisis, comme l'ont montré [BR94], [Sho01], [FOPS01].

Il existe, bien évidemment, de nombreux autres schémas de chiffrement. Nous présenterons, par la suite, des constructions plus spécifiques à nos besoins.

1.4.2 Signature

Les signatures numériques utilisées en cryptographie ont les mêmes objectifs que les signatures manuscrites, c'est-à-dire garantir l'authenticité et l'intégrité d'un document. Pour signer un message, l'utilisateur A utilise sa clé privée et le destinataire B vérifie la signature à l'aide de la clé publique de A.

1.4.2.1 Définition

De manière plus formelle, on définit une signature numérique de la façon suivante :

Définition 42 (Signature numérique)**.** *Un* schéma de signature numérique à clé publique *est défini par les algorithmes suivants :*

- Setup*, algorithme probabiliste de génération des paramètres du système, qui prend en entrée un entier k et retourne les paramètres publics $params$, ainsi qu'une paire de clés publique/privée (pk, sk).*
  $(params, pk, sk) \leftarrow \mathsf{Setup}(1^k)$.
- Sign*, algorithme déterministe ou probabiliste de signature, qui prend en entrée les paramètres publics, un message $m \in \{0,1\}^*$ et une clé secrète sk, et qui retourne une signature σ de m.*
  $\sigma \leftarrow \mathsf{Sign}(params, m, sk)$.
- Verify*, algorithme déterministe de vérification, qui prend en entrée les paramètres publics, une signature σ, un message m et une clé publique pk, et qui vérifie que σ est bien une signature de m. Il renvoie 1 en cas de succès, 0 sinon.*
  $\{0,1\} \leftarrow \mathsf{Verify}(params, \sigma, m, pk)$.

1.4.2.2 Sécurité

Tout comme nous l'avons fait pour le chiffrement, afin d'expliciter la sécurité d'un schéma de signature, nous définissons les moyens que possède l'attaquant ainsi que les buts qu'il poursuit.

Il existe plusieurs moyens pour un attaquant d'attaquer un schéma de signature numérique :

cassage total : comme pour un schéma de chiffrement, l'attaquant retrouve la clé secrète du signataire.

forge universelle : l'attaquant est capable de signer n'importe quel message sans connaître la clé secrète.

forge existentielle : l'attaquant réussit à forger une signature valide sur un message de son choix.

forge sélective : l'attaquant réussit à forger une signature valide sur un message choisi dans une liste.

Pour atteindre son objectif, l'attaquant dispose de plusieurs modes d'attaques différents, selon les ressources auxquelles il a accès. On distingue trois sortes d'attaques :

attaque sans message : l'attaquant est uniquement en possession de la clé publique du signataire.

attaque à messages connus (Known Message Attack - KMA) : l'attaquant connaît la clé publique du signataire et une liste de message/signature associés à cette clé.

attaque à messages choisi (Chosen Message Attack Adaptive - CMA) : l'attaquant connaît la clé publique du signataire, et est capable d'obtenir des signatures sur des messages de son choix. Il est ainsi capable d'adapter le choix de ses messages en fonction des signatures qu'il a obtenues.

D'un point de vue théorique, il suffit d'une fonction à sens unique pour construire un schéma de signature, comme l'a montré Rompel dans [Rom90]. Cependant, ce résultat ne permet pas d'obtenir de construction efficace. Le résultat le plus souvent utilisé est celui du *hash and sign*, qui consiste à hacher tout d'abord le message avant de le signer. Ceci permet de diminuer la taille du message et d'augmenter l'efficacité des algorithmes de signature. Cela améliore aussi la sécurité contre les forges existentielles. En revanche, il est nécessaire, pour garantir la sécurité des schémas, de faire des hypothèses sur ces fonctions de hachage et de supposer qu'elles se comportent comme des fonctions aléatoires, comme nous le verrons par la suite.

Signature RSA. La signature RSA a été construite en même temps que le schéma de chiffrement RSA [RSA78]. Son principe extrêmement simple en fait une signature couramment utilisée. Elle repose sur le principe du *hash and sign* et sa sécurité nécessite ce qu'on appelle des fonctions FDH (pour *Full Domain Hash* [BR93]).

Soit un utilisateur A souhaitant signer un message de son choix $m \in \{0,1\}^*$. Setup
Les paramètres sont définis comme suit :

- k est un paramètre de sécurité,
- p, $q \in \mathbb{N}$ sont deux entiers premiers sûrs,
- $n = pq$,
- e, un entier premier avec $\phi(n) = (p-1)(q-1)$, où ϕ est l'indicatrice d'Euler,
- et $d = e^{-1} \bmod \phi(n)$,
- $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_n$ est une fonction de hachage.

La clé publique de chiffrement est donnée par (n, e), la clé privée par d.

Sign
Afin de signer le message, A effectue les calculs suivants :

- $m' = \mathrm{H}(m)$,
- $\sigma = m'^d \mod n$,
- la signature est donnée par σ.

Verify
Pour vérifier la signature, B vérifie l'égalité

$$\mathrm{H}(m) \stackrel{?}{=} \sigma^e \mod n.$$

La consistance du schéma est donnée par l'égalité suivante

$$\sigma^e \mod n = (m'^d)^e \mod n = m' \mod n = \mathrm{H}(m)$$

qui se vérifie en utilisant les mêmes arguments que pour le schéma de chiffrement.

1.4.3 Authentification et preuves de connaissance

L'authentification est une procédure permettant à un utilisateur de convaincre son interlocuteur de son identité. De ce fait, les protocoles d'authentification sont parfois appelés protocoles d'*identification*.

Les preuves de connaissance constituent une généralisation des schémas d'authentification. Elles permettent à un utilisateur de prouver sa connaissance d'un secret qui peut être son identité ou toute autre valeur dont lui seul doit avoir connaissance.

Une preuve de connaissance un protocole interactif, tel que nous l'avons décrit dans la définition 33, dans lequel le prouveur $\mathcal{P}$ cherche à prouver au vérificateur $\mathcal{V}$ sa connaissance d'un secret. En fin de procédure, le vérificateur $\mathcal{V}$ doit avoir l'assurance que $\mathcal{P}$ connaît bien le secret qu'il prétend posséder, sans toutefois dévoiler ce dernier. Cette notion a été introduite en 1985 par Goldwasser, Micali et Rackoff [GMR85], [GMR89] qui ont prouvé qu'il était possible de montrer l'appartenance d'un mot à un langage, sans que le mot ne soit révélé. En 1988, Feige, Fiat et Shamir ont introduit les *preuves de connaissance à divulgation nulle de connaissance* [FFS88] qui permettent de prouver la connaissance d'un secret sans rien révéler d'autre que la réponse à la question « $\mathcal{P}$ connaît-il réellement le secret ou non ? ».

1.4.3.1 Définitions

Plus formellement, on définit les preuves d'appartenance et de connaissance de la manière suivante. On note x l'entrée commune à $\mathcal{P}$ et $\mathcal{V}$ et la sortie du protocole prenant en entrée x est notée $\langle \mathcal{P}; \mathcal{V} \rangle(x)$.

Définition 43 (Preuve d'appartenance à un langage)**.** *Soit* L *un langage. Un protocole interactif* $(\mathcal{P}, \mathcal{V})$ *constitue une* preuve d'appartenance à un langage *pour* L *si les conditions suivantes sont satisfaites :*

consistance *: un prouveur honnête est accepté avec une probabilité proche de 1, c'est-à-dire qu'il existe une fonction négligeable* ν *telle que*

$$\forall x \in \mathrm{L}, \quad \Pr[\langle \mathcal{P}; \mathcal{V} \rangle(x) = 1] = 1 - \nu(|x|).$$

significativité *: un prouveur malhonnête est accepté avec une probabilité négligeable, c'est-à-dire qu'il existe une fonction négligeable* ν *telle que*

$$\forall \tilde{\mathcal{P}}, \forall x \notin \mathrm{L}, \quad \Pr\left[\langle \tilde{\mathcal{P}}; \mathcal{V} \rangle\right] < \nu(|x|).$$

On dit alors que le langage L *admet un système de preuve interactive pour l'appartenance.*

Si T est un prédicat à deux variables, on dit que ω est un *témoin* pour x si $\mathrm{T}(\omega, x) = 1$. On définit maintenant une preuve de connaissance, c'est-à-dire le protocole permettant à $\mathcal{P}$ de convaincre $\mathcal{V}$ de sa connaissance d'un secret ω. On note alors $\mathcal{P}(\omega)$ un tel prouveur.

Définition 44 (Preuve de connaissance)**.** *Un protocole interactif* $(\mathcal{P}, \mathcal{V})$ *constitue une* preuve de connaissance *pour un témoin associé à* x *si les conditions suivantes sont satisfaites :*

consistance *: un prouveur honnête est accepté avec une probabilité proche de 1, c'est-à-dire qu'il existe une fonction négligeable* ν *telle que*

$$\Pr[\langle \mathcal{P}(\omega); \mathcal{V} \rangle(x) = 1] = 1 - \nu(|x|).$$

significativité *: un prouveur malhonnête* $\tilde{\mathrm{P}}$ *est accepté avec une probabilité négligeable, c'est-à-dire qu'il existe une fonction négligeable* ν *telle que*

$$\Pr\left[\langle \tilde{\mathcal{P}}(\star); \mathcal{V} \rangle\right] < \nu(|x|).$$

On dit alors que le prédicat T *admet un système de preuve interactive de connaissance.*

Afin de prouver la propriété de significativité d'une preuve de connaissance, la plupart des preuves montrent l'existance d'un *extracteur*, c'est-à-dire une machine de Turing qui, en interagissant avec le prouveur malhonnête, en tant qu'oracle est capable de calculer un témoin.

Nous nous intéressons plus particulièrement, dans ce mémoire, aux preuves de connaissance à divulgation nulle de connaissance (appelée aussi preuves de connaissance *zero-knowledge*). Intuitivement, cette notion signifie qu'aucune information concernant la clé secrète n'est divulguée durant les échanges. Pour construire un tel protocole, on utilise un *simulateur*, c'est-à-dire un algorithme qui permet de simuler, en temps polynomial, des communications entre $\mathcal{P}$ et $\mathcal{V}$ sans connaître le secret. Un tel simulateur a été décrit par Goldwasser *et al.* dans [GMR85]. Afin de qualifier la simulation, nous rappelons ici quelques définitions d'indistinguabilité de variables aléatoires.

Définition 45 (Indistinguabilité de variables aléatoires)**.** *Soient* $\mathcal{U}$ *et* $\mathcal{V}$ *deux variables aléatoires définies sur* X *et à valeurs dans* Y*.*

$\mathcal{U}$ *et* $\mathcal{V}$ *sont* parfaitement indistinguables *si pour tout* y *dans* Y

$$\Pr_{x \in \mathrm{X}}(\mathcal{U}(x) = y) = \Pr_{x \in \mathrm{X}}(\mathcal{V}(x) = y).$$

$\mathcal{U}$ *et* $\mathcal{V}$ *sont* statistiquement indistinguables *si il existe* $k \in \mathbb{N}$ *tel que*

$$\sum_{y \in \mathrm{Y}} |\Pr_{x \in \mathrm{X}}(\mathcal{U}(x) = y) - \Pr_{x \in \mathrm{X}}(\mathcal{V}(x) = y)| < \frac{1}{|x|^k}.$$

$\mathcal{U}$ *et* $\mathcal{V}$ *sont* polynomialement(calculatoirement) indistinguables *si pour tout algorithme* D *à valeur dans* $\{0,1\}$*, pour tout entier* $k \in \mathbb{N}$ *et pour tout* $x \in \mathrm{X}$ *suffisamment grand*

$$|\Pr(\mathrm{D}(y) = 1 | y = \mathcal{U}(x)) - \Pr(\mathrm{D}(y) = 1 | y = \mathcal{V}(x))| < \frac{1}{|x|^k}.$$

Par la suite, nous notons $\mathcal{V}iew_{\mathcal{V}}\langle \mathcal{P}(\omega) \rangle(x)$ la variable aléatoire relative aux communications réelles entre $\mathcal{P}$ et $\mathcal{V}$ lorsque le prouveur opère avec le secret ω associé à x. La variable aléatoire associée aux communications simulées est notée $\mathcal{M}_{\mathcal{V}}(x)$.

Définition 46 (Divulgation nulle de connaissance)**.** *Un protocole interactif $\langle \mathcal{P}; \tilde{\mathcal{V}} \rangle$ est à divulgation nulle de connaissance si, pour tout vérificateur (non nécessairement honnête) $\tilde{\mathcal{V}}$, il existe un simulateur polynomial $\mathcal{M}_{\tilde{\mathcal{V}}}$ tel que*

$$\forall \omega, \forall x \in \{0,1\}^*, \quad \mathcal{V}iew_{\mathcal{V}}\langle \mathcal{P}(\omega) \rangle(x) \sim \mathcal{M}_{\tilde{\mathcal{V}}}(x)$$

où $\sim$ signifie parfaitement, statistiquement ou calculatoirement indistinguable.

On dit alors que le protocole est respectivement parfaitement, statistiquement ou calculatoirement à divulgation nulle de connaissance.

Contrairement au chiffrement ou à la signature, il n'existe pas d'algorithme pour décrire le fonctionnement des preuves de connaissance. En effet, celles ci peuvent se dérouler de diverses manières. Nous n'utiliserons, pour notre part, que des preuves de connaissance s'exécutant en trois passes qui peuvent être décrites de la manière suivante. Celles ci ne peuvent être à divulgation nulle de connaissance que vis à vis de vérificateurs honnêtes, mais cela suffira à nos constructions.

Définition 47 (Preuve de connaissance en trois passes)**.** *Une* preuve de connaissance en trois passes *est définie par les algorithmes suivants :*

- Setup*, algorithme probabiliste de génération de paramètres du système qui prend en entrée un entier k et qui renvoie les paramètres publics params ainsi qu'une paire de clés privée/publique (sk, pk) pour le prouveur.*
  $(sk, pk) \leftarrow \mathsf{Setup}(1^k)$
- Interactions*, protocole interactif entre le prouveur $\mathcal{P}$ qui prend en entrée la clé secrète sk, et le vérificateur $\mathcal{V}$ qui prend en entrée la clé publique pk. Le protocole se déroule alors en trois étapes :*
  - `Commitment`*, le prouveur envoie un engagement,*
  - `Challenge`*, en réponse, le vérificateur envoie au prouveur un* challenge *ou* défi*,*
  - `Response`*, le prouveur envoie une* réponse *qui permet au vérificateur de vérifier qu'il connaît bien le secret.*

En fin de procédure, $\mathcal{V}$ renvoie 1 s'il est convaincu que la réponse est correcte, 0 sinon.

1.4.3.2 Exemples

Nous donnons ici quelques exemples classiques de protocole d'authentification et de preuve de connaissance. Nous donnerons au chapitre 3 des constructions plus complexes.

Protocole d'authentification de Schnorr. Le protocole d'authentification le plus répandu est très certainement celui dû à Schnorr et proposé en 1989 dans [Sch89].

En voici la description (figure 1.3).

Setup

Les paramètres du protocole sont :

- k et k' deux paramètres de sécurité,
- p et q, deux entiers premiers tels que $q|p-1$, $p > 2^{k'}$ et $q > 2^k$,
- g un élément de $\mathbb{Z}_p^*$ d'ordre q.

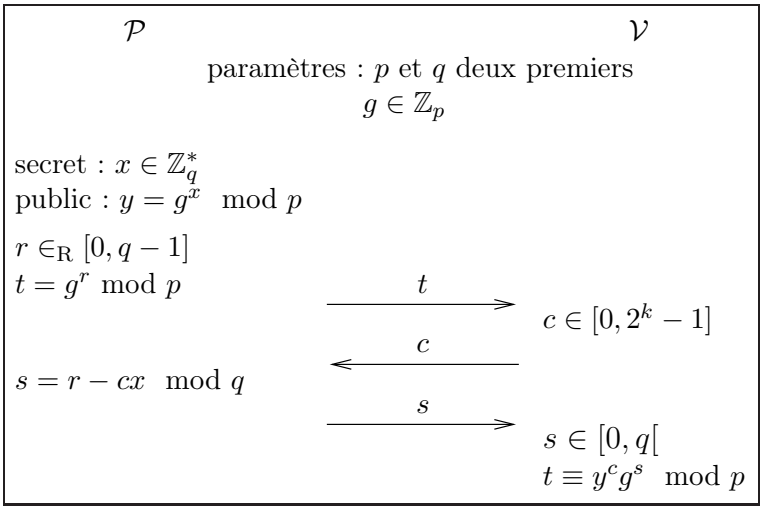

FIG. 1.3 – Protocole d'authentification de Schnorr.

Le prouveur possède une clé secrète $x \in \mathbb{Z}_q$ et sa clé publique est donnée par $y = g^x \mod p$.

Interactions

Afin de prouver son identité, le prouveur $\mathcal{P}$ certifie le vérificateur $\mathcal{V}$ qu'il connaît la clé secrète associée à y, c'est-à-dire le logarithme discret en base g de y. L'authentification se fait alors en trois passes :

- `Commitment` : $\mathcal{P}$ choisit une valeur aléatoire $r \in [0, q-1]$, calcule $t = g^r \mod p$ et l'envoie à $\mathcal{V}$,
- `Challenge` : $\mathcal{V}$ choisit une valeur aléatoire $c \in [0, 2^k - 1]$ et l'envoie à $\mathcal{P}$,
- `Response` : $\mathcal{P}$ calcule sa réponse $s := r - cx \mod q$ et l'envoie à $\mathcal{V}$.

Ce dernier vérifie alors si $t \equiv y^c g^s \mod p$.

Si k est suffisamment grand, alors on peut montrer que le protocole d'authentification de Schnorr est une preuve de connaissance parfaitement à divulgation nulle de connaissance face à un vérificateur honnête.

Par extension, ce protocole permet de prouver la connaissance d'un logarithme discret dans n'importe quel groupe d'ordre connu q.

Authentification dans un groupe d'ordre inconnu. On cherche maintenant à prouver la connaissance d'un logarithme discret dans un groupe d'ordre inconnu (typiquement, un groupe d'ordre n où $n = pq$ est le produit de deux nombres premiers et la décomposition de n n'est pas connue). Il n'est alors plus possible de calculer la réduction modulaire dans la dernière passe du protocole de Schnorr.

On peut dans ce cas utiliser le protocole de Girault [Gir90], dont Poupard et Stern ont montré qu'il s'agissait d'une preuve de connaissance statistiquement à divulgation nulle de connaissance [PS98]. Ce protocole est souvent évoqué sous l'abréviation GPS [GPS06b].

Nous donnons ici une description du protocole rappelé à la figure 1.4.

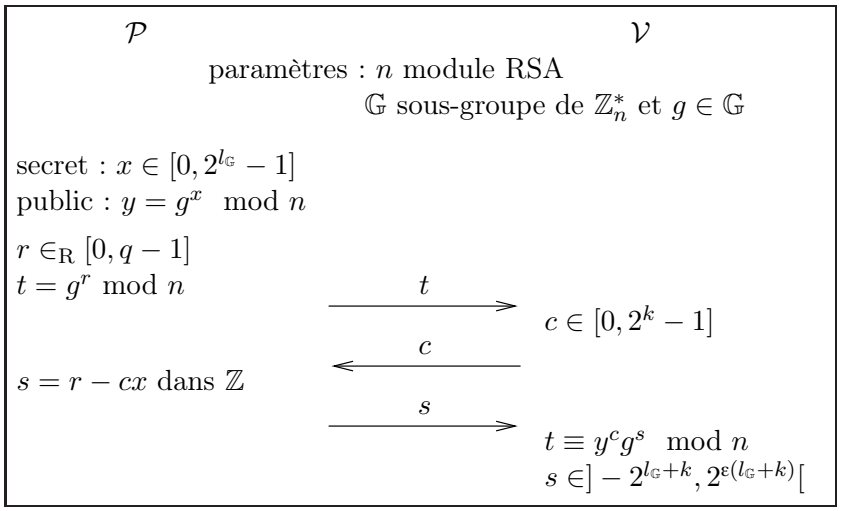

FIG. 1.4 – Protocole d'authentification GPS.

Setup

Les paramètres du protocole sont donnés par les éléments suivants :

- $\varepsilon > 1$ et k, deux paramètres de sécurité,
- n un module RSA et $\mathbb{G}$ un sous-groupe cyclique de $\mathbb{Z}_n^*$ d'ordre inconnu $\text{ord}(\mathbb{G})$,
- g un élément de $\mathbb{G}$,
- $l_\mathbb{G} = \lceil \log_2(\text{ord}(\mathbb{G})) \rceil$ est connu.

Le prouveur possède une clé secrète $x \in [0, 2^{l_\mathbb{G}} - 1]$ et sa clé publique est donnée par $y = g^x \mod n$.

Interactions

Afin de prouver son identité, le prouveur $\mathcal{P}$ certifie le vérificateur $\mathcal{V}$ qu'il connaît la clé secrète associée à y, c'est-à-dire le logarithme discret en base g de y. L'authentification se fait alors en trois passes :

- `Commitment` : $\mathcal{P}$ choisit une valeur aléatoire $r \in [0, q-1]$, calcule $t = g^r \mod n$ et l'envoie à $\mathcal{V}$,
- `Challenge` : $\mathcal{V}$ choisit une valeur aléatoire $c \in [0, 2^k]$ et l'envoie à $\mathcal{P}$,
- `Response` : $\mathcal{P}$ calcule sa réponse $s := r - cx$ dans $\mathbb{Z}$ et l'envoie à $\mathcal{V}$.

Ce dernier vérifie alors si $t \equiv y^c g^s \mod n$ et si $s \in]-2^{l_\mathbb{G}+k}, 2^{\varepsilon(l_\mathbb{G}+k)}[$.

Preuve de connaissance d'une représentation. Tout comme la connaissance d'un logarithme discret, il est possible de prouver la connaissance d'une représentation en généralisant le protocole de Schnorr au cas où plusieurs générateurs du groupe sont connus.

Le prouveur possède un élément $y \in \mathbb{G}$. Il veut alors prouver à $\mathcal{V}$ qu'il connaît une représentation de y dans la « base » $(g_1, \ldots, g_m)$ où $(g_1, \ldots, g_m)$ est une famille de

générateurs de $\mathbb{G}$. La preuve se déroule comme détaillé dans la figure 1.5 et consiste en une composition de preuves de connaissance de logarithme discret de Schnorr.

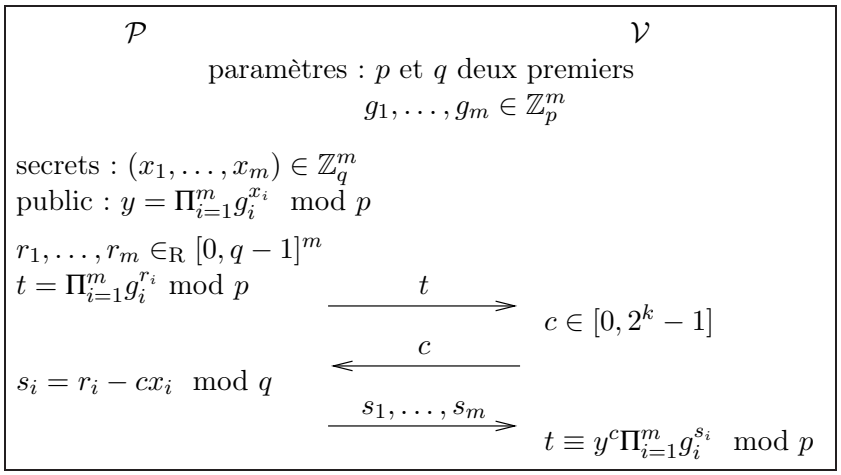

FIG. 1.5 – Preuve de connaissance d'une représentation.

Preuve d'égalité de logarithmes discrets. Le prouveur peut aussi être amené, dans certaines applications, à prouver que deux logarithmes sont égaux. Cette preuve est elle aussi inspirée directement du protocole d'authentification de Schnorr et est détaillée à la figure 1.6. Le prouveur exécute simultanément deux preuves de connaissance de logarithme discret en utilisant le même aléa. Cette preuve et la preuve de connaissance

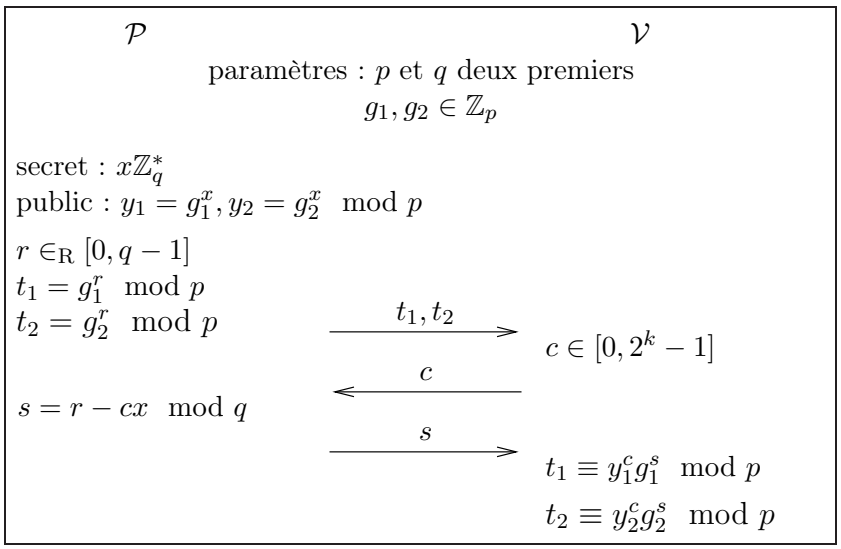

FIG. 1.6 – Preuve d'égalité de logarithmes discrets.

d'une représentation peuvent être combinées et donner une preuve d'égalité de deux

représentations.

1.4.3.3 Preuves et signatures de connaissance

Les protocoles que nous avons décrits ci-dessus sont tous des protocoles à trois passes et nécessitent donc une interaction entre le prouveur et le vérificateur. Cependant, pour certaines applications, il peut être important d'éviter au maximum les échanges entre prouveur et vérificateur. La connaissance d'un secret va alors être prouvée à l'aide d'une *preuve non interactive*. En 1986, Fiat et Shamir ont proposé un procédé heuristique permettant de transformer un protocole en trois passes en une *signature de connaissance* non interactive [FS86]. Pointcheval et Stern ont montré formellement dans [PS00] comment transformer un protocole interactif en trois passes en une signature de connaissance non interactive, dans laquelle le challenge envoyé par le vérificateur dans la deuxième phase est un haché des paramètres publics et de l'engagement.

Pour illustrer cette technique, nous décrivons ici la signature de connaissance obtenue à partir du protocole d'authentification de Schnorr.

Setup

Les paramètres sont définis comme suit :

- k, un paramètre de sécurité,
- p et q, deux entiers premiers tels que $q|p-1$ et $q > 2^k$,
- g un élément de $\mathbb{Z}_p^*$ d'ordre q,
- $\mathrm{H} = \{0,1\}^* \rightarrow [0, 2^k - 1]$ est une fonction de hachage.

La clé secrète de signature sk est donnée par $x \in \mathbb{Z}_q$ et la clé publique pk par $y = g^x \mod p$.

Sign

On retrouve dans la phase de signature les trois passes du protocole d'authentification. Afin de prouver la connaissance de la valeur x, A effectue les opérations suivantes :

- « `Commitment` » : $\mathcal{A}$ choisit une valeur aléatoire $r \in \mathbb{Z}/q\mathbb{Z}$ et calcule $t = g^r \mod p$,
- « `Challenge` » : A calcule $c = \mathrm{H}(y||p||q||t)$ et $s = r - cx \mod q$,
- « `Response` » : la signature est donnée par $\sigma = (t, s)$.

Verify

La vérification se fait en trois étapes :

- B calcule $c = \mathrm{H}(y||p||q||t)$,
- il vérifie que $t = y^c g^s \mod p$.

1.4.3.4 Notation

Tout au long de ce mémoire, nous utiliserons la notation suivante pour décrire la preuve de connaissance d'une valeur α qui vérifie le prédicat T :

$$pok(\alpha : \mathrm{T}(\alpha, \ldots)).$$

Les preuves que nous avons introduites ci-dessus seront donc notées :

- preuve de connaissance d'un logarithme discret : $pok(\alpha : y = g^{\alpha})$,
- preuve de connaissance d'une représentation : $pok(\alpha_1, \alpha_m : y = \Pi_{i=1}^{m} g_i^{\alpha_i})$,
- preuve d'égalité de logarithmes : $pok(\alpha : y_1 = g_1^{\alpha} \wedge y_2 = g_2^{\alpha})$.

Chapitre 2

Sécurité prouvée

Si décrire un nouveau schéma efficace est une tâche importante, s'assurer de sa sécurité l'est encore plus. La sécurité d'un schéma cryptographique à clé publique repose sur trois points majeurs :

- préciser les notions de sécurité à garantir,
- préciser les hypothèses calculatoires (typiquement : tel problème est difficile),
- présenter une réduction (de la sécurité du schéma à la difficulté du problème).

Nous avons décrit, au chapitre précédent, les notions de sécurité des principaux cryptosystèmes. Ce chapitre s'intéresse aux deux points restants (hypothèses calculatoires et réductions) et présente les différents modèles dans lesquels un adversaire peut se placer pour mener ses attaques.

Sommaire

2.1 Problèmes difficiles

La sécurité de nombreux protocoles à clé publique repose sur des problèmes dits "difficiles". Il s'agit en fait de problèmes pour lesquels il est facile de créer des instances dont une solution est connue, mais dont la résolution générale est, d'un point de vue algorithmique, difficile.

Informellement, on définit un *problème (calculatoire) difficile* $\mathcal{P}$, dont les instances sont dimensionnées par un paramètre de sécurité k, comme étant un problème pour lequel tout algorithme efficace s'exécutant en temps polynomial t retourne une solution

avec probabilité ε négligeable, (t et ε étant des fonctions de k). Nous donnerons une définition plus formelle dans le cadre du problème RSA, à la définition 51.

La cryptographie à clé publique repose sur deux grandes familles de problèmes difficiles. Nous définissons ici les problèmes et hypothèses les plus répandus ainsi que ceux qui nous serons nécessaires pour prouver la sécurité de nos schémas.

L'avantage d'un adversaire contre un problème supposé difficile sera noté $\mathsf{Adv}^{\mathbb{P}}_{\mathsf{G},\mathcal{A}}(k)$ où $\mathbb{P}$ représente le problème difficile, G le générateur pris en entrée, $\mathcal{A}$ l'adversaire et k le paramètre de sécurité.

De même, l'avantage d'un adversaire $\mathcal{A}$ contre la propriété P d'un schéma S sera noté $\mathsf{Adv}^{\mathrm{P}}_{\mathrm{S},\mathcal{A}}(k)$.

2.1.1 Problèmes liés à la factorisation

La première grande famille de problèmes difficiles repose sur le problème de la factorisation, c'est-à-dire connaissant un entier, trouver sa décomposition en nombres premiers.

2.1.1.1 Problème RSA

Le problème RSA a été introduit en même temps que le système à clé publique RSA (*cf.* section 1.4.1) ([RSA78]).

Il faut au préalable définir un *générateur de clés RSA*, c'est-à-dire un algorithme capable de produire les clés nécessaires au chiffrement RSA.

Définition 48 (Générateur de clés RSA)**.** *Un* générateur de clés RSA $\mathsf{G}_{\mathrm{RSA}}$ *est un algorithme prenant en entrée un paramètre de sécurité k et renvoyant un triplet $(n, e, d) \in \mathbb{N} \times (\mathbb{Z}^*_{\phi(n)})^2$ tel que*

- *n est un module RSA. De plus, en notant $n = pq$, p et q premiers , on a $2^{(k-1)/2} < p, q < 2^{k/2}$,*
- *$ed \equiv 1 \mod \phi(n)$.*

*On dit que e est l'*exposant public *et d, l'*exposant secret*.*

On définit alors intuitivement le problème RSA de la manière suivante :

Définition 49 (Problème RSA)**.** *Soient $n = pq$ (avec p et q deux entiers premiers) un entier, e un nombre entier premier avec $\varphi(n)$ et y appartenant à $[0, n-1]$. Le* problème RSA *consiste à trouver l'unique entier x inférieur à n tel que $y \equiv x^e \mod n$.*

Autrement dit, étant donné un module RSA n suffisamment grand, il est difficile de trouver la racine e-ème d'un élément de $\mathbb{Z}_n$, tout en sachant que cette racine existe et est unique.

On considère généralement que le problème RSA est aussi dur que le problème de la factorisation. Il n'existe aucune preuve d'une telle assertion, mais à ce jour, aucune méthode efficace n'a été proposée permettant de résoudre le problème RSA autrement qu'en factorisant le module. De ce fait, il suffit à garantir la sécurité.

Afin de quantifier l'avantage d'un adversaire contre le problème RSA, on définit sa probabilité de succès dans une expérience visant à lui trouver une solution. Ceci permet ensuite de définir précisément l'hypothèse RSA.

Définition 50 (Avantage RSA)**.** *Soient k un entier et $\mathsf{G}_{\mathrm{RSA}}$ un générateur de clés RSA. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un couple (n, e) issu du générateur $\mathsf{G}_{\mathrm{RSA}}$ et un élément $y \in \mathbb{Z}_n$ retournant un élément x. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{RSA}}_{\mathsf{G}_{\mathrm{RSA}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{RSA}}_{\mathsf{G}_{\mathrm{RSA}},\mathcal{A}}(k)$
$(n, e, d) \leftarrow \mathsf{G}_{\mathrm{RSA}}(k)$
$params \leftarrow (n, e)$
$y \xleftarrow{\mathrm{R}} (\mathbb{Z}/n\mathbb{Z})^*$
$x \leftarrow \mathcal{A}(k, params, y)$
`Si` $x^e \equiv y \mod n$ *renvoie 1*
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{RSA}}_{\mathsf{G}_{\mathrm{RSA}},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{RSA}}_{\mathsf{G}_{\mathrm{RSA}},\mathcal{A}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0, 1]$, le problème RSA est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage $\mathbf{Adv}^{\mathrm{RSA}}_{\mathsf{G}_{\mathrm{RSA}},\mathcal{A}}(k) \geq \varepsilon$.

Définition 51 (Hypothèse RSA)**.** *Pour tous entiers positifs c et c', et pour k suffisamment grand, le problème RSA est $(k, k^c, k^{-c'})$-difficile.*

Note : Plus généralement, quand on considérera un problème $\mathcal{P}$ et que l'on aura défini et quantifié l'avantage d'un attaquant contre celui-ci de façon analogue à la définition 50, on fera l'hypothèse que, pour tous entiers positifs c et c', et pour tout k suffisamment grand, il est $(k, k^c, k^{-c'})$-difficile.

2.1.1.2 Problème RSA flexible

Le problème RSA flexible a été introduit simultanément par Barić et Pfitzmann [BP97] et Fujisaki et Okamoto [FO97]. Il s'agit d'une version affaiblie du problème RSA (ou, au choix, d'une version renforcée de l'hypothèse RSA). Il se définit de la manière suivante :

Définition 52 (Problème RSA flexible)**.** *Soient $n = pq$ (avec p et q deux entiers premiers) un entier et y un élément de $\mathbb{Z}_n$. Le* problème RSA fort *ou* flexible *consiste à trouver un couple (x, e) tel que $x \in \mathbb{Z}_n$, $e \geq 2$ et $y = x^e \mod n$.*

Le problème du RSA flexible se réduit au problème RSA, ce qui signifie que connaissant une solution pour une instance d'un problème RSA on en déduit une solution pour le problème du RSA flexible (à condition que le n soit le même). On peut aussi dire que l'hypothèse RSA flexible est plus forte que l'hypothèse RSA, c'est pourquoi cette hypothèse est souvent appelée *hypothèse RSA fort* (*Strong RSA assumption).*

2.1.2 Problèmes liés au logarithme discret

La famille des problèmes basés sur le logarithme discret représente une autre part importante des problèmes difficiles utilisés en cryptographie. La complexité de ces problèmes repose sur la difficulté d'extraire un logarithme discret et contrairement au problème RSA, ce problème se pose même lorsqu'on connaît l'ordre du groupe.

Le logarithme d'un élément est défini de la manière suivante :

Définition 53 (Logarithme discret)**.** *Soient $\mathbb{G}$ un groupe cyclique fini, g un générateur de ce groupe et y un élément de $\mathbb{G}$. Le logarithme discret de y en base g, noté $\log_g y$, est l'entier $x \in \mathbb{Z}$ inférieur à $\mathrm{ord}(g)$ vérifiant $y = g^x$.*

Nous ne considérons dans cette section que des groupes d'ordre premier. En effet, il est possible, lorsqu'on travaille dans un groupe d'ordre n quelconque de se ramener à des problèmes basés sur le logarithme discret dans les sous-groupes d'ordre premier divisant n, lorsqu'on connaît les facteurs de n. Un générateur de groupe d'ordre premier est alors donné par la définition suivante.

Définition 54 (Générateur de groupe premier)**.** *Un* générateur de groupe premier G_{DL} *est un algorithme efficace prenant en entrée un paramètre de sécurité k et retournant un triplet $(q, g, \mathbb{G})$ tel que g est un générateur du groupe $\mathbb{G}$ d'ordre premier q et vérifiant $2^{k-1} < q < 2^k$.*

2.1.2.1 Problème du logarithme discret

Le problème du logarithme discret est défini de la manière suivante.

Définition 55 (Problème du logarithme discret)**.** *Soient $\mathbb{G}$ un groupe cyclique fini, g un générateur de ce groupe et y un élément de $\mathbb{G}$. Le problème du logarithme discret général, noté* DL*, consiste à calculer $\log_g y$, c'est-à-dire , trouver un entier $x \in \mathbb{Z}$ inférieur à $\mathrm{ord}(g)$ vérifiant $y = g^x$.*

Comme précédemment, nous évaluons maintenant l'avantage d'un adversaire pour résoudre le problème du logarithme discret.

Définition 56 (Expérience du logarithme discret)**.** *Soient k un entier et G_{DL} un générateur de groupe premier. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un triplet $(q, g, \mathbb{G})$ issu du générateur G_{DL} et un élément h de $\mathbb{G}$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{DL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{DL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$

$(q, g, \mathbb{G}) \leftarrow \mathsf{G}_{\mathrm{DL}}(k)$
$params \leftarrow (q, g, \mathbb{G})$
$x \xleftarrow{\mathrm{R}} [1, q-1]$
$h = g^x$
$x' \leftarrow \mathcal{A}(k, params, h)$
`Si` $x' = x$ renvoie 1
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{DL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{DL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0,1]$, le problème DL est dit (k,τ,ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}^{\mathrm{DL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) \geq \varepsilon.$$

L'algorithme le plus efficace pour résoudre le problème du logarithme discret dépend essentiellement du groupe dans lequel on travaille. Par exemple, pour les sous-groupes de $\mathbb{Z}_q^*$, il existe des algorithmes sous-exponentiels efficaces pour le résoudre. En revanche, sur les courbes elliptiques, seuls des algorithmes classiques, de complexité $\sqrt{q}$ peuvent être utilisés.

2.1.2.2 Problème du logarithme discret de plus

Le problème du *logarithme discret de plus* (*one-more discrete logarithm*) a été introduit par Bellare, Namprempre, Pointcheval et Semanko dans [BNPS03] afin de prouver la sécurité du schéma de signature aveugle de Chaum [Cha83]. Pour mener à bien son attaque, l'adversaire a accès à un oracle $\mathsf{Dlog}(.)$ capable de résoudre le problème du logarithme discret défini ci-dessus. L'oracle reçoit en entrée une valeur $h \in \mathbb{G}$ et renvoie une valeur x, satisfaisant $h = g^x$.

Définition 57 (Problème du logarithme discret de plus)**.** *Étant donnés un groupe $\mathbb{G}$ d'ordre p, un générateur g et un ensemble de $\ell+1$ valeurs $(z_1,\ldots,z_\ell,z_{\ell+1}) \in \mathbb{G}^\ell$, le* problème du logarithme discret de plus*, noté OMDL, consiste à renvoyer $\ell+1$ couples $((x_1,z_1),\ldots,(x_{\ell+1},z_{\ell+1}))$ tels que $\log_g z_i = x_i$ pour $i \in [1,\ell+1]$ en ayant accédé au plus ℓ fois à l'oracle $\mathsf{Dlog}(.)$.*

Définition 58 (Expérience du logarithme discret de plus)**.** *Soient k un entier, $m : \mathbb{N} \to \mathbb{N}$ une fonction de k, et G_{DL} un générateur de groupe premier. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un triplet $(q,g,\mathbb{G})$ issu du générateur G_{DL} et ayant accès à l'oracle $\mathsf{Dlog}(.)$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{OMDL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{OMDL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$

$(q,g,\mathbb{G}) \leftarrow \mathsf{G}_{\mathrm{DL}}(k)$
$params \leftarrow (q,g,\mathbb{G})$
`Pour` $i = 1$ *à* $m(k)+1$ `faire` $x_i \leftarrow \mathbb{Z}_q; y_i \leftarrow g^{x_i}$
$(z_1,\ldots,z_{m(k)+1}) \leftarrow \mathcal{A}(k, params, y_1,\ldots,y_{m(k)+1} : \mathsf{Dlog})$
`Si` les conditions suivantes sont vraies :
- $\forall i \in \{1,\ldots,m(k)+1\}; y_i = g^{z_i}$
- $\mathcal{A}$ a fait au plus $m(k)$ requêtes à l'oracle Dlog

renvoie 1
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{OMDL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{OMDL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ *et* $\varepsilon \in [0,1]$*, le problème OMDL est dit* (k,τ,ε)*-difficile, s'il n'existe pas d'adversaire* $\mathcal{A}$ *fonctionnant en un temps* τ *et ayant un avantage*

$$\mathbf{Adv}^{\mathrm{OMDL}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) \geq \varepsilon.$$

La famille des problèmes basés sur le logarithme discret contient une très large sous-famille, celle des problèmes Diffie-Hellman, liés au protocole d'échange de clés introduit par Diffie et Hellman en 1976 [DH76].

Le problème le plus fort est le problème Diffie-Hellman calculatoire. Il existe ensuite de très nombreuses variantes et nous ne détaillons ici que celles nécessaires à la compréhension de notre mémoire.

2.1.2.3 Problème Diffie-Hellman calculatoire

Définition 59 (Problème CDH)**.** *Étant donnés un groupe* $\mathbb{G}$*, un générateur* g *de ce groupe et* $\mathrm{A} = g^a$ *et* $\mathrm{B} = g^b$ *deux éléments de* $\mathbb{G}$*, le* problème calculatoire Diffie-Hellman *consiste à calculer* g^{ab}*.*

Définition 60 (Expérience CDH)**.** *Soient* k *un entier et* G_{DL} *un générateur de groupe premier. Soit* $\mathcal{A}$ *un attaquant prenant en entrée un paramètre de sécurité* k*, un triplet* $(q,g,\mathbb{G})$ *issu du générateur* G_{DL} *et un couple* $(\mathrm{A},\mathrm{B}) \in \mathbb{G}$ *et renvoyant un élément* $\mathrm{C} \in \mathbb{G}$*. On associe à cet adversaire l'expérience* $\mathbf{Exp}^{\mathrm{CDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$ *décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{CDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$

$(q,g,\mathbb{G}) \leftarrow \mathsf{G}_{\mathrm{DL}}(k)$
$params \leftarrow (q,g,\mathbb{G})$
$a \xleftarrow{\mathrm{R}} [1,q-1], \mathrm{A} = g^a$
$b \xleftarrow{\mathrm{R}} [1,q-1], \mathrm{B} = g^b$
$\mathrm{C} \leftarrow \mathcal{A}(k, params, \mathrm{A}, \mathrm{B})$
`Si` $\mathrm{C} = g^{ab}$ renvoie 1
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{CDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{CDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ *et* $\varepsilon \in [0,1]$*, le problème CDH est dit* (k,τ,ε)*-difficile, s'il n'existe pas d'adversaire* $\mathcal{A}$ *fonctionnant en un temps* τ *et ayant un avantage*

$$\mathbf{Adv}^{\mathrm{CDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) \geq \varepsilon.$$

On dit alors que le triplet $(\mathrm{A} = g^a, \mathrm{B} = g^b, \mathrm{C} = g^c) \in \mathbb{G}$ pour $(a,b,c) \in [0,q-1]$ est un *triplet Diffie-Hellman* si $\mathrm{C} = g^{ab}$.

L'hypothèse CDH est plus forte que l'hypothèse du logarithme discret. En effet, la résolution du problème du logarithme discret, entraîne celle du problème CDH est facile.

2.1.2.4 Problème Diffie-Hellman décisionnel

Il existe aussi une version décisionnelle du problème CDH. Cette fois l'adversaire reçoit un triplet de valeurs et il doit décider s'il s'agit d'un triplet Diffie-Hellman ou non.

Définition 61 (Problème DDH)**.** *Étant donnés un groupe $\mathbb{G}$, un générateur g de ce groupe et $\mathrm{A} = g^a$, $\mathrm{B} = g^b$ et $\mathrm{C} = g^c$ trois éléments de $\mathbb{G}$, le* problème décisionnel Diffie-Hellman *consiste à décider si $c = ab$ est vrai ou non.*

Définition 62 (Expérience DDH)**.** *Soient k un entier et G_{DL} un générateur de groupe premier. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un triplet $(q, g, \mathbb{G})$ issu du générateur G_{DL}, un triplet $(\mathrm{A}, \mathrm{B}, \mathrm{C}) \in \mathbb{G}$ et renvoyant un bit $b \in \{0, 1\}$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{DDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{DDH}-r}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$

$(q, g, \mathbb{G}) \leftarrow \mathsf{G}_{\mathrm{DL}}(k)$
$params \leftarrow (q, g, \mathbb{G})$
$a \xleftarrow{\mathrm{R}} [1, q-1], \mathrm{A} = g^a$
$b \xleftarrow{\mathrm{R}} [1, q-1], \mathrm{B} = g^b$
`Si` $r = 0$ `faire` $\mathrm{C} = g^{ab}$
`Si` $r = 1$ `faire` $\mathrm{C} \xleftarrow{\mathrm{R}} \mathbb{G}$
$b \leftarrow \mathcal{A}(k, params, \mathrm{A}, \mathrm{B}, \mathrm{C})$
Renvoie b.

L'avantage de l'adversaire est donné par

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

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0, 1]$, le problème DDH est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}^{\mathrm{DDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) \geq \varepsilon$$

.

Ce problème est plus facile que le problème CDH. En effet, s'il existe un algorithme polynomial capable de résoudre le problème CDH à partir des valeurs a et b, alors il est facile de calculer $\mathrm{C}' = g^{ab}$ et de tester $\mathrm{C} = \mathrm{C}'$.

Les deux hypothèses que nous venons de présenter sont considérées comme des hypothèses relativement faibles et permettent généralement d'assurer des conditions de sécurité efficaces pour les protocoles les utilisant. Les problèmes que nous définissons dans la suite de cette section sont des problèmes se ramenant soit au problème CDH soit au problème DDH. Par conséquent, ils définissent des hypothèses plus fortes et ne permettent pas de garantir le même niveau de sécurité. Ils restent cependant tout à fait satisfaisants pour nos besoins.

2.1.2.5 Problèmes gap

Les problèmes *gap* ont été introduits par Okamoto et Pointcheval en 2001 [OP01]. L'idée est de résoudre un problème calculatoire en autorisant l'accès à un oracle pour un problème décisionnel associé.

On définit ainsi le problème Gap Diffie-Hellman. L'attaquant a alors accès à un oracle DDH qui prend en entrée une instance donnée par l'attaquant et renvoie 1 s'il s'agit d'une instance DDH, 0 sinon.

Définition 63 (Problème GDH)**.** *Soient $\mathbb{G}$ un groupe d'ordre premier q et g un générateur de $\mathbb{G}$. Étant donné $(\mathrm{A} = g^a, \mathrm{B} = g^b) \in \mathbb{G}^2$ le problème* gap Diffie-Hellman*(GDH) consiste à calculer l'élément $\mathrm{Z} = g^{ab}$ avec l'aide d'un oracle pour le problème DDH, noté $\mathsf{O}_{\mathrm{DDH}}$.*

Définition 64 (Expérience GDH)**.** *Soient k un entier et G_{DL} un générateur de groupe premier. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un triplet $(q, g, \mathbb{G})$ issu du générateur G_{DL}, un couple $(\mathrm{A}, \mathrm{B}) \in \mathbb{G}$ et ayant accès à un oracle DDH $\mathsf{O}_{\mathrm{DDH}}$et renvoyant un élément $\mathrm{C} \in \mathbb{G}$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{GDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{GDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$

$(q, g, \mathbb{G}) \leftarrow \mathsf{G}_{\mathrm{DL}}(k)$
$params \leftarrow (q, g, \mathbb{G})$
$a \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{A} = g^a$
$b \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{B} = g^b$
$\mathrm{C} \leftarrow \mathcal{A}(k, params, \mathrm{A}, \mathrm{B} : \mathsf{O}_{\mathrm{DDH}})$
`Si` $c = \mathrm{G}^{\mathrm{AB}}$ renvoie 1
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{GDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{GDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0, 1]$, le problème GDH est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}^{\mathrm{GDH}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) \geq \varepsilon$$

.

2.1.2.6 Problème Diffie-Hellman inversé

Définition 65 (Problème y-DDHI)**.** *Soit $\mathbb{G}$ un groupe premier q engendré par g. Étant donnés $(g, g^x, \ldots, g^{x^y})$ pour une valeur aléatoire $x \in \mathbb{Z}_q$ et une valeur $\mathrm{D} \in \mathbb{G}$, le problème d'*inversion Diffie-Hellman décisionnel *consiste à décider si $\mathrm{D} = g^{1/x}$ ou pas.*

Définition 66 (Expérience y-DDHI)**.** *Soient k un entier et G_{DL} un générateur de groupe premier. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un triplet $(q, g, \mathbb{G})$ issu du générateur G_{DL}, un ensemble $(g, g^x, \ldots, g^{x^y})$ pour une valeur aléatoire $x \in \mathbb{Z}_q$ et une valeur $\mathrm{D} \in \mathbb{G}$ et renvoyant un bit $b \in \{0, 1\}$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{y-\mathrm{DDHI}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{y-\mathrm{DDHI}-r}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k)$

$(q, g, \mathbb{G}) \leftarrow \mathsf{G}_{\mathrm{DL}}(k)$
$params \leftarrow (q, g, \mathbb{G})$
$x \overset{\mathrm{R}}{\leftarrow} [1, q-1]$
`Si` $r = 0$ `faire` $\mathrm{D} = g^{1/x}$
`Si` $r = 1$ `faire` $\mathrm{D} \overset{\mathrm{R}}{\leftarrow} \mathbb{G}$
$b \leftarrow \mathcal{A}(k, params, (g, g^x, \ldots, g^{x^y}, \mathrm{D})$
Renvoie b.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{y-\mathrm{DDHI}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = \left| \Pr[\mathbf{Exp}^{y-\mathrm{DDHI}-1}_{\mathsf{G}_{\mathrm{D}},\mathcal{A}}(k) = 1] - \Pr[\mathbf{Exp}^{y-\mathrm{DDHI}-0}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) = 1] \right|.$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0, 1]$, le problème y-DDHI est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}^{y-\mathrm{DDHI}}_{\mathsf{G}_{\mathrm{DL}},\mathcal{A}}(k) \geq \varepsilon$$

.

2.1.3 Problèmes Diffie-Hellman bilinéaires

Nous nous intéressons, dans cette section à décrire les problèmes spécifiques aux cryptosystèmes basés sur les applications bilinéaires. En effet, le développement de ces systèmes a engendré de nombreux problèmes nouveaux. Certains sont devenus des problèmes standards, d'autres, plus particuliers, sont souvent liés à un schéma. Nous décrivons ici les problèmes les plus couramment utilisés ainsi que ceux qui nous serviront à prouver la sécurité des schémas par la suite. Une liste plus exhaustive de ces problèmes est donnée dans [DBS04].

La plupart des problèmes bilinéaires sont, dans leurs conceptions, des transpositions des problèmes classiques, tout en tenant compte des contraintes induites par les applications bilinéaires.

Nous avons tout d'abord besoin de définir un générateur de paramètres bilinéaires, c'est-à-dire un générateur de groupe qui, prenant en entrée un paramètre de sécurité, renvoie les éléments nécessaires à la construction d'une application bilinéaire admissible.

Définition 67 (Générateur de groupe BDH)**.** *Un générateur de groupe BDH, noté $\mathsf{G}_{\mathrm{BDG}}$, est un algorithme prenant en entrée un paramètre de sécurité k et renvoyant un ensemble $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$ tel que q est un nombre premier tel que $2^{k-1} < q < 2^k$, g_1 (*resp. *g_2) est un générateur du groupe $\mathbb{G}_1$ (*resp. *$\mathbb{G}_2$) d'ordre q, $\mathbb{G}_{\mathrm{T}}$ est un groupe d'ordre q et $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_{\mathrm{T}}$ est une application bilinéaire admissible.*

2.1.3.1 Problème Diffie-Hellman bilinéaire calculatoire

Le problème Diffie-Hellman bilinéaire calculatoire se définit de la façon suivante :

Définition 68 (Problème CBDH)**.** *Soient $\mathbb{G}_1$ et $\mathbb{G}_2$ deux groupes d'ordre premier q, engendrés par g_1 et g_2 respectivement. Étant donnés $(\mathrm{A} = g_1^a, \mathrm{B} = g_1^b, \mathrm{C} = g_2^c) \in \mathbb{G}_1^2 \times \mathbb{G}_2$ le* problème Diffie-Hellman bilinéaire calculatoire *(CBDH) consiste à calculer $e(g_1, g_2)^{abc}$.*

L'expérience associée est donnée par la définition suivante.

Définition 69 (Expérience CBDH)**.** *Soient k un entier et $\mathsf{G}_{\mathrm{BDH}}$ un générateur de groupe BDH. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un ensemble $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e)$ issu du générateur $\mathsf{G}_{\mathrm{BDH}}$ et un triplet $(\mathrm{A}, \mathrm{B}, \mathrm{C}) \in \mathbb{G}_1^2 \times \mathrm{G}_2$ et renvoyant un élément $\mathrm{D} \in \mathbb{G}_\mathrm{T}$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{CBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{CBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$

$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e) \leftarrow \mathsf{G}_{\mathrm{BDH}}(k)$
$params \leftarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e)$
$a \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{A} = g_1^a$
$b \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{B} = g_1^b$
$c \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{C} = g_2^c$
$\mathrm{D} \leftarrow \mathcal{A}(k, params, \mathrm{A}, \mathrm{B}, \mathrm{C})$
`Si` $\mathrm{D} = e(g_1, g_2)^{abc}$ renvoie 1
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{CBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{CBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0, 1]$, le problème CBDH est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}^{\mathrm{CBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) \geq \varepsilon$$

.

Le problème CBDH se ramène au problème CDH, c'est-à-dire que le problème CBDH est plus facile que le problème CDH.

2.1.3.2 Problème Diffie-Hellman bilinéaire décisionnel

De même, il existe une version décisionnelle de ce problème. Le problème Diffie-Hellman bilinéaire décisionnel est défini comme suit :

Définition 70 (DBDH). *Soient $\mathbb{G}_1$ et $\mathbb{G}_2$ deux groupes d'ordre premier q, engendrés par g_1 et g_2 respectivement. Étant donnés $(\mathrm{A} = g_1^a, \mathrm{B} = g_1^b, \mathrm{C} = g_2^c, \mathrm{D}) \in \mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{G}_\mathrm{T}$ le* problème Diffie-Hellman bilinéaire décisionnel *(DBDH) consiste à décider si $e(g_1, g_2)^{abc} = \mathrm{D}$ est vrai.*

Définition 71 (Expérience DBDH). *Soient k un entier et $\mathsf{G}_{\mathrm{BDH}}$ un générateur de groupe BDH. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un ensemble $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e)$ issu du générateur $\mathsf{G}_{\mathrm{BDH}}$, un triplet $(\mathrm{A}, \mathrm{B}, \mathrm{C}) \in \mathbb{G}^3$ et renvoyant un bit $b \in \{0, 1\}$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{DBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{DBDH}-r}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$

$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e) \leftarrow \mathsf{G}_{\mathrm{BDH}}(k)$
$params \leftarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e)$
$a \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{A} = g_1^a$
$b \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{B} = g_1^b$
$c \overset{\mathrm{R}}{\leftarrow} [1, q-1], \mathrm{C} = g_1^c$
`Si` $r = 0$ `faire` $\mathrm{D} = e(g_1, g_2)^{abc}$
`Si` $r = 1$ `faire` $\mathrm{D} \overset{\mathrm{R}}{\leftarrow} \mathbb{G}_\mathrm{T}$
$b \leftarrow \mathcal{A}(k, params, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D})$
Renvoie b.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{DBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = \left| \Pr[\mathbf{Exp}^{\mathrm{DBDH}-1}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = 1] - \Pr[\mathbf{Exp}^{\mathrm{DBDH}-0}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = 1]. \right|$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0, 1]$, le problème DBDH est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}^{\mathrm{DBDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) \geq \varepsilon$$

.

2.1.3.3 Problème Diffie-Hellman flexible

Le problème q-Diffie-Hellman flexible (q-SDH) a été introduit par Boneh et Boyen pour prouver la sécurité de leur schéma de signature [BB04]. On définit le problème q-SDH de la manière suivante :

Définition 72 (Problème q-SDH). *Soit $x \in [1, q-1]$. Étant donnés un entier l et l'ensemble $(g, g^x, g^{x^2}, \ldots, g^{x^l}) \in \mathbb{G}^{l+1}$, le* problème q-SDH *consiste à calculer un couple $(g^{\frac{1}{x+h}}, h)$ pour un certain $h \in [0, q-1]$.*

Définition 73 (Expérience q-SDH). *Soient k un entier et $\mathsf{G}_{\mathrm{BDH}}$ un générateur de groupe BDH. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un ensemble*

$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ *issu du générateur* $\mathsf{G}_{\mathrm{BDH}}$ *et un ensemble* $(g_2^x, g_2^{x^2}, \ldots, g_2^{x^l})$ *et renvoyant un couple* $(h, \mathrm{X}) \in (\mathbb{G}, [0, q-1])$. *On associe à cet adversaire l'expérience* $\mathbf{Exp}^{q-\mathrm{SDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$ *décrite ci-dessous.*

Expérience $\mathbf{Exp}^{q-\mathrm{SDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$

$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{G}_{\mathrm{BDH}}(k)$
$params \leftarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$
$x \overset{\mathrm{R}}{\leftarrow} [1, q-1]$
$(h, \mathrm{X}) \leftarrow \mathcal{A}(k, params, g_2^x, g_2^{x^2}, \ldots, g_2^{x^l})$
Si $\mathrm{X} = g_1^{\frac{1}{x+h}}$ renvoie 1
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{q-\mathrm{SDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{q-\mathrm{SDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ *et* $\varepsilon \in [0,1]$, *le problème q-SDH est dit* (k, τ, ε)*-difficile, s'il n'existe pas d'adversaire* $\mathcal{A}$ *fonctionnant en un temps* τ *et ayant un avantage*

$$\mathbf{Adv}^{q-\mathrm{SDH}}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) \geq \varepsilon$$

.

Le problème q-SDH peut être vu, pour le logarithme discret, comme l'analogue du RSA flexible pour la factorisation. De même, l'hypothèse associée est appelée *hypothèse q-Diffie-Hellman fort* (*q-Strong Diffie-Hellman assumption*).

La définition donnée ici reprend celle de [BB04] et fait donc appel à un générateur de groupe BDH. Il est cependant tout à fait possible de définir le problème dans un groupe d'ordre premier.

2.1.3.4 Problème Diffie-Hellman externe

Cette hypothèse a été introduite pour la première fois dans la version longue de [BBS04] afin de prouver la sécurité du schéma présenté. Cette hypothèse repose sur l'existence de groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ tels que le problème DDH soit dur dans $\mathbb{G}_1$ même s'il est facile dans $\mathbb{G}_2$ et qu'il soit toujours possible d'utiliser des applications bilinéaires dans ces groupes.

Définition 74 (Hypothèse XDH)**.** *Étant donné un générateur de groupe BDH qui renvoie les paramètres* $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, *tels que le problème DDH soit facile dans* $\mathbb{G}_2$, *l'hypothèse* Diffie-Hellman eXterne *(XDH) suppose que le problème DDH est dur dans le groupe* $\mathbb{G}_1$.

Ceci implique qu'il n'existe pas d'isomorphisme calculable efficacement $\psi' : \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Il est aussi possible de définir une version plus forte du problème XDH en supposant que le problème DDH est aussi dur dans le groupe $\mathbb{G}_2$.

2.1.4 Autres problèmes

Nous définissons dans cette section quelques problèmes plus exotiques qui ont souvent été définis pour prouver la sécurité d'un cryptosystème particulier et de ses dérivés. Ces problèmes sont des variantes souvent plus faibles des problèmes que nous avons énoncés ci-dessus.

2.1.4.1 Problème décisionnel linéaire

Le *problème décisionnel linéaire* a été introduit par Boneh, Boyen et Sacham [BBS04] pour prouver la sécurité de leur schéma de signature de groupe que nous étudierons au chapitre 4.

Définition 75 (Problème décisionnel linéaire)**.** *Soient $\mathbb{G}$ un groupe cyclique d'ordre premier p et g, h et f des générateurs de $\mathbb{G}$. Étant donnés g^a, h^b, f^c, le* problème décisionnel linéaire*, noté DLin, consiste à décider si $a + b = c$ est vrai ou non.*

Définition 76 (Expérience décisionelle linéaire)**.** *Soient k un entier et $\mathsf{G}_{\mathrm{BDH}}$ un générateur de groupe BDH. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un ensemble $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, e)$ issu du générateur $\mathsf{G}_{\mathrm{BDH}}$, un 6-uplet $(g, h, f, \mathrm{G}, \mathrm{H}, \mathrm{F}) \in \mathbb{G}$ et renvoyant bit $b \in \{0, 1\}$. On associe à cet adversaire l'expérience $\mathbf{Exp}^{\mathrm{DL}in}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}^{\mathrm{DL}in-r}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k)$

$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, e) \leftarrow \mathsf{G}_{\mathrm{BDH}}(k)$
$params \leftarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, e)$
$g, h, f \overset{\mathrm{R}}{\leftarrow} \mathbb{G}_1$
$a \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p, \mathrm{G} = g^a$
$b \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p, \mathrm{H} = h^b$
`Si` $r = 0$ `faire` $\mathrm{F} = h^{a+b}$
`Si` $r = 1$ `faire` $\mathrm{F} \overset{\mathrm{R}}{\leftarrow} \mathbb{G}_1$
$b \leftarrow \mathcal{A}(k, params, g, h, f, \mathrm{G}, \mathrm{H}, \mathrm{F})$
Renvoie b.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{DL}in}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = \left| \Pr[\mathbf{Exp}^{\mathrm{DL}in-1}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = 1] - \Pr[\mathbf{Exp}^{\mathrm{DL}in-0}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) = 1] \right|.$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0, 1]$, le problème DLin est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}^{\mathrm{DL}in}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}(k) \geq \varepsilon$$

.

2.1.4.2 Hypothèse décisionnelle du résidu composite

Cette hypothèse a été introduite par Paillier [Pai99] afin de prouver la sécurité de son schéma de chiffrement. Nous donnons ici la définition de l'hypothèse sans toutefois détailler l'avantage de l'attaquant. Cette hypothèse repose sur la difficulté de décider si un élément de $\mathbb{Z}_{n^2}^*$ est une puissance n-ème modulo n^2.

Définition 77 (Hypothèse DCR)**.** *Soit un entier $n \in \mathbb{Z}$. L'*hypothèse décisionnelle du résidu composite *(DCR) affirme qu'il est difficile de distinguer $\mathbb{Z}_{n^2}^n$ de $\mathbb{Z}_{n^2}^*$ où $\mathbb{Z}_{n^2}^n = \{z \in \mathbb{Z}_{n^2}^* | \exists y \in \mathbb{Z}_{n^2}^* : z = y^n \mod n^2\}$.*

2.1.4.3 Hypothèse LRSW

Cette hypothèse est due à Lysyanskaya, Rivest, Sahai et Wolf [LRSW99] afin de prouver la sécurité de leur schéma.

Définition 78 (Hypothèse LRSW)**.** *Soient $\mathbb{G}$ un groupe d'ordre premier q et g un générateur de $\mathbb{G}$. Étant donnés $\mathrm{X} = g^x$, $\mathrm{Y} = g^y$, un oracle $\mathsf{O}_{\mathrm{X,Y}}^{\mathsf{LRSW}}$ qui prend en entrée $m \in \mathbb{Z}_q$ et renvoie un triplet $\mathrm{A} = (a, a^y, a^{x+mxy})$ pour un a aléatoire, l'*hypothèse LRSW *assume qu'il est difficile pour un adversaire de renvoyer un triplet $\mathrm{A}' = (a', a'^y, a'^{x+m'xy})$ si m n'a pas été donné en entrée à l'oracle.*

Définition 79 (Expérience LRSW)**.** *Soient k un entier et $\mathsf{G}_{\mathrm{BDH}}$ un générateur de groupe BDH. Soit $\mathcal{A}$ un attaquant prenant en entrée un paramètre de sécurité k, un 7-uplet $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e)$ issu du générateur $\mathsf{G}_{\mathrm{BDH}}$, deux éléments $\mathrm{X} = g^x$, $\mathrm{Y} = g^y$ et ayant accès à un oracle $\mathsf{O}_{\mathrm{X,Y}}^{\mathsf{LRSW}}$ et renvoyant un bit $b \in \{0,1\}$. On associe à cet adversaire l'expérience $\mathbf{Exp}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}^{\mathrm{LRSW}}(k)$ décrite ci-dessous.*

Expérience $\mathbf{Exp}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}^{\mathrm{LRSW}}(k)$

$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e) \leftarrow \mathsf{G}_{\mathrm{BDH}}(k)$
$params \leftarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e)$
$x \overset{\mathrm{R}}{\leftarrow} \mathbb{G}_1, \mathrm{X} = g_1^x$
$y \overset{\mathrm{R}}{\leftarrow} \mathbb{G}_1, \mathrm{Y} = g_1^y$
$(\mathrm{Q}, m, a, b, c) \leftarrow \mathcal{A}(k, params, \mathrm{X}, \mathrm{Y} : \mathsf{O}^{\mathsf{LRSW}_{\mathrm{X,Y}}})$
`Si` $(m \notin \mathrm{Q}) \wedge (a \in \mathbb{G}_1), \wedge (b = a^x) \wedge, (c = a^{x+mxy})$
renvoie 1
`Sinon` renvoie 0.

L'avantage de l'adversaire est donné par

$$\mathbf{Adv}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}^{\mathrm{LRSW}}(k) = \Pr[\mathbf{Exp}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}^{\mathrm{LRSW}}(k) = 1].$$

Étant donnés $\tau \in \mathbb{N}$ et $\varepsilon \in [0,1]$, le problème q-SDH est dit (k, τ, ε)-difficile, s'il n'existe pas d'adversaire $\mathcal{A}$ fonctionnant en un temps τ et ayant un avantage

$$\mathbf{Adv}_{\mathsf{G}_{\mathrm{BDH}},\mathcal{A}}^{\mathrm{LRSW}}(k) \geq \varepsilon$$

.

Lysysankaya *et al.* ont montré dans [LRSW99] que le problème associé à cette hypothèse est difficile, à condition que l'ordre du groupe q ne soit pas divisible par un premier petit.

2.2 Les preuves de sécurité

Maintenant que les hypothèses calculatoires sont posées, nous nous intéressons à la façon dont elles peuvent être utilisées pour prouver la sécurité d'un protocole.

Nous étudions aussi les divers environnements dans lesquels l'adversaire peut se placer pour mener son attaque.

2.2.1 Sécurité réductionniste

En 1984, Goldwasser et Micali ont posé les premiers fondements de la sécurité prouvée dans [GM84]. Le principe consiste à faire reposer la sécurité sur la difficulté d'un problème comme ceux que nous venons d'énoncer, et ce, de façon prouvée. Plus formellement, on montre qu'un adversaire $\mathcal{A}$ qui parviendrait à casser une des propriétés de sécurité pourrait être utilisé comme sous-programme d'un algorithme efficace pour résoudre un problème difficile.

Les premières notions de sécurité conformes à ce principe ont été définies pour le chiffrement [GM84] et la signature [GMR84], [GMR88]. Cependant, les réductions proposées, étaient de coût trop élevé, bien que polynomial. Par la suite, Bellare et Rogaway ont introduit le concept de *sécurité exacte* [BR96] puis Ohta et Okamoto celui de *sécurité concrète* [OO98] qui a permis d'obtenir des résultats de sécurité plus pratiques. Pointcheval, en 2001, a même introduit la notion de *sécurité pratique* [Poi01].

Considérons un attaquant $\mathcal{A}$ atteignant son but en un temps t et supposons qu'une réduction permet de résoudre le problème difficile en un temps $f(t)$. On définit alors trois notions de sécurité :

- la sécurité *asymptotique*, lorsque f est polynomiale (majorée par un polynôme en t),
- la sécurité *exacte*, lorsque f est explicite,
- la sécurité *pratique*, lorsque f est « petite » (par exemple linéaire).

Si la réduction est polynomiale on peut alors dire qu'attaquer le protocole est au moins aussi dur que de résoudre le problème difficile. Afin de construire la meilleure réduction possible, il est important de définir précisément les buts que l'adversaire doit atteindre, comme nous l'avons fait pour les primitives classiques dans la section 1.3.1.

2.2.2 Modèles de sécurité

Cette section décrit les différents environnements dans lesquels l'adversaire est susceptible de se trouver pour réaliser ses attaques. Afin de prouver la sécurité d'un schémas, il peut parfois être nécessaire de faire certaines hypothèses sur les primitives utilisés. Dans ce cas là, celles-ci sont idéalisées et représentées par des oracles tout-puissants auxquels l'adversaire peut faire appel.

2.2.2.1 Modes d'attaque

Lorsqu'on décrit une attaque il faut préciser si l'adversaire peut, ou non, lancer plusieurs exécutions du protocole en parallèle. On définit alors deux modes d'attaque possibles :

- mode *en série* : l'adversaire attend que l'exécution en cours soit achevée avant de pouvoir en relancer une autre,
- mode *concurrent* ou *parallèle* : l'adversaire est autorisé à initier de nouvelles instances du protocole sans que les précédentes soient forcément terminées.

Le mode concurrent caractérise des adversaires plus puissants face auxquels il n'est pas toujours possible de prouver la sécurité du schéma considéré.

2.2.2.2 Modèle de l'oracle aléatoire

Comme nous l'avons dit un schéma est prouvé sûr si l'on peut montrer que sa sécurité repose sur un problème difficile. De nombreux schémas utilisent, dans leur construction, des fonctions de hachage, aussi est-il important de s'assurer qu'elles n'introduisent pas de faille dans le système. La sécurité n'est prouvée qu'en supposant qu'il est impossible de trouver de collision à partir de la fonction utilisée.

C'est dans ce but que Fiat et Shamir [FS86] ont introduit en 1986, le concept d'oracle aléatoire que Bellare et Rogaway ont modélisé en 1993 [BR93]. Il consiste, informellement, à assimiler les fonctions de hachage utilisées à des fonctions aléatoires. La sécurité du schéma ne dépend donc pas de la fonction de hachage spécifiée. Celle-ci est définie comme un oracle envoyant des réponses parfaitement aléatoires, avec comme seule restriction de toujours renvoyer la même réponse pour une même entrée. Ainsi l'attaquant n'apprend aucune information à partir des valeurs hachées qu'il a déjà obtenues, notamment il ne peut pas construire de hachés pour de nouvelles valeurs.

Bien que ce modèle soit très largement répandu certains le considèrent trop restrictif. Canetti, Goldreich et Halevi ont montré dans [CGH98] et [CGH04] qu'il était possible de construire des schémas prouvés sûrs dans le modèle de l'oracle aléatoire tels que toute implémentation de l'oracle rende le schéma non sûr en pratique. Bellare, Boldyreva et Palacio, ont proposé dans [BBP04] des constructions atteignant certaines propriétés uniquement dans le modèle de l'oracle aléatoire, autrement dit, relâcher cette contrainte implique qu'il est impossible de prouver la sécurité du schéma. Cependant, tous ces contre-exemples étant obtenus à partir de constructions très spécifiques et très artificielles ils ne remettent pas vraiment en cause ce modèle.

2.2.2.3 Modèle CRS

Dans le modèle CRS (pour *Common Reference String*) on suppose que toutes les parties impliquées dans le protocole ont accès aux mêmes données en entrées distribuées par un tiers de confiance. Ce sont ces entrées qui composent la « chaîne de référence » et sa distribution est supposée idéale et sûre. De plus, aucun participant (ce qui inclut les adversaires) ne peut avoir connaissance des trappes utilisées pour la construction de cette chaîne. Celles-ci sont connues uniquement du simulateur dans les preuves de sécurité, c'est-à-dire de l'autorité générant les paramètres.

En pratique, on suppose qu'un tiers de confiance génère la chaîne de référence crs à l'aide d'un générateur K, c'est-à-dire $(crs, \tau) \leftarrow \mathrm{K}(1^k)$ où k est un paramètre de sécurité, et garde secrète la trappe τ. La chaîne crs est rendue publique et toutes les parties la reçoivent en entrée additionnelle.

2.2.2.4 Modèle standard

Le modèle standard, contrairement au modèle de l'oracle aléatoire et du modèle CRS, ne fait aucune hypothèse sur les fonctions de hachage utilisées (autres que celles figurant dans leur définition et notamment la propriété d'anti-collision) ni sur la mise en place des paramètres. Cela permet d'atteindre des propriétés de sécurité plus fortes, puisqu'elles reposent uniquement sur des hypothèses calculatoires et non plus sur des constructions idéalisées. La contrepartie est que les constructions obtenues sont souvent moins efficaces, défaut que de nombreux chercheurs s'emploient à surmonter, comme nous le verrons par la suite dans ce mémoire.

2.2.2.5 Modèle de la composabilité universelle

La notion de composabilité universelle (*Universal Composability*), notée UC, a été introduite par Canetti en 2001 [Can01]. Elle a pour but de garantir la sécurité d'un protocole dans les cas particuliers suivants :

- le même protocole est exécuté avec des entrées identiques[1]/différentes, avec des parties identiques/différentes, en série, en parallèle ou en mode concurrent ;
- le protocole est appelé en tant que sous-programme d'un autre protocole, en mode concurrent ou non ;
- des protocoles arbitraires sont exécutés dans le même système, sans coordination.

Le protocole est plongé dans un environnement qui représente toutes les informations extérieures à l'exécution du protocole. C'est cet environnement qui est chargé de donner les entrées aux joueurs et qui récupère les sorties en fin d'exécution. On considère alors d'un côté le *monde réel*, c'est-à-dire le protocole, les joueurs et l'adversaire et d'un autre le *monde idéal* avec un adversaire idéal et une *fonctionnalité idéale* qui exécute parfaitement le protocole et ne peut être corrompue. On dit alors qu'un protocole π réalise de manière sûre une tâche $\mathcal{F}$ si, pour tout adversaire $\mathcal{A}$, il existe un adversaire idéal $\mathcal{S}$ tel qu'aucun environnement $\mathcal{Z}$ ne puisse savoir avec probabilité non-négligeable s'il est en train d'interagir avec π et $\mathcal{A}$ ou avec $\mathcal{F}$ et $\mathcal{S}$.

Ce modèle est de plus en plus utilisé pour prouver la sécurité des protocoles et il existe maintenant des définitions pour la plupart des primitives. Certains le considèrent même comme étant le modèle ultime que tout protocole devrait atteindre.

2.2.2.6 Autres modèles

Le modèle de l'oracle aléatoire n'est pas le seul modèle à poser des conditions sur les primitives utilisées pour la construction des cryptosystèmes. Le *modèle générique*, par exemple, épure les opérations arithmétiques de toute propriété particulière qui pourrait

[1] Ce cas de figure est appelé *Join State Universal Composability* (JUC) et a été introduit par Canetti et Rabin [CR03]

être exploitée à mauvais escient. Le *modèle du chiffrement idéal* quant à lui considère des blocs de chiffrement symétriques idéaux en ce sens qu'ils sont considérés comme des permutations aléatoires. Ces modèles ne nous étant pas utiles par la suite, nous ne les détaillons pas plus.

2.2.3 Techniques de preuve

Nous décrivons dans cette section deux techniques couramment utilisées dans les preuves.

2.2.3.1 Preuves par jeux

La *preuve par jeux* a été utilisée pour la première fois en cryptographie par Kilian et Rogaway [KR96] et formalisé par Shoup [Sho00] afin de clarifier certaines démonstrations. Cette technique n'est cependant pas toujours applicable à toutes les preuves et constitue une méthode parmi d'autres.

De manière générale, la définition de sécurité est liée à une expérience particulière et l'avantage de l'adversaire est alors défini par la probabilité que l'adversaire a de réussir cette expérience. Une preuve par séquence de jeux se déroule alors de la manière suivante. On construit tout d'abord un Jeu 0 qui représente l'attaque originale, en prenant en compte la définition de l'adversaire et les paramètres du système. On définit alors S_0 comme étant l'événement S lié à ce jeu (typiquement « l'attaque a réussi »). À partir de ce Jeu 0, on dérive les jeux Jeu_i $i = 1, \ldots, n-1$ qui définissent les événements S_i de manière à ce que la probabilité $\Pr[\mathrm{S}_i]$ soit très proche de la probabilité $\Pr[\mathrm{S}_{i+1}]$. Et ainsi, de proche en proche, on arrive au jeu Jeu_n construit de tel sorte que la probabilité $\Pr[\mathrm{S}_n]$ est égale ou est très proche de la probabilité de l'événement ciblé (souvent 0 ou $1/2$).

La probabilité étant calculée à chaque étape en fonction de la précédente, il est possible de calculer précisément la probabilité $\Pr[\mathrm{S}] = \Pr[\mathrm{S}_0]$.

Le calcul d'une probabilité à l'autre se fait souvent en supposant que les deux événements sont identiques à moins qu'un certain événement F se produise, ce qui mathématiquement s'écrit de la manière suivante :

$$\mathrm{S}_i \wedge \neg \mathrm{F} \iff \mathrm{S}_{i+1} \wedge \neg \mathrm{F}.$$

L'estimation de la probabilité $\Pr[\mathrm{S}_{i+1}]$ fait alors appel au lemme suivant :

Lemme 2 (Lemme des différences)**.** *Soient* A, B *et* F *trois événements tels que de plus que* $\mathrm{A} \wedge \neg \mathrm{F} \iff \mathrm{B} \wedge \neg \mathrm{F}$*. Alors on a*

$$|\Pr[\mathrm{A}] - \Pr[\mathrm{B}]| \leq \Pr[\mathrm{F}].$$

Les preuves par jeux sont aujourd'hui couramment utilisées et par de nombreux auteurs. Nous utiliserons d'ailleurs cette technique pour l'une de nos preuves.

Nous renvoyons à l'article de Shoup [Sho06] pour une description plus complète de cette technique.

2.2.3.2 Forking Lemma

Nous décrivons dans cette section une autre technique de preuve appelée preuve par *rejeu* ou *oracle replay*. Ce procédé a été introduit par Pointcheval et Stern [PS96b] pour prouver la sécurité de schémas de signature dans le modèle de l'oracle aléatoire. Il est basé sur un lemme principal appelé *lemme de bifurcation* ou *forking lemma*.

Le principe repose sur la sécurité réductionniste que nous avons présentée. La technique consiste donc à construire une machine $\mathcal{M}$, qui, utilisant l'adversaire $\mathcal{A}$ va résoudre un problème difficile.

La technique de rejeu d'oracle se définit informellement de la manière suivante : par un rejeu polynomial de l'attaque avec les mêmes données en entrée et un oracle différent, nous obtenons deux signatures d'une forme spécifique qui permettent de résoudre le problème difficile sous-jacent.

La probabilité d'obtenir deux signatures différentes grâce au rejeu est donnée par le lemme de bifurcation :

Lemme 3 (Lemme de bifurcation - *Forking lemma*)**.** *Soit $\mathcal{A}$ une machine de Turing probabiliste s'exécutant en temps polynomial, qui reçoit en entrée les données publiques du système. Si $\mathcal{A}$ peut trouver, avec probabilité non négligeable, une signature valide $(m, \sigma_1, h, \sigma_2)$, alors, avec probabilité non négligeable, un rejeu de la machine, prenant en entrée le même ruban aléatoire et un oracle différent, retourne deux signatures valides $(m, \sigma_1, h, \sigma_2)$ et $(m, \sigma_1, h', \sigma'_2)$ telles que $h \neq h'$.*

La preuve de ce lemme, donnée dans [PS96a], nécessite le lemme suivant :

Lemme 4 (Lemme de séparation - *Splitting lemma*)**.** *Si* A *est un événement de l'espace* $\mathrm{X} \times \mathrm{Y}$ *tel que* $\Pr_{x,y}[\mathrm{A}(x,y)] \geq \varepsilon$*, alors, pour tout* $\alpha < \varepsilon$*, en notant*

$$\mathrm{X}_0 = \left\{ a \in \mathrm{X} \middle| \Pr_y[\mathrm{A}(x,y)|x = a] \geq \varepsilon - \alpha \right\},$$

i) $\Pr_x[x \in \mathrm{X}_0] \geq \alpha$,
ii) $(\forall a \in \mathrm{X}_0) \Pr_y[\mathrm{A}(a,y)] \geq \varepsilon - \alpha$,
iii) $\Pr_{x,y}[x \in \mathrm{X}_0 | \mathrm{A}(x,y)] \geq \frac{\alpha}{\varepsilon}$.

L'énoncé du lemme de bifurcation impose que le schéma de signature produise des signatures de la forme $(m, \sigma_1, h, \sigma_2)$ où $h = f(m, \sigma_1)$ si f est l'oracle aléatoire auquel la machine adresse ses requêtes.

L'énoncé que nous avons donné ici est l'énoncé le plus simple. Il est aussi possible, comme nous le verrons dans la suite de ce mémoire, de l'adapter au cas des attaques à messages choisis et à d'autres formes de signatures. Mais il est surtout possible de définir de manière plus précise les réductions polynomiales. Nous renvoyons le lecteur à la thèse de Pointcheval [Poi96] pour une étude plus précise du coût de ces réductions.

Chapitre 3

Quelques systèmes utiles

Dans ce chapitre, nous présentons différents systèmes cryptographiques qui seront nécessaires aux constructions que nous présenterons dans la suite de ce mémoire.

Sommaire

3.1 Fonctions particulières

3.1.1 Engagement

Le protocole d'engagement le plus couramment utilisé est celui dû à Pedersen [Ped91]. Nous en donnons ici une description.

Setup

Les paramètres du système sont donnés par :

- soient $\mathbb{G}$ un groupe cyclique d'ordre premier q et g un générateur de $\mathbb{G}$,
- g et h deux générateurs de $\mathbb{G}$.

Commit
Pour s'engager sur le message $m \in \mathbb{Z}_q$ de son choix, le prouveur choisit une valeur aléatoire $r \in \mathbb{Z}_q$ et calcule $\mathrm{C} = g^m h^r$ et l'envoie au vérificateur.

Decommit
Le prouveur envoie les valeurs m et r afin que le vérificateur vérifie l'égalité : $\mathrm{C} = g^m h^r$.

La sécurité de ce protocole a été étudiée dans [Ped91].

Engagements extractables Les *engagements extractables* constituent une variante des schémas d'engagement permettant de casser la propriété d'indistinguabilité. La différence provient de l'existence d'une trappe sk associée à chaque clé publique pk permettant à son propriétaire d'extraire le message m à partir de l'engagement. La définition est la même si ce n'est que l'algorithme Setup renvoie aussi une clé privée sk et que l'on ajoute une procédure Extract prenant en entrée la clé secrète sk et l'engagement c et renvoyant le message : $m \leftarrow \mathsf{Extract}(c, sk)$.

Ainsi, n'importe quel schéma de chiffrement peut être utilisé comme engagement extractable.

À l'inverse on peut aussi définir des schémas d'*engagement à trappe* qui permettent de casser la propriété de résistance aux collisions.

3.1.2 Fonction pseudo-aléatoire

Nous utiliserons dans ce mémoire la fonction pseudo-aléatoire introduite par Dodis et Yampolskiy dans [DY05]. Soient $\mathbb{G}$ un groupe d'ordre premier q, g un générateur de ce groupe et $s \in \mathbb{Z}_q$ une « graine ». Le fonction pseudo-aléatoire $f^{\mathrm{DY}}_{g,s}$ prend en entrée une valeur $x \in \mathbb{Z}_q$ et renvoie $f^{\mathrm{DY}}_{g,s}(x) = g^{\frac{1}{s+x+1}}$.

Les propriétés de la fonction donnée dans la définition 39 sont montrées sous l'hypothèse y-DDHI.

3.2 Chiffrement

Le chiffrement ElGamal est un algorithme à clé publique basé sur l'échange de clé Diffie-Hellman [DH76]. Il a été décrit par ElGamal en 1985 dans [ElG85].

3.2.1 ElGamal

Setup
Les paramètres sont définis comme suit :
- $\mathbb{G}$ est un groupe cyclique d'ordre q premier et g est un générateur de $\mathbb{G}$,
- x est un élément de $\mathbb{Z}_q$,
- $h = g^x$.

La clé privée de chiffrement est x et la clé publique est (g, h).

Encrypt
Afin de chiffrer un message $m \in \mathbb{G}$, A procède de la manière suivante :
- il choisit un élément aléatoire $r \in \mathbb{Z}_q$,
- il calcule $\mathrm{M}_1 = mh^r$ et $\mathrm{M}_2 = g^r$.

Le chiffré de m est le couple $(\mathrm{M}_1, \mathrm{M}_2)$.

Decrypt
B déchiffre le message en calculant

$$m = \mathrm{M}_1/{\mathrm{M}_2}^x.$$

La validité du chiffrement se montre à l'aide de la suite d'égalités suivante :

$$\frac{\mathrm{M}_1}{(\mathrm{M}_2)^x} = \frac{mh^r}{(g^r)^x} = \frac{mh^r}{g^{xr}} = \frac{mh^r}{h^r} = m.$$

La sécurité de ce schéma est basée sur le problème décisionnel Diffie-Hellman (DDH). Dans la version de base, le schéma est malléable et n'est prouvé sûr que contre des attaques à textes clairs choisis. Nous aurons besoin, dans la suite de ce mémoire d'un schéma de chiffrement sémantiquement sûr contre des attaques à chiffrés choisis adaptatives. C'est pourquoi, nous utiliserons une variante, appelée le double chiffrement ElGamal, dont voici le fonctionnement :
Setup
Les paramètres sont définis comme suit :
- $\mathbb{G}$ est un groupe cyclique d'ordre q premier et g un générateur de $\mathbb{G}$,
- x, y sont des éléments de $\mathbb{Z}_q^2$,
- $h_1 = g^x$ et $h_2 = g^y$.

La clé privée de chiffrement est le couple (x, y) et la clé publique est (g, h_1, h_2).

Encrypt
Afin de chiffrer un message $m \in \mathbb{G}$, A le chiffre deux fois avec le schéma ElGamal de base, en utilisant deux aléas différents :
- il choisit $r, s \in_{\mathrm{R}} \mathbb{Z}_q$,
- il calcule $\mathrm{M}_1 = mh_1^r$, $\mathrm{M}_2 = g^r$, $\mathrm{M}_3 = mh_2^s$ et $\mathrm{M}_4 = g^s$.

Le chiffré c de m est alors le quadruplet $(\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3, \mathrm{M}_4)$, complété d'une preuve que les messages contenus dans les chiffrés M_1 et M_3 sont bien les mêmes. Cette preuve consiste à prouver l'existence de r et s tels que $\mathrm{M}_2 = g^r$, $\mathrm{M}_4 = g^s$ et $\mathrm{M}_1/\mathrm{M}_3 = h_1^r h_2^{-s}$.

On écrit alors :

$$c = \big(\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3, \mathrm{M}_4, pok(\alpha, \beta : \mathrm{M}_2 = g^\alpha \wedge \mathrm{M}_4 = g^\beta \wedge \mathrm{M}_1/\mathrm{M}_3 = h_1^\alpha h_2^{-\beta})\big).$$

Decrypt
B déchiffre le message c en calculant :

$$m_1 = \mathrm{M}_1/{\mathrm{M}_2}^x, \qquad m_2 = \mathrm{M}_3/{\mathrm{M}_4}^y.$$

Il vérifie que la preuve donnée par A est valide et ensuite que $m_1 = m_2$.

L'ajout de la preuve d'égalité entre les deux chiffrés permet d'obtenir un schéma IND-CCA2, comme l'ont montré Fouque et Pointcheval dans [FP01].

3.2.2 Linéaire

Le chiffrement linéaire a été introduit par Boneh, Boyen et Sacham dans [BBS04]. Ce chiffrement est une extension du chiffrement ElGamal, mais, contrairement à ce dernier, il reste sûr dans les groupes où le problème DDH est facile (typiquement, ce chiffrement est utilisé dans les applications faisant appel à des applications bilinéaires).

Setup

Les paramètres et les clés sont définis de la manière suivante :

- $\mathbb{G}$ est un groupe cyclique d'ordre premier q et g un générateur de $\mathbb{G}$,
- x et y sont deux éléments de $\mathbb{Z}_q^*$,
- $h_1 = g^{1/x}$ et $h_2 = g^{1/y}$.

La clé privée du schéma de chiffrement est le couple (x, y) et la clé publique est le triplet (h_1, h_2, g).

Encrypt

Afin de chiffrer un message $m \in \mathbb{G}$, A procède de la manière suivante :

- il choisit r et $s \in_{\mathrm{R}} \mathbb{Z}_q$,
- il calcule $\mathrm{M}_1 = h_1^r$, $\mathrm{M}_2 = h_2^s$ et $\mathrm{M}_3 = m.g^{r+s}$.

Le chiffrement de m est le triplet $(\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3)$.

Decrypt

Pour déchiffrer le triplet $(\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3)$, B calcule :

$$m = \mathrm{M}_3/(\mathrm{M}_1^x \mathrm{M}_2^y).$$

La validité du schéma est donnée par la suite d'égalités suivante :

$$\frac{\mathrm{M}_3}{(\mathrm{M}_1^x \mathrm{M}_2^y)} = \frac{mg^{r+s}}{(h_1^r)^x (h_2^s)^y} = \frac{mg^{r+s}}{h_1^{xr} h_2^{ys}} = m.$$

En suivant une preuve très proche de celle du schéma de chiffrement ElGamal, il est possible de montrer que le chiffrement linéaire est sémantiquement sûr contre des attaques à textes clairs choisis sous l'hypothèse linéaire décisionnelle.

Comme nous l'avons dit plus haut, nos futures constructions nécessiteront des schémas de chiffrement IND-CCA2. C'est pourquoi, nous utiliserons le même type de variante que ci-dessus, le double chiffrement linéaire. Le message est chiffré deux fois, en utilisant deux clés secrètes différentes et est complété d'une preuve d'égalité des deux chiffrés.

Setup

Les paramètres et les clés sont définis de la manière suivante :

- $\mathbb{G}$ est un groupe cyclique d'ordre premier q et g_1 et g_2 sont deux générateurs de $\mathbb{G}$,

- x_1, x_2, y_1 et y_2 sont des éléments de $\mathbb{Z}_q$,
- $h_1 = g_1^{1/x_1}$, $h_2 = g_1^{1/y_1}$, $h_3 = g_2^{1/x_2}$ et $h_4 = g_2^{1/y_2}$.

La clé privée du schéma de chiffrement est le quadruplet (x_1, y_1, x_2, y_2) et la clé publique est le quintuplet (h_1, h_2, h_3, h_4, g).

Encrypt
Afin de chiffrer un message $m \in \mathbb{G}$, A procède de la manière suivante :
- il choisit r_1, r_2, s_1 et $s_2 \in_{\mathrm{R}} \mathbb{Z}_p$,
- il calcule $\mathrm{M}_1 = h_1^{r_1}$, $\mathrm{M}_2 = h_2^{s_1}$, $\mathrm{M}_3 = m.g_1^{r_1+s_1}$, $\mathrm{M}_4 = h_3^{r_2}$, $\mathrm{M}_5 = h_4^{s_2}$ et $\mathrm{M}_6 = m.g_2^{r_2+s_2}$.

Le chiffré c de m est alors le 6-uplet $(\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3, \mathrm{M}_4, \mathrm{M}_5, \mathrm{M}_6)$, complété d'une preuve que les messages contenus dans les chiffrés M_3 et M_6 sont bien les mêmes. Cette preuve consiste à prouver l'existence de r_1, s_1 et r_2, s_2 tels que $\mathrm{M}_1 = h_1^{r_1}$, $\mathrm{M}_2 = h_2^{s_1}$, $\mathrm{M}_4 = h_3^{r_2}$, $\mathrm{M}_5 = h_4^{s_2}$ et $\mathrm{M}_3/\mathrm{M}_6 = g_1^{r_1+s_1} g_2^{-(r_2+s_2)}$.

On écrit alors :

$$c = \big(\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3, \mathrm{M}_4, \mathrm{M}_5, \mathrm{M}_6,$$
$$pok(\alpha, \beta, \gamma, \delta : \mathrm{M}_1 = h_1^{\alpha} \wedge \mathrm{M}_2 = h_2^{\beta} \wedge \mathrm{M}_4 = h_3^{\gamma} \wedge \mathrm{M}_5 = h_3^{\delta} \wedge \mathrm{M}_3/\mathrm{M}_6 = g_1^{\alpha+\beta} g_2^{-(\gamma+\delta)})\big).$$

Decrypt
Pour déchiffrer le 6-uplet $(\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3, \mathrm{M}_4, \mathrm{M}_5, \mathrm{M}_6)$, B calcule :

$$m = \mathrm{M}_3/(\mathrm{M}_1^{x_1}\mathrm{M}_2^{y_1}), \qquad m = \mathrm{M}_6/(\mathrm{M}_4^{x_2}\mathrm{M}_5^{y_2}).$$

Il vérifie que la preuve donnée par A est valide et ensuite que $m_1 = m_2$.

Nous disposons donc maintenant de deux schémas de chiffrement IND-CCA2. L'avantage du double chiffrement linéaire face au double chiffrement ElGamal est de pouvoir être utilisé dans des groupes où le problème DDH est facile. En particulier, cela signifie qu'il peut être utilisé sans l'hypothèse XDH, ce qui est préférable dans certaines constructions. En revanche, le chiffré du double chiffrement linéaire comporte six éléments dans un groupe $\mathbb{G}$ d'ordre premier, contrairement au double chiffrement ElGamal qui n'a que quatre éléments. Cette différence de taille peut aussi être déterminante quant au choix du schéma de chiffrement.

3.2.3 Paillier et engagements extractables

Le schéma de chiffrement de Paillier tient son nom de son inventeur [Pai99].

Setup
Les paramètres et les clés du système sont définis de la manière suivante :
- soient p et q deux nombres premiers distincts tels que $p, q > 2, |p| = |q|$ et $\mathrm{pgcd}(pq, (p-1)(q-1)) = 1$,
- soit $n = pq$, et $g = (1+n)$.

La clé publique est alors $pk = n$ et la clé secrète est donnée par $sk = \lambda(n)$ où λ est la fonction de Carmichael (*cf.* définition 13). Par la suite, on notera $\lambda(n) = \lambda$.

Encrypt
Nous introduisons ici l'ensemble $\mathrm{S}_n = \{u | u < n^2, u = 1 \mod n\}$ et la fonction L définie sur S_n par $\mathrm{L}(u) = \frac{u-1}{n}$.
Pour chiffrer un message $m < n$, A procède de la manière suivante :
- il choisit $r \in_{\mathrm{R}} \mathbb{Z}_n^*$,
- il calcule $\mathrm{C} = g^m r^n \mod n^2$.

Le chiffré de m est donné par C.

Decrypt Pour déchiffrer C, l'utilisateur calcule

$$m = \frac{\mathrm{L}(\mathrm{C}^\lambda \mod n^2)}{\mathrm{L}(g^\lambda \mod n^2)} \mod n.$$

On vérifie la validité du schéma comme suit.
En utilisant les propriétés de la fonction λ (*cf.* proposition 2) on obtient que :

$$\begin{aligned} \mathrm{C}^\lambda &= g^{m\lambda} r^{n\lambda} \mod n^2 \\ &= g^{m\lambda} \mod n^2 \\ &= (1+n)^{m\lambda} \mod n^2 \\ &= 1 + mn\lambda \mod n^2. \end{aligned}$$

D'où

$$\mathrm{L}(c^\lambda \mod n^2) = m\lambda \mod n.$$

De même, on obtient

$$\mathrm{L}(g^\lambda \mod n^2) = \lambda \mod n.$$

Ce qui nous donne le résultat

$$\frac{\mathrm{L}(c^\lambda \mod n^2)}{\mathrm{L}(g^\lambda \mod n^2)} \mod n = \frac{m\lambda}{\lambda} \mod n = m.$$

Ce schéma de chiffrement particulier sera utilisé dans nos constructions comme engagement extractable.

3.3 Preuves de connaissance

Nous avons, au chapitre précédent, décrit les preuves de connaissance les plus répandues et les plus couramment utilisées. Nous nous intéressons ici à des preuves plus complexes.

3.3.1 Preuve du OU

Une *preuve du OU* permet de prouver la connaissance d'une valeur parmi un ensemble de valeurs, sans révéler laquelle. Soient $\mathbb{G}$ un groupe et $(g_1, \ldots, g_l)$ des générateurs de $\mathbb{G}$. Soient $\mathrm{C}_1, \ldots, \mathrm{C}_k$ des engagements, tels que $\mathrm{C}_i = \Pi_{j \in \mathrm{J}_i} g_j^{x_{ij}}$ pour $i \in [1, k]$

et où $\emptyset \subsetneq \mathrm{J}_i \subseteq [1,l]$. La preuve du OU prouve alors la connaissance de l'ensemble $\{x_{ij}; j \in \mathrm{J}_i\}$, pour au moins une valeur de i. La preuve interactive est notée :

$$pok\Big(\{\alpha_{ij}; i \in [1,k], j \in \mathrm{J}_i\}; \bigvee_{i=1}^{k} \big(\mathrm{C}_i = \prod_{j \in \mathrm{J}_i} g_j^{\alpha_{ij}}\big)\Big).$$

De telles preuves sont décrites dans [CDS94] et [SCPY94]. Nous détaillons ici comment un prouveur peut montrer sa connaissance d'un logarithme discret parmi l relativement à une même base g_1, c'est-à-dire dans le cas $\mathrm{J}_i = \{1\}$. Cette preuve est due à Cramer *et al.* [CDS94]. Pour simplifier, g_1 sera noté g et les x_{i1} seront notés x_i.

Les paramètres du protocole sont donnés par les éléments suivants :
- κ, un paramètre de sécurité,
- $\mathbb{G}$, un groupe cyclique d'ordre q premier tel que $q > 2^\kappa$,
- g, un générateur de $\mathbb{G}$,
- $y_1, \ldots, y_l \in \mathbb{G}$.

On suppose, pour plus de simplicité que le prouveur $\mathcal{P}$ connaît x_1, le logarithme discret de y_1 en base g. La preuve détaillée ici prouve donc la connaissance d'un logarithme parmi l.

1. $\mathcal{P}$ choisit $r_1 \in \mathbb{Z}_p$ et pose $t_1 = g^{r_1}$.
2. Pour $i \in \{2,l\}$, $\mathcal{P}$ choisit $s_i \in \mathbb{Z}_q$ et $c_i \in \{0,1\}^k$ et pose $t_i = g^{s_i} y_i^{c_i}$.
3. Il envoie alors $t_1, \ldots, t_l$ au vérificateur $\mathcal{V}$.
4. $\mathcal{V}$ renvoie à $\mathcal{P}$ un challenge $c \in \{0,1\}^\kappa$.
5. $\mathcal{P}$ pose $c_1 = c - c_2 - \cdots - c_l$ et $s_1 = r_1 - x c_1 \mod q$. Il envoie les valeurs $(c_i, s_i)_{i=1,\ldots,l}$.
6. $\mathcal{V}$ vérifie que $c = c_1 + \cdots + c_l$ et que pour tout $i = 1, \ldots, l$, $t_i = g^{s_i} y_i^{c_i}$.

3.3.2 Preuves d'inégalités

Nous détaillons ici trois preuves permettant de prouver des inégalités entre deux valeurs.

3.3.2.1 Preuve qu'une valeur engagée est plus petite qu'une valeur connue

Cette preuve permet au prouveur $\mathcal{P}$ d'apporter la preuve qu'il connaît une valeur $x \geq 0$ plus petite qu'une autre valeur connue a, autrement dit que cette valeur appartient à un intervalle de la forme $[0,a]$, sans rien révéler d'autre. On suppose que a est de la forme 2^l et est fixé. De telles preuves ont été décrites dans [Bou00], [CM99], [CFT98].

Soient $\mathbb{G}$ un groupe cyclique d'ordre q premier et g un générateur de $\mathbb{G}$. Soit $h \in \mathbb{G}$ tel que $\log_g h$ ne soit pas connu.

$\mathcal{P}$ calcule tout d'abord un engagement sur la valeur x : $\mathrm{C} = g^x h^r$ avec r un aléa appartenant à $\mathbb{Z}_q$. $\mathcal{P}$ effectue alors la preuve

$$pok(\alpha, \beta : \mathrm{C} = g^\alpha h^\beta \wedge 0 \leq \alpha \leq a).$$

Dans ce mémoire, nous utiliserons une telle preuve dans le cas où la valeur engagée est petite comparée à 2^ℓ. Il est alors possible d'utiliser une technique (différente de celles citées plus haut) consistant à utiliser la représentation binaire de x. La sécurité de la preuve repose alors sur l'hypothèse du logarithme discret, comme Bellare et Goldwasser l'ont montré dans [BG97].

Les paramètres du protocole sont donnés par les éléments suivants :
- k, un paramètre de sécurité,
- $\mathbb{G}$, un groupe cyclique d'ordre q premier tel que $q > 2^k$,
- g, un générateur de $\mathbb{G}$,
- $h \in \mathbb{G}$ tel que $\log_g h$ ne soit pas connu,
- a, un entier de la forme $a = 2^\ell$.

La preuve se déroule de la manière suivante :

1. Le prouveur écrit la valeur x sous la forme $x = x_0 + x_1 2^1 + \cdots + x_{\ell-1} 2^{\ell-1}$.
2. Il choisit des valeurs aléatoires $r, r_0, \ldots, r_{\ell-1} \in \mathbb{Z}_p$ et calcule

$$\begin{aligned} \mathrm{C} &= g^x h^r \\ \mathrm{C}_0 &= g^{x_0} h^{r_0} \\ \mathrm{C}_1 &= g^{x_1} h^{r_1} \\ \cdots & \\ \mathrm{C}_{\ell-1} &= g^{x_{\ell-1}} h^{r_{\ell-1}} \\ \tilde{\mathrm{C}} &= \prod_{i=0}^{\ell-1} \mathrm{C}_i^{2^i}. \end{aligned}$$

3. $\mathcal{P}$ envoie alors $\mathrm{C}, \mathrm{C}_0, \ldots, \mathrm{C}_{\ell-1}$ à $\mathcal{V}$. Notons que l'élément $\tilde{\mathrm{C}}$ peut être calculé par le prouveur et le vérificateur.

   Cette valeur peut s'écrire sous la forme $\tilde{\mathrm{C}} = g^{\tilde{x}} h^{\tilde{r}}$ et par conséquent on a $\mathrm{C}\tilde{\mathrm{C}}^{-1} = g^{x-\tilde{x}} h^{r-\tilde{r}}$.

4. Le prouveur et le vérificateur effectuent alors la preuve interactive suivante :

$$\begin{aligned} pok\big(&\alpha, \beta, \gamma_0, \ldots, \gamma_{\ell-1}, \delta : (\mathrm{C}_0 = h^{\gamma_0} \vee \mathrm{C}_0/g = h^{\gamma_0}) \wedge \ldots \wedge \\ &(\mathrm{C}_{\ell-1} = h^{\gamma_{\ell-1}} \vee \mathrm{C}_{\ell-1}/g = h^{\gamma_{\ell-1}}) \wedge \mathrm{C} = g^\alpha h^\beta \wedge \mathrm{C}\tilde{\mathrm{C}}^{-1} = h^\delta\big). \end{aligned}$$

   Dans la première partie de la preuve, pour chaque i, $\mathcal{P}$ prouve qu'il connaît la valeur x_i et que celle-ci vaut 0 ou 1. Il prouve ensuite que l'engagement $\tilde{\mathrm{C}}$ se décompose bien en les valeurs C_i. Dans la dernière partie de la preuve, $\mathcal{P}$ prouve qu'il connaît le logarithme discret de $\mathrm{C}\tilde{\mathrm{C}}^{-1}$ en base h, c'est-à-dire $r - \tilde{r}$. Il montre ainsi que $x = \tilde{x}$ et par conséquent que $x \leq a = 2^\ell$.

Cette preuve nécessite ℓ preuves du OU. Cependant, chacune de ces preuves du OU sert à montrer la connaissance d'une valeur parmi $\{0, 1\}$ mais dans le cas particulier où la valeur x n'est pas trop grande, ce qui sera le cas dans nos constructions, cette preuve est plus efficace que les constructions de [Bou00], [CM99], [CFT98].

3.3.2.2 Preuve qu'une valeur engagée est plus grande qu'une valeur connue

$\mathcal{P}$ veut prouver à $\mathcal{V}$ qu'il connaît une valeur x plus grande qu'une autre valeur a connue sans dévoiler cette valeur.

Soient $\mathbb{G}$ un groupe de générateur g et $h \in \mathbb{G}$ tel que $\log_g h$ ne soit pas connu. $\mathcal{P}$ veut alors prouver sa connaissance de (x, r) avec r un aléa, tels que $\mathrm{C} = g^x h^r$ et $a \leq x$ où a est un entier défini et connu. Il effectue donc la preuve

$$pok(\alpha, \beta : \mathrm{C} = g^\alpha h^\beta \wedge a \leq \alpha).$$

Nous nous plaçons dans le cas où x et a sont des entiers de taille ℓ (en bits) avec ℓ relativement petit. Cette preuve se déroule suivant le même principe que la preuve précédente. Nous utilisons une décomposition binaire des valeurs a et x et prouvons l'inégalité pour chaque bit.

Les paramètres du protocole sont donnés par les éléments suivants :
- k, un paramètre de sécurité,
- $\mathbb{G}$, un groupe cyclique d'ordre q premier tel que $q > 2^k$,
- g, un générateur de $\mathbb{G}$,
- $h \in \mathbb{G}$ tel que $\log_g h$ ne soit pas connu,
- a, un entier de la forme $a = 2^\ell$.

La preuve se déroule de la manière suivante :

1. $\mathcal{P}$ décompose a et x en $x = x_0 + x_1 2^1 + \ldots + x_{\ell-1} 2^{\ell-1}$ et $a = a_0 + a_1 2^1 + \ldots + a_{\ell-1} 2^{\ell-1}$.
2. Il choisit aléatoirement $r, r_0, \ldots, r_{\ell-1} \in_{\mathrm{R}} \mathbb{Z}_p$ et calcule
$$\begin{aligned} \mathrm{C} &= g^x h^r \\ \mathrm{C}_0 &= g^{x_0} h^{r_0} \\ \mathrm{C}_1 &= g^{x_1} h^{r_1} \\ \ldots & \\ \mathrm{C}_{\ell-1} &= g^{x_{\ell-1}} h^{r_{\ell-1}} \\ \tilde{\mathrm{C}} &= \prod_{i=0}^{\ell-1} \mathrm{C}_i^{2^i}. \end{aligned}$$
Il envoie ensuite à $\mathcal{V}$ les valeurs $\mathrm{C}, \mathrm{C}_0, \ldots, \mathrm{C}_{\ell-1}$. L'élément $\tilde{\mathrm{C}}$ peut être calculé par $\mathcal{P}$ et $\mathcal{V}$. Cette valeur peut s'écrire sous la forme $\tilde{\mathrm{C}} = g^{\tilde{x}} h^{\tilde{r}}$ et par conséquent on a $\mathrm{C}\tilde{\mathrm{C}}^{-1} = g^{x-\tilde{x}} h^{r-\tilde{r}}$.
3. $\mathcal{P}$ et $\mathcal{V}$ mènent ensuite dans la preuve de connaissance suivante
$$\begin{gathered} pok\Big(\alpha, \beta, \gamma_0, \ldots, \gamma_{\ell-1}, \delta : (\mathrm{C}_0 = h^{\gamma_0} \vee \mathrm{C}_0/g = h^{\gamma_0}) \\ \wedge \ldots \wedge \\ (\mathrm{C}_{\ell-1} = h^{\gamma_{\ell-1}} \vee \mathrm{C}_{\ell-1}/g = h^{\gamma_{\ell-1}}) \wedge \mathrm{C} = g^\alpha h^\beta \wedge \mathrm{C}\tilde{\mathrm{C}}^{-1} = h^\delta \wedge \\ \big((\mathrm{C}_{\ell-1}/g = h^{\gamma_{\ell-1}} \wedge a_{\ell-1} = 0) \vee (\mathrm{C}_{\ell-1}/g^{a_{\ell-1}} = h^{\gamma_{\ell-1}} \wedge \mathrm{C}_{\ell-2}/g = h^{\gamma_{\ell-2}} \wedge a_{\ell-2} = 0) \\ \vee \ldots \vee \\ (\mathrm{C}_{\ell-1}/g^{\ell-1} = h^{\gamma_{\ell-1}} \wedge \ldots \wedge \mathrm{C}_1/g^{a_1} = h^{\gamma_1} \wedge \mathrm{C}_0/g = h^{\gamma_0})\big)\Big). \end{gathered}$$

Dans la première partie de la preuve, $\mathcal{P}$ montre qu'il connaît la décomposition binaire de x en montrant que chaque x_i vaut soit 0 ou 1. Dans la deuxième partie, il montre que le bit de poids fort de x a un rang plus élevé que celui de a. Pour cela, il prouve que soit « $x_{\ell-1}$ vaut 1 et $a_{\ell-1}$ vaut 0 »soit que « $x_{\ell-1} = a_{\ell-1}$ et que $x_{\ell-2}$ vaut 1 et que $a_{\ell-2}$ vaut 0 » et ainsi de suite. Afin de ne rien révéler sur la valeur x il est obligé de faire cette preuve du OU pour chaque valeur x_i. Par conséquent, cette preuve comporte 2ℓ preuves du OU et n'est efficace que pour des valeurs x et a petites.

3.3.2.3 Preuve qu'une valeur engagée est plus petite qu'une autre valeur engagée

Cette fois, $\mathcal{P}$ veut prouver à $\mathcal{V}$ qu'il connaît une valeur x plus petite qu'une valeur y, mais sans les révéler. Soit $\mathbb{G}$ un groupe de générateur g et soit $h \in \mathbb{G}$ tel que $\log_g h$ ne soit pas connu. $\mathcal{P}$ cherche donc à montrer que $0 \leq x < y$ où x et y sont engagés dans $\mathrm{C} = g^x h^r$ et $\mathrm{D} = g^y h^w$ avec r et w des aléas. Il effectue donc la preuve

$$pok(\alpha, \beta, \gamma, \delta : \mathrm{C} = g^\alpha h^\beta \wedge \mathrm{D} = g^\gamma h^\delta \wedge 0 \leq \alpha < \gamma).$$

Nous nous plaçons, comme précédemment, dans le cas où x et y sont deux entiers de taille ℓ où ℓ est relativement petit. De même, nous utilisons une décomposition binaire des valeurs a et x et prouvons l'inégalité bit à bit. Les paramètres du protocole sont donnés par les éléments suivants :

- k, un paramètre de sécurité,
- $\mathbb{G}$, un groupe cyclique d'ordre q premier tel que $q > 2^k$,
- g, un générateur de $\mathbb{G}$ d'ordre p,
- $h \in \mathbb{G}$ tel que $\log_g h$ ne soit pas connu,
- a, un entier de la forme $a = 2^\ell$.

$\mathcal{P}$ écrit x et y sous leur forme binaire, c'est-à-dire $x = x_0 + x_1 2^1 + \cdots x_{\ell-1} 2^{\ell-1}$ et $y = y_0 + y_1 2^1 + \cdots + y_{\ell-1} 2^{\ell-1}$.

Le preuve peut se faire de deux manières différentes.

i) En utilisant toutes les inégalités possibles.

1. Le prouveur choisit aléatoirement $r, r_0, \ldots, r_{\ell-1} \in \mathbb{Z}_p$ et $w, w_0, \ldots, w_{\ell-1} \in \mathbb{Z}_p$ et calcule

$$\begin{aligned}
\mathrm{C} &= g^x h^r & \mathrm{D} &= g^y h^w \\
\mathrm{C}_0 &= g^{x_0} h^{r_0} & \mathrm{D}_0 &= g^{y_0} h^{w_0} \\
\mathrm{C}_1 &= g^{x_1} h^{r_1} & \mathrm{D}_1 &= g^{y_1} h^{w_1} \\
&\cdots & & \\
\mathrm{C}_{\ell-1} &= g^{x_{\ell-1}} h^{r_{\ell-1}} & \mathrm{D}_{\ell-1} &= g^{y_{\ell-1}} h^{w_{\ell-1}} \\
\tilde{\mathrm{C}} &= \prod_{i=0}^{\ell-1} \mathrm{C}_i^{2^i} & \tilde{\mathrm{D}} &= \prod_{i=0}^{\ell-1} \mathrm{D}_i^{2^i}.
\end{aligned}$$

Il envoie alors à $\mathcal{V}$ les valeurs $\mathrm{C}, \mathrm{C}_0, \ldots, \mathrm{C}_{\ell-1}$ et $\mathrm{D}, \mathrm{D}_0, \ldots, \mathrm{D}_{\ell-1}$. Comme dans la preuve précédente, les éléments $\tilde{\mathrm{C}}$ et $\tilde{\mathrm{D}}$ peuvent être calculés par $\mathcal{P}$

et $\mathcal{V}$. On remarque aussi que $\tilde{\mathrm{C}}$ et $\tilde{\mathrm{D}}$ peuvent s'écrire sous la forme $\tilde{\mathrm{C}} = g^{\tilde{x}}h^{\tilde{r}}$ et $\tilde{\mathrm{D}} = g^{\tilde{y}}h^{\tilde{w}}$ et par conséquent que $\mathrm{C}\tilde{\mathrm{C}}^{-1} = g^{x-\tilde{x}}h^{r-\tilde{r}}$ et $\mathrm{D}\tilde{\mathrm{D}}^{-1} = g^{y-\tilde{y}}h^{w-\tilde{w}}$.

2. Ensuite, $\mathcal{P}$ et $\mathcal{V}$ effectuent la preuve de connaissance suivante :

$$\begin{gathered}\mathrm{PK}\Big(\alpha,\beta,\gamma_0,\ldots,\gamma_{\ell-1},\delta,\varepsilon,\zeta,\eta_0,\ldots,\eta_{\ell-1},\theta/\\ (\mathrm{C}_0 = h^{\gamma_0} \vee \mathrm{C}_0/g = h^{\gamma_0}) \wedge \ldots \wedge (\mathrm{C}_{\ell-1} = h^{\gamma_{\ell-1}} \vee \mathrm{C}_{\ell-1}/g = h^{\gamma_{\ell-1}}) \wedge\\ (\mathrm{D}_0 = h^{\eta_0} \vee \mathrm{D}_0/g = h^{\eta_0}) \wedge \ldots \wedge (\mathrm{D}_{\ell-1} = h^{\eta_{\ell-1}} \vee \mathrm{D}_{\ell-1}/g = h^{\eta_{\ell-1}}) \wedge\\ \mathrm{C} = g^{\alpha}h^{\beta} \wedge \mathrm{C}\tilde{\mathrm{C}}^{-1} = h^{\delta} \wedge \mathrm{D} = g^{\varepsilon}h^{\zeta} \wedge \mathrm{D}\tilde{\mathrm{D}}^{-1} = h^{\theta} \wedge\\ \big((\mathrm{C}_{\ell-1}/\mathrm{D}_{\ell-1} = h^{\gamma_{\ell-1}}/h^{\eta_{\ell-1}} \wedge \mathrm{C}_{\ell-2} = h^{\gamma_{\ell-2}} \wedge \mathrm{D}_{\ell-2}/g = h^{\eta_{\ell-2}}) \vee\\ (\mathrm{C}_{\ell-1}/\mathrm{D}_{\ell-1} = h^{\gamma_{\ell-1}}/h^{\eta_{\ell-1}} \wedge \mathrm{C}_{\ell-2}/\mathrm{D}_{\ell-2} = h^{\gamma_{\ell-2}}/h^{\eta_{\ell-2}} \wedge \mathrm{C}_{\ell-3} = h^{\gamma_{\ell-3}} \wedge \mathrm{D}_{\ell-3}/g = h^{\eta_{\ell-3}})\\ \vee \ldots \vee\\ (\mathrm{C}_{\ell-1}/\mathrm{D}_{\ell-1} = h^{\gamma_{\ell-1}}/h^{\eta_{\ell-1}} \wedge \ldots \wedge \mathrm{C}_1/\mathrm{D}_1 = h^{\gamma_1}/h^{\eta_1} \wedge \mathrm{C}_0 = h^{\gamma_0} \wedge \mathrm{D}_0/g = h^{\eta_0}))\Big).\end{gathered}$$

Cette preuve repose sur le même fonctionnement que la preuve 3.3.2.1 à la différence que la valeur y est engagée. $\mathcal{P}$ doit donc aussi prouver qu'il connaît la décomposition de y et, en utilisant une méthode similaire à la preuve 3.3.2.2, il prouve que le rang du bit de poids fort de y est plus élevé que celui de y.

Cette preuve a pour inconvénient de comprendre $\mathrm{O}(\ell)$ preuves du OU, ce qui peut être calculatoirement coûteux. C'est pourquoi nous proposons une autre manière de construire cette preuve, dans le cas où $2^{\ell} < p/2$.

ii) En utilisant l'inégalité $y - x - 1 \geq 0$.

1. Le prouveur choisit aléatoirement $r, r_0, \ldots, r_{\ell-1} \in \mathbb{Z}_p$ et $w, w_0, \ldots, w_{\ell-1} \in \mathbb{Z}_p$. On note $u = y - x - 1$ et $u = u_0 + u_1 2 + \ldots + u_{\ell-1}2^{\ell-1}$. Le prouveur calcule ensuite

$$\begin{aligned}&\mathrm{C} = g^{x}h^{r}, && \mathrm{C}_0 = g^{x_0}h^{r_0}, \ldots, \mathrm{C}_{\ell-1} = g^{x_{\ell-1}}h^{r_{\ell-1}}\\ &\mathrm{D} = g^{y}h^{w}, && \mathrm{D}_0 = g^{u_0}h^{w_0}, \ldots, \mathrm{D}_{\ell-1} = g^{u_{\ell-1}}h^{w_{\ell-1}}\\ &\widetilde{\mathrm{C}} = \textstyle\prod_{i=0}^{\ell-1} \mathrm{C}_i^{2^i}, && \widetilde{\mathrm{D}} = \textstyle\prod_{i=0}^{\ell-1} \mathrm{D}_i^{2^i}, \overline{\mathrm{D}} = \mathrm{D}/(g\mathrm{C}).\end{aligned}$$

$\mathcal{P}$ envoie à $\mathcal{V}$ les valeurs $\mathrm{C}, \mathrm{C}_0, \ldots, \mathrm{C}_{\ell-1}$ et $\mathrm{D}, \mathrm{D}_0, \ldots, \mathrm{D}_{\ell-1}$. On note, une fois de plus, que les éléments $\tilde{\mathrm{C}}$, $\tilde{\mathrm{D}}$ et $\overline{\mathrm{D}}$ peuvent être calculés par $\mathcal{P}$ et $\mathcal{V}$. De plus, on remarque que $\overline{\mathrm{D}} = g^{y-x-1}h^{w-r} = g^{u}h^{w-r}$. En notant que $\widetilde{\mathrm{C}} = g^{\tilde{x}}h^{\tilde{r}}$ et $\widetilde{\mathrm{D}} = g^{\tilde{u}}h^{\tilde{w}}$, on obtient que $\mathrm{C}\widetilde{\mathrm{C}}^{-1} = g^{x-\tilde{x}}h^{r-\tilde{r}}$ et $\overline{\mathrm{D}}\widetilde{\mathrm{D}}^{-1} = g^{u-\tilde{u}}h^{w-r-\tilde{w}}$.

2. Ensuite, le prouveur et le vérificateur effectuent la preuve de connaissance suivante :

$$\begin{gathered}pok\Big(\alpha,\beta,\gamma_0,\ldots,\gamma_{\ell-1},\delta,\varepsilon,\zeta,\eta_0,\ldots,\eta_{\ell-1},\theta,\rho,\iota :\\ (\mathrm{C}_0 = h^{\gamma_0} \vee \mathrm{C}_0/g = h^{\gamma_0}) \wedge \ldots \wedge (\mathrm{C}_{\ell-1} = h^{\gamma_{\ell-1}} \vee \mathrm{C}_{\ell-1}/g = h^{\gamma_{\ell-1}}) \wedge\\ (\mathrm{D}_0 = h^{\eta_0} \vee \mathrm{D}_0/g = h^{\eta_0}) \wedge \ldots \wedge (\mathrm{D}_{\ell-1} = h^{\eta_{\ell-1}} \vee \mathrm{D}_{\ell-1}/g = h^{\eta_{\ell-1}}) \wedge\\ \mathrm{C} = g^{\alpha}h^{\beta} \wedge \mathrm{C}\widetilde{\mathrm{C}}^{-1} = h^{\delta} \wedge \mathrm{D} = g^{\varepsilon}h^{\zeta} \wedge \overline{\mathrm{D}} = g^{\rho}h^{\iota} \wedge \overline{\mathrm{D}}\widetilde{\mathrm{D}}^{-1} = h^{\theta}\Big).\end{gathered}$$

$\mathcal{P}$ prouve bien qu'il connaît les représentations binaires de x et y, mais la preuve repose sur la valeur u. En effet, $\mathcal{P}$ prouve qu'il connaît les représentations de u et de x mais aussi que $u \geq 0$. Cette preuve nécessite moins de preuves du OU que la précédente et est

donc plus efficace. En revanche, la restriction $2^\ell < p/2$ est nécessaire afin d'empêcher tout attaquant d'utiliser sa connaissance de l'ordre p de g

Il est aussi possible d'utiliser une preuve telle que décrite par Boudot [Bou00]. Cependant, celles-ci ne peuvent être utilisées que dans des groupes d'ordre inconnu et par conséquent nécessitent des paramètres plus grands. Ainsi, même si la complexité de cette preuve est proportionnelle à O(1), il est, dans notre cas, c'est-à-dire quand ℓ est petit, plus efficace d'utiliser la preuve que nous venons de décrire.

3.4 Signatures

Nous présentons dans cette section plusieurs schémas de signatures ainsi que des protocoles dérivés de ces signatures.

3.4.1 Signature Camenisch-Lysyanskaya sur une valeur engagée

La signature Camenisch-Lysyanskaya que nous étudions ici permet à un signataire de signer des valeurs engagées. Cette signature a été présentée dans [CL04] et sa sécurité repose sur l'hypothèse LRSW.

Le protocole d'engagement utilisé ici est celui de Pedersen [Ped91] que nous avons présenté dans la section 1.4.1. Sa sécurité repose sur l'hypothèse du logarithme discret qui est, en fait, impliquée par l'hypothèse LRSW. De plus, nous avons vu au chapitre précédent qu'il existait des protocoles efficaces pour prouver la connaissance et l'égalité de valeurs engagées.

Nous détaillons ici le schéma de signature sur un message par bloc, c'est-à-dire un message m tel que m s'écrive sous la forme $m = (m_1, \ldots, m_l)$ où le nombre l de blocs est fixé par le système[1]. Il s'agit là du protocole de signature D de [CL04]. Cette signature est obtenue par une interaction entre un signataire qui produit effectivement la signature et l'utilisateur qui souhaite obtenir la signature. De ce fait, les propriétés attendues de la signature sont différentes de celles détaillées dans le chapitre 1. Cette signature est en fait très proche des signatures aveugles que nous étudierons au chapitre 5.

Setup

L'algorithme Setup prend en entrée un paramètre de sécurité k et renvoie les éléments suivants :

- $\mathbb{G}$ (*resp.* G) un groupe cyclique d'ordre p tel que $|p| = \mathrm{O}(2^k)$ et de générateur g (*resp.* g),
- e une application bilinéaire admissible $e : \mathbb{G} \times \mathbb{G} \to \mathsf{G}$.
- $x \in \mathbb{Z}_p$ et $y \in \mathbb{Z}_p$ et pour $1 \leq i \leq l, z_i \in \mathbb{Z}_p$,
- $\mathrm{X} = g^x$, $\mathrm{Y} = g^y$ et pour $1 \leq i \leq l, \mathrm{Z}_i = g^{z_i}$,
- pour $1 \leq i \leq l$, $\mathrm{W}_i = \mathrm{Y}^{z_i}$. Les valeurs W_i ne servent pas directement à la construction du schéma mais servent à prouver la connaissance de la signature.

[1] Cette procédure sert à obtenir une signature sur l messages plus efficacement qu'en itérant le protocole l fois.

La clé privée sk est alors $sk = (x, y, \{z_i\}; i \in \{1, l\})$ et la clé publique pk est $pk = (p, \mathbb{G}, \mathsf{G}, g, \mathsf{g}, e, \mathrm{X}, \mathrm{Y}, \{\mathrm{Z}_i\}, \{\mathrm{W}_i\}; i \in \{1, l\})$.

Sign

Le protocole de signature est un protocole interactif entre un utilisateur $\mathcal{U}$ qui prend en entrée $(m_0, \ldots, m_l)$ et un signataire $\mathcal{S}$ qui prend en entrée la clé secrète $sk = (x, y, \{z_i\})$.

1. $\mathcal{U}$ commence par calculer $\mathrm{M} = g^{m_0} \Pi_{i=1}^{l} \mathrm{Z}_i^{m_i}$, c'est-à-dire un engagement généralisé sur ses messages.
2. Il envoie ensuite au signataire une preuve de connaissance de l'ouverture de cet engagement, c'est-à-dire

$$pok\{\mu_0, \ldots, \mu_l\} : \mathrm{M} = g^{\mu_0} \Pi_{i=1}^{l} \mathrm{Z}_i^{\mu_i}.$$

3. $\mathcal{S}$ calcule alors la signature $\sigma = (a, \{\mathrm{A}_i\}, b, \{\mathrm{B}_i\}, c)$ où :
   - $\alpha \in \mathbb{Z}_p, a = g^{\alpha}$,
   - $\mathrm{A}_i = a^{z_i}$ pour $1 \leq i \leq l$,
   - $b = a^y$ et $\mathrm{B}_i = \mathrm{A}_i^y$ pour $1 \leq i \leq l$,
   - et $c = a^x \mathrm{M}^{\alpha x y}$.
4. $\mathcal{S}$ renvoie cette signature à $\mathcal{U}$.

Verify

Le destinataire valide la signature $\sigma = (a, \{\mathrm{A}_i\}, b, \{\mathrm{B}_i\}, c)$ sur les messages $(m_0, \ldots, m_1)$ en vérifiant que :

- les $\{\mathrm{A}_i\}$ sont bien formés : $e(a, \mathbb{Z}_i) = e(g, \mathrm{A}_i)$,
- b et les $\{\mathrm{B}_i\}$ sont bien formés : $e(a, \mathrm{Y}) = e(g, b)$ et $e(\mathrm{A}_i, \mathrm{Y}) = e(g, \mathrm{B}_i)$ pour tout $i \in \{1, \ldots, l\}$,
- c est bien formé : $e(\mathrm{X}, a).e(\mathrm{X}, b)^{m_0}.\Pi_{i=1}^{l} e(\mathrm{X}, \mathrm{B}_i)^{m_i} = e(g, c)$.

3.4.1.1 Preuve de connaissance d'une signature Camenisch-Lysyanskaya

Afin de prouver la connaissance de sa signature $\sigma = (a, \{\mathrm{A}_i\}, b, \{\mathrm{B}_i\}, c)$ sur le bloc de messages $(m_0, \ldots, m_l)$, le prouveur agit de la manière suivante :

1. Il calcule une version aveugle de sa signature σ, c'est-à-dire une version de sa signature telle que le vérificateur ne puisse pas la reconnaître : il choisit r et r' et calcule $\tilde{\sigma} = (\tilde{a}, \{\tilde{\mathrm{A}}_i\}, \tilde{b}, \{\tilde{\mathrm{B}}_i\}, \tilde{c}; i \in \{1, \ldots, l\})$ de la manière suivante :
   - $\tilde{a} = a^r$,
   - $\tilde{b} = b^{r'}$,
   - $\tilde{\mathrm{A}}_i = \mathrm{A}_i^r$ et $\tilde{\mathrm{B}}_i = \mathrm{B}_i^{r'}$, pour $i \in \{1, \ldots, l\}$,
   - $\hat{c} = c^{rr'}$.

   Il envoie les valeurs $\tilde{a}, \{\tilde{\mathrm{A}}_i\}, \tilde{b}, \{\tilde{\mathrm{B}}_i\}, \hat{c}$ au vérificateur.
2. On pose alors v_x, v_{xy},v_s et $\mathsf{V}_{(xy,i)}$ comme étant :

$$\mathsf{v}_x = e(\mathrm{X}, \tilde{a}), \qquad \mathsf{v}_{xy} = e(\mathrm{X}, \tilde{b}), \qquad \mathsf{v}_s = e(g, \hat{c}), \qquad \mathsf{V}_{(xy,i)} = e(\mathrm{X}, \tilde{\mathrm{B}}_i).$$

3. Le prouveur et le vérificateur calculent ces valeurs chacun de leur côté et effectuent la preuve de connaissance suivante :

$$pok\Big((\mu_0,\ldots,\mu_l),\rho : (\mathsf{v}_s)^{\rho} = \mathsf{v}_x(\mathsf{v}_{xy})^{\mu_0}\prod_{i=1}^{l}(\mathsf{V}_{(xy,i)})^{\mu_i}\Big).$$

4. $\mathcal{V}$ accepte la preuve si :
   - les valeurs $\{\tilde{\mathrm{A}}_i\}$, $i \in \{1,\ldots,l\}$ sont bien formées, c'est-à-dire $e(\tilde{a},\mathrm{Z}_i) = e(g,\tilde{\mathrm{A}}_i)$,
   - les valeurs $\tilde{b}$ et $\{\tilde{\mathrm{B}}_i\}$, $i \in \{1,\ldots,l\}$ sont bien formées, c'est-à-dire $e(\tilde{a},\mathrm{Y}) = e(g,\tilde{b})$ et $e(\tilde{a},\mathrm{Y}) = e(g,\tilde{\mathrm{B}}_i)$,
   - et la preuve ci-dessus est correcte.

Ce que nous appellerons dans ce mémoire une signature de type Camenisch-Lysyanskaya est une signature sur un engagement telle qu'il soit possible de prouver efficacement la connaissance de la signature.

3.4.1.2 Autre signature de type Camenisch-Lysyanskaya

La construction que nous venons de montrer est assez proche de celle des schémas de signature de groupe que nous verrons plus en détail au chapitre 4. De de fait, il est possible d'utiliser de tels schémas pour construire des signatures de type Camenisch-Lysyanskaya. Nous détaillons ici comment le schéma de signature de groupe ACJT [ACJT00] peut être transformé en schéma de signature de type Camenisch-Lysyanskaya dont on trouvera une description détaillée au chapitre 4.

Signature ACJT avec engagement.

Schéma de signature Setup

Les paramètres du système sont donnés par les éléments suivants :
- soit l_p un paramètre de sécurité,
- soit $n = pq$ un module RSA sûr tel que p et q soient de taille l_p,
- soient $a_0,\ldots,a_{l+1}$, $l+2$ éléments de $\mathrm{QR}(n)$.

La clé publique est donnée par $pk = (n, a_0,\ldots,a_{l+1})$ et la clé privée du signataire par $sk = (p,q)$.

Sign

Pour signer un message par blocs $m_1,\ldots,m_l$, le signataire choisit deux nombres premiers aléatoires e et s. Il calcule ensuite la valeur A telle que

$$\mathrm{A}^e = a_0\Pi_{i=1}^{l}a_i^{m_i}a_{l+1}^{s} \mod n.$$

La signature de $m = (m_1,\ldots,m_l)$ est donnée par le triplet (A,s,e).

Verify

La signature est validée en vérifiant que $\mathrm{A}^e \equiv a_0\Pi_{i=1}^{l}a_i^{m_i}a_{l+1}^{s} \mod n$.

Preuve de connaissance La preuve de connaissance d'une signature (A, e, s) sur un message $m = (m_0, \ldots, m_l)$ est notée :

$$\mathrm{PK}(\alpha_1, \ldots, \alpha_l, \beta, \gamma, \delta \backslash (\beta, \gamma, \delta) = \mathtt{Sign}(\alpha_1, \ldots, \alpha_l)).$$

Autrement dit, $\mathcal{P}$ produit la preuve suivante :

$$\mathrm{PK}(\alpha_1, \ldots, \alpha_l, \beta, \gamma, \delta \backslash \beta^\delta = a_0 \prod_{i=1}^{l} a_i^{\alpha_i} a_{l+1}^{\gamma}).$$

La preuve se déroule de la manière suivante :

1. $\mathcal{P}$ calcule d'abord un témoin de (A, e, s).
2. Il choisit $\omega \in \{0,1\}^{2l_p}$, et les valeurs aléatoires r_ω, r_e, r_s, $r_{e\omega} \in \{0,1\}^{2l_p}$.
3. Il calcule ensuite les valeurs $\mathrm{T}_1 = \mathrm{A}g^\omega$, $\mathrm{T}_2 = g^\omega h^{r_\omega}$, $\mathrm{T}_3 = g^e h^{r_e}$, $\mathrm{T}_4 = g^s h^{r_s}$,$\mathrm{T}_5 = g^{e\omega} h^{r_{e\omega}}$ et $\mathrm{T}_6 = \mathrm{T}_1^e h^r$.
4. $\mathcal{P}$ et $\mathcal{V}$ effectuent finalement la preuve de connaissance suivante :

$$\mathrm{PK}(\alpha_1, \ldots, \alpha_l, \beta, \gamma, \delta, \varepsilon, \zeta, \eta, \theta, \iota, \kappa / \mathrm{T}_6 = \mathrm{T}_1^\beta h^\kappa \wedge \mathrm{T}_3 = g^\beta h^\eta \wedge$$
$$\mathrm{T}_6 / a_0 = \textstyle\prod_{i=1}^{l} a_i^{\alpha_i} a_{l+1}^{\gamma} g^\zeta h^\kappa \wedge \mathrm{T}_2 = g^\delta h^\varepsilon \wedge \mathrm{T}_4 = g^\gamma h^\theta \wedge$$
$$\mathrm{T}_5 = g^\zeta h^\iota \wedge \mathrm{T}_5 = \mathrm{T}_2^\beta h^\mu).$$

La sécurité de ce schéma a été étudiée dans [CL04] et repose sur l'hypothèse LRSW.

3.4.2 Signature BBS

Le schéma de signature de Boneh, Boyen et Shacham [BBS04] peut aussi être utilisé pour produire une signature de type Camenisch-Lysyanskaya. Ce schéma présente un intérêt tout particulier pour nos travaux. Tout d'abord, il est possible d'adapter le schéma de signature initial pour signer des messages par blocs, tels que nous les avons décrits précédemment. Ensuite, ce nouveau schéma peut être utilisé pour construire une signature de type Camenisch-Lysyanskaya. La présentation du schéma de signature de groupe original sera faite au chapitre 4. Nous nous intéressons ici uniquement à certains schémas de signature qui peuvent en être dérivés.

Signature BBS. Setup
Soient
- $\mathbb{G}_1$ et $\mathbb{G}_2$, deux groupes cycliques d'ordre p,
- $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ un isomorphisme,
- g_2 un générateur de $\mathbb{G}_2$ et $g_1 = \psi(g_2)$ un générateur de $\mathbb{G}_1$,
- h, un générateur de $\mathbb{G}_1$,
- $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ une application bilinéaire admissible,
- $\gamma \in \mathbb{Z}_p$.

La clé secrète de signature est alors $sk = \gamma$ et la clé publique est $pk = g_2^\gamma$.

Sign
Pour obtenir une signature sur le message $m \in \mathbb{Z}_p$ de son choix, l'utilisateur procède de la façon suivante.

1. Il choisit une valeur aléatoire $x \in \mathbb{Z}_p$.
2. Il calcule la valeur A telle que : $\mathrm{A}^{\gamma+x} h^m = g_1$.

La signature est alors le couple (A, x).

Verify

Le destinataire vérifie la signature (A, x) du message m à l'aide de l'équation suivante :

$$e(\mathrm{A}, g_2)^x . e(\mathrm{A}, pk) . e(h_1, g_2)^m \stackrel{?}{=} e(g_1, g_2).$$

Signature BBS étendue. Cette signature est très facilement transposable en signature sur des messages par blocs. De plus, les techniques de Camenisch et Lysyanskaya pour construire des signatures sur des engagements sont tout à fait applicables. Nous détaillons ici, comment un utilisateur peut obtenir une signature sur un engagement d'un message par bloc $m = (m_1, \ldots, m_l)$ de taille l, de la part d'un signataire et comment il peut ensuite prouver qu'il possède une telle signature. Nous appellerons dans la suite de ce mémoire, une telle signature, une signature BBS étendue et nous la noterons BBS_l^{ext}.

Setup

Les paramètres du système sont donnés par les éléments suivants :

- $\mathbb{G}_1$ et $\mathbb{G}_2$, deux groupes cycliques d'ordre p,
- $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ un isomorphisme,
- g_2, un générateur de $\mathbb{G}_2$ et $g_1 = \psi(g_2)$ un générateur de $\mathbb{G}_1$,
- $h_1, \ldots, h_l$, des générateurs de $\mathbb{G}_1$,
- $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ une application bilinéaire admissible,
- $\gamma \in \mathbb{Z}_p$.

La clé secrète du signataire est $sk = \gamma$ et la clé publique est $pk = g_2^\gamma$.

Sign

Pour signer un bloc de message $m_1, \ldots, m_l$, le signataire choisit un entier premier aléatoire $x \in [0, p-1]$ et calcule la valeur A telle que

$$\mathrm{A} = (g_1 \prod_{i=1}^{l} h_i^{-m_i})^{\frac{1}{x+\gamma}}.$$

La signature de $m = (m_1, \ldots, m_l)$ est donnée par le couple (A, x).

Verify

La signature est validée en vérifiant que

$$e(\mathrm{A}, pk.g_2^x) \prod_{i=1}^{l} e(h_i, g_2)^{m_i} = e(g_1, g_2),$$

c'est-à-dire

$$\mathrm{A}^{x+\gamma} \prod_{i=1}^{l} h_i^{m_i} = g_1.$$

Preuve de connaissance d'une signature BBS étendue La preuve de connaissance d'une signature (A, x) sur un message $m = (m_1, \ldots, m_l)$ est notée

$$\mathrm{PK}(\alpha_1, \ldots, \alpha_l, \beta, \gamma / (\beta, \gamma) = \mathtt{Sign}(\alpha_1, \ldots, \alpha_l)).$$

Plus précisément, $\mathcal{P}$ produit la preuve

$$\mathrm{PK}(\alpha_1, \ldots, \alpha_l, \beta, \gamma / \beta^{\gamma+x} \prod_{i=1}^{l} h_i^{\alpha_i} = g_1),$$

ce qui revient à effectuer la preuve suivante

$$\mathrm{PK}(\alpha_1, \ldots, \alpha_l, \beta, \gamma / e(\beta, w g_2^{\gamma}) \prod_{i=1}^{l} e(h_i, g_2)^{\alpha_i} = e(g_1, g_2)).$$

Comme nous l'avons vu pour la signature ACJT, cette preuve est divisée en deux parties.

1. Le prouveur calcule d'abord un témoin de (A, x) :
   - il choisit $r \in [0, p-1]$ et calcule $\mathrm{T} = \mathrm{A} h_1^r$,
   - de cette équation et de la relation $\mathrm{A}^{x+\gamma} \prod_{i=1}^{l} h_i^{m_i} = g_1$ il obtient

$$e(\mathrm{T}, g_2)^x e(h_1, g_2)^{m_1 - rx} \prod_{i=2}^{l} e(h_i, g_2)^{m_i} e(h_1, w)^{-r} = e(g_1, g_2) e(\mathrm{T}, w)^{-1}.$$

2. $\mathcal{P}$ et $\mathcal{V}$ produisent finalement la preuve de connaissance suivante :

$$\mathrm{PK}(\alpha_1, \ldots, \alpha_l, \beta, \gamma /$$
$$e(\mathrm{T}, g_2)^{\beta} e(h_1, g_2)^{\alpha_1} \textstyle\prod_{i=2}^{l} e(h_i, g_2)^{\alpha_i} e(h_1, w)^{-\gamma} = e(g_1, g_2) e(\mathrm{T}, w)^{-1}.$$

La sécurité de la signature BBS étendue sera détaillée au chapitre 9.

Deuxième partie

Signatures pour l'anonymat

Chapitre 4

Signatures de groupe

Les signatures de groupe ont été introduites par Chaum et Van Heyst à *Eurocrypt'91* [CvH91]. Un schéma de signature de groupe est un procédé cryptographique permettant à un utilisateur de signer anonymement au sein d'un groupe. Ainsi le destinataire de la signature a l'assurance que celle-ci a été produite par l'un des membres du groupe mais ne sait rien de plus. Généralement, un groupe est composé de membres et d'un manager de groupe chargé de la gestion des membres. Pour pouvoir signer, le nouvel utilisateur doit d'abord s'enregistrer auprès du manager de groupe. Il obtient ainsi une clé privée de signature. En cas de litige, l'anonymat de la signature peut être levé par une autorité de confiance, en utilisant des trappes spécialement introduites dans les protocoles.

Nous présentons, dans ce chapitre, une définition formelle des signatures de groupe et détaillons deux protocoles majeurs ([ACJT00], [BBS04]) qui nous serviront dans nos future constructions.

Sommaire

4.1 Définition et état de l'art

Dans leur article [CvH91], Chaum et Van Heyst définissent intuitivement une signature de groupe comme suit. Une signature de groupe est une signature ayant les

propriétés suivantes :
- seuls les membres du groupe peuvent signer des messages ;
- le destinataire de la signature peut vérifier la validité de la signature mais ne peut pas découvrir quel membre est à l'origine de la signature ;
- en cas de contestation, la signature peut être *ouverte* (avec ou sans l'aide des membres du groupe) et révéler l'identité du signataire.

4.1.1 Usages

En introduisant les signatures de groupe, Chaum et Van Heyst [CvH91] ont proposé une application assez particulière, dans laquelle les employés d'une entreprise peuvent utiliser une imprimante de manière anonyme. En cas d'abus de l'un d'entre eux, son anonymat peut être levé. Depuis, des applications beaucoup plus générales ont été proposées, utilisant soit directement les signatures de groupe, soit des variantes. Les signatures de groupe peuvent, par exemple, être utilisées dans le cadre d'enchères électroniques [NT00]. Dans ce contexte, une autorité d'anonymat est chargée de l'enregistrement des enchérisseurs. Ceux-ci obtiennent alors un certificat qui leur permet d'enchérir anonymement et, à la fin de l'enchère, seule l'identité du meilleur enchérisseur est levée par l'autorité.

Le vote électronique a aussi été proposé comme application des signatures de groupe. Cependant par sa construction même, cette signature n'est pas souhaitable pour le vote. En effet l'existence d'une autorité capable de lever l'anonymat de n'importe quel votant va à l'encontre même de l'anonymat des bulletins. Pour parer à ces défauts, les signatures de liste qui sont une variante des signatures de groupe, ont été utilisées pour construire des schémas de vote électronique [CT03a].

4.1.2 Sécurité

Afin d'être prouvé sûr, un schéma de signature de groupe doit respecter certaines propriétés. Nous donnons ici une définition informelle des principales propriétés. Nous verrons dans la section 4.2 comment certains auteurs ont réussi à diminuer le nombre de propriétés en affinant les définitions.

Pour être déclaré sûr, un schéma de signature de groupe doit vérifier les propriétés suivantes :
- l'*inforgeabilité* : seuls les membres du groupe doivent pouvoir signer.
- l'*anonymat* : seule l'autorité de révocation est capable de retrouver l'identité d'un utilisateur à partir d'un couple message/signature valide.
- la *non-reliabilité* : décider si deux signatures proviennent du même signataire ou non est calculatoirement difficile.
- la *non-diffamation* : ni les membres du groupe, ni le manager de groupe ne peuvent signer au nom d'autres membres.
- la *traçabilité* : l'autorité de révocation doit toujours être capable de lever l'anonymat à partir d'une signature valide.
- la *résistance aux coalitions* : aucune coalition de membres ne peut produire de signature qui ne soit pas ouvrable sur l'identité d'un des membres de la coalition.

4.1.3 Construction

Un schéma de signature de groupe est donc formé de trois types de participants :
- un *manager de groupe*, dont le rôle est d'enregistrer les nouveaux membres du groupe,
- les *membres du groupe* qui s'enregistrent auprès du manager du groupe et qui peuvent ensuite signer anonymement,
- une *autorité de révocation* qui est capable de lever l'anonymat sur les signatures. Dans certains schémas, le rôle d'autorité est donné au manager de groupe.

On considère alors deux sortes de groupes :
- les groupes statiques, dans lesquels la taille du groupe est fixée au moment de la mise en place du groupe et où aucun nouveau membre ne peut être ajouté ou retiré par la suite.
- les groupes dynamiques, dans lesquels, un membre peut adhérer au groupe ou le quitter à n'importe quel moment.

Il est alors possible de définir un schéma de signature de groupe à l'aide des algorithmes suivants :

Définition 80 (Construction d'un schéma de signature de groupe)**.** *Un schéma de signature de groupe est un schéma de signature défini par les cinq procédures suivantes :*
- Setup*, algorithme probabiliste de génération des paramètres du système et publics. Il prend en entrée un paramètre de sécurité k et retourne la clé publique du groupe pk_{G}, la clé secrète du manager de groupe $sk_{\mathcal{MG}}$, la clé secrète de l'autorité de révocation sk_{RA} et si besoin est, une clé secrète $sk_{\mathcal{U}}$ pour chaque utilisateur.*
- Join*, protocole interactif d'enregistrement entre un utilisateur et le manager de groupe. En fin de procédure, le nouveau membre est en possession de son certificat d'appartenance au groupe et sa clé secrète de signature $sk_{\mathcal{U}}$.*
- Sign*, algorithme probabiliste de signature. Il prend en entrée un message m, une clé privée d'utilisateur $sk_{\mathcal{U}}$ et un certificat, et retourne une signature σ du message.*
- Verify*, algorithme de vérification de signature. Il prend en entrée un message m, une signature σ et la clé publique du groupe pk_{G}. Il retourne 1 si la signature est valide et 0 sinon.*
- Open*, algorithme d'ouverture de signature. Il prend en entrée un message m, une signature valide σ de ce message, la clé publique du groupe pk_{G} et la clé secrète de l'autorité sk_{RA} et retourne l'identité $\mathcal{U}$ du signataire.*

Cette définition présente les cinq algorithmes nécessaires pour construire un schéma de signature de groupe. Certains auteurs ont proposé des schémas ne comprenant pas de phase de Join, et, comme nous le verrons par la suite, les protocoles résultants ne permettent pas de garantir la meilleure sécurité possible pour un schéma de signature de groupe.

4.1.4 État de l'art

Dans leur article [CvH91], Chaum et Van Heyst ont proposé les premiers schémas de signature de groupe. Malheureusement, les solutions proposées ne sont pas efficaces et l'ouverture des signatures est effectuée par les membres eux-mêmes, ce qui peut poser des problèmes de confiance. Par la suite, Chen et Pedersen [CP94] ont proposé deux schémas plus intéressants. En particulier, l'ouverture est faite par une autorité de révocation indépendante du groupe. La première construction générique de signature de groupe à partir de n'importe quel schéma de signature a été proposée en 1997 par Petersen [Pet97], mais celle-ci s'avère peu efficace en pratique.

Le premier schéma pratique (non publié) a été proposé par Traoré en 1995. Dans ce schéma, les tailles des clés et des signatures sont indépendantes du nombre de membres dans le groupe. Le même résultat a été obtenu en 1997 par Camenisch et Stadler [CS97]. La sécurité de ces deux schémas repose sur la difficulté de trouver des racines d'un polynôme choisi judicieusement. Mais aucun d'entre eux ne garantit la sécurité dans le cas de coalitions de membres.

Pour résoudre ce problème, Camenisch et Michels ont proposé l'année suivante, un nouveau schéma [CM98] résistant aux attaques par coalition, dans le cas d'attaques non-adaptatives. La sécurité de ce schéma repose, sur le problème RSA fort et le problème Diffie-Hellman décisionnel et la sécurité est montrée dans le modèle de l'oracle aléatoire.

Le premier schéma efficace résistant aux coalitions face à des attaques adaptatives est dû à Ateniese, Camenisch, Stadler et Tsudik [ACJT00]. Ce schéma est une modification de [CM98] et repose comme celui-ci sur les hypothèses RSA fort et Diffie-Hellman décisionnel.

Par la suite, le travail sur les signatures de groupe a principalement consisté à réduire la taille des signatures et augmenter l'efficacité et la sécurité des schémas. Tout d'abord, en 2003, Bellare, Micciancio et Warinschi [BMW03] ont posé un nouveau formalisme pour les signatures de groupe dans le cas statique. Bellare *et al.* ont aussi proposé aussi un schéma générique de signature de groupe. Nous donnons, dans la section 4.2, une description plus détaillée de ce modèle. Celui-ci a été utilisé par Boneh, Boyen et Shacham pour prouver la sécurité de leur schéma [BBS04]. Cette construction fournit une signature courte (de la taille d'une signature RSA) avec un niveau de sécurité élevé. Le schéma utilise des applications bilinéaires et sa sécurité est basée sur l'hypothèse RSA fort ainsi qu'une nouvelle hypothèse, appelée l'hypothèse linéaire décisionnelle. Le point faible de ce schéma réside dans la mise en place des clés des utilisateurs. En effet, le manager de groupe distribue les clés aux nouveaux membres et est donc en possession de tous les secrets des utilisateurs. Boneh *et al.* proposent à la fin de leur article une construction pour éviter ce problème, mais ceci rallonge la taille de leur signature. Nous donnons une description plus complète de ce schéma dans la section 4.3. La même année, Boneh et Shacham [BS04] ont proposé un schéma améliorant la révocation de la construction précédente. En particulier, lorsqu'un membre est exclu du groupe, les membres restants n'ont pas de mise à jour à faire.

Parallèlement à ces constructions basées sur le problème RSA fort, Camenisch et Lysyanskaya ont présenté un schéma [CL04] dont la sécurité est prouvée sous l'hypothèse LRSW [LRSW99] dans le modèle de l'oracle aléatoire.

A la suite du modèle BWM, Bellare, Shi et Zhang [BSZ05] ont défini un modèle de sécurité pour les groupes dynamiques et présenté une construction générique de schéma de signature de groupe. Ce modèle est détaillé dans la section 4.2. De même, il a été utilisé pour prouver la sécurité de nombreux schémas de signature de groupe ([NSN04], [DP06]).

Plus récemment, Boyen et Waters [BW06] ont proposé un schéma prouvé sûr dans le modèle de [BMW03] , mais sans utiliser d'oracle aléatoire. La taille de leur signature est logarithmique en le nombre de membres du groupe et la sécurité de leur schéma repose sur le problème calculatoire Diffie-Hellman et l'hypothèse décisionnelle d'indistinguabilité de distribution de sous-groupes, hypothèse qu'ils introduisent dans l'article. À *PKC'07*, ces mêmes auteurs ont présenté une nouvelle construction telle que la taille de la signature soit constante et dont la sécurité est prouvée dans le modèle standard en utilisant les définitions de [BMW03].

Il existe en parallèle des schémas de signature de groupe, de nombreuses constructions qui s'inspirent des propriétés de ces signatures. Par exemple, les schémas de traçage de traître (*Traitor Tracing*) [KY03], d'usurpation d'identité (*Identity Escrow*) [CL01], les signatures de liste, les signatures traçables, les signature en anneau (cette notion sera vue plus en détail dans le chapitre 6)... Certains de ces schémas ont d'abord été construits en signature de groupe, puis modifiés, ou inversement, ont servi de base pour la construction de signatures de groupe.

4.2 Sécurité des signatures de groupe

Dans cette section, nous évoquons les modèles de sécurité définis pour les signatures de groupe sans néanmoins entrer dans les détails. Dans l'article de Chaum *et al.* [CvH91], les schémas sont présentés sans preuve formelle de sécurité. Par la suite, les définitions sont restées plutôt informelles ([CS97], [CM98], [ACJT00], [Tra99]) et les modèles présentés assez faibles. En 2001, Camenisch et Lysyanskaya ont défini la sécurité des signatures de groupe dans le modèle de la composabilité universelle.

Plus récemment, de nouveaux modèles définissant les buts de l'adversaire et les moyens mis à sa disposition ont été présentés. En 2003, Kiayias et Yung [KY03] ont présenté le premier modèle formel pour les signatures de groupe dans le cas d'une *attaque à message choisis avec capacité de sélection de membres* : l'adversaire est capable d'obtenir des signatures sur des messages de son choix de la part de membres de son choix. La même année, Bellare, Micciancio et Warinschi [BMW03] ont proposé un modèle encore plus fort dans lequel ils introduisent notamment un nouveau participant, le Juge, dont le rôle est de vérifier les actions de l'autorité de révocation. Ce modèle se restreint aux schémas de signature de groupe statique et par la suite, Bellare, Shi et Zhang ont étendu ce modèle aux groupes dynamiques [BSZ05]. Dans la suite de cette partie, nous présentons ces deux modèles, que nous nommons BMW et BSZ, d'après leurs auteurs.

4.2.1 Modèle BMW

Le modèle de sécurité BMW s'applique aux schémas de signature de groupe statiques. Bellare *et al.* commencent tout d'abord par redéfinir les algorithmes en s'adaptant au cas des groupes statiques.

Dans leur modèle, l'algorithme Join est inclus dans la phase de Setup, c'est-à-dire que l'utilisateur reçoit sa clé privée de signature $sk_{\mathcal{U}_i}$ en même temps que le manager et l'autorité de révocation. Les algorithmes Sign, Verify et Open ne sont pas changés.

L'intérêt principal de ce modèle est de réunir toutes les propriétés nécessaires à la sécurité d'un schéma de signature de groupe en seulement trois notions que nous décrivons maintenant.

Consistance : cette propriété garantit, tout d'abord, toute signature émise de manière honnête doit toujours être vérifiable. De plus, l'algorithme d'ouverture doit toujours être capable de retrouver l'identité du signataire à partir d'une signature valide.

Anonymat total : cette propriété garantit qu'un adversaire ne connaissant pas la clé secrète du manager ne peut retrouver les identités des signataires à partir de leurs signatures. Cette notion englobe à la fois les propriétés d'anonymat et de non-reliabilité. Afin de se prémunir contre les coalitions d'utilisateur, l'adversaire peut avoir accès aux clés secrètes des membres du groupe. Malgré son appellation, cette notion d'anonymat est différente de l'anonymat inconditionnel qui précise que l'adversaire ne peut distinguer, même avec une puissance de calcul infini, la provenance d'une signature.

Traçabilité totale : cette propriété garantit qu'aucune coalition d'utilisateurs ne peut créer de signature qui ne puisse être ouverte ou de signature qui ne soit pas reliée à un membre de la coalition. La notion de traçabilité totale est plus forte que la notion de traçabilité telle qu'elle était définie auparavant. Dans leur définition, Bellare, Micciancio et Warinschi englobent les notions de traçabilité, d'inforgeabilité, de résistance aux coalitions et de non-diffamation.

4.2.2 Modèle BSZ

Ce modèle de sécurité fait suite au modèle BMW. Dans cette construction, Bellare, Shi et Zhang définissent un modèle de sécurité pour les groupes dynamiques, en redéfinissant les propriétés données dans BMW.

Dans ce modèle, un membre peut adhérer au groupe et obtenir un *certificat de membre* à n'importe quel moment, en engageant un protocole Join avec le manager de groupe. De plus, afin de prouver son honnêteté, l'autorité de révocation doit fournir une preuve qu'elle a correctement exécuté l'algorithme Open. Cette preuve est ensuite fournie à un Juge, qui grâce à un algorithme Juge est capable de confirmer ou non la bonne foi de l'autorité de révocation. La description d'un schéma de signature de groupe comprend donc un participant et un algorithme de plus que dans la définition 80.

Ce modèle regroupe les propriétés de sécurité d'un schéma de signature de groupe dynamique en quatre propriétés : la consistance (*correctness*, l'anonymat (*anonymity*), la traçabilité (*traceability*) et la non-diffamation (*non-frameability*).

Consistance : si une signature sur un message a été émise de manière honnête par un membre honnête alors elle est valide et, étant donné un couple message/signature, l'algorithme de traçabilité doit retrouver l'identité du membre. De plus, une preuve émise par l'autorité de révocation doit être acceptée par le Juge.

Anonymat : il ne doit pas être possible de retirer d'information significative d'un couple message/signature. Le but de l'adversaire est plus faible dans ce modèle que dans le modèle BMW. Pour casser l'anonymat d'un schéma, l'adversaire ne doit plus retrouver l'identité d'un membre à partir d'une signature mais distinguer quel utilisateur, parmi deux de son choix, a signé un message qu'il a fourni.

Traçabilité : cette propriété garantit que toute signature doit être révocable. Un adversaire contre la traçabilité d'un schéma ne doit pas pouvoir produire une signature de groupe valide telle que l'autorité de révocation (qui est honnête lors de cette attaque) soit incapable d'en identifier l'origine.

Non-diffamation : cette propriété garantit qu'un utilisateur ne peut pas se voir attribuer des signatures à la production desquelles il n'aurait pas coopéré. Autrement dit, un adversaire ne doit pas être capable de créer une preuve acceptée par le juge qu'un utilisateur honnête a produit une signature valide sans que cet utilisateur ne soit à l'origine de cette signature. Ceci implique la définition plus classique de non-diffamation, à savoir qu'il ne doit pas être possible de produire une signature valide telle qu'une autorité honnête soit capable de l'ouvrir et produise une preuve valide que cette signature provient d'un autre utilisateur (honnête).

4.3 Schémas

Dans cette partie, nous décrivons deux schémas de signature de groupe, le schéma ACJT[ACJT00] et le schéma BBS[BBS04], nommés ainsi d'après le nom de leurs auteurs. Ces schémas sont tous les deux prouvés sûrs et seront utilisés par la suite dans nos constructions.

4.3.1 ACJT

Dans leur article *A Practical and Provably Secure Coalition-Resistant Group signature Scheme*, Ateniese, Camenisch, Joye et Tsudik présentent un schéma efficace de signature de groupe, dont la sécurité (notamment la résistance aux attaques par coalition) est basée sur l'hypothèse RSA fort et l'hypothèse Diffie-Hellman décisionnelle.

4.3.1.1 Création des paramètres : Setup

Les paramètres du système sont donnés par :
- $\varepsilon > 1$, k et l_p les paramètres de sécurité,
- $\lambda_1, \lambda_2, \gamma_1$ et γ_2 des longueurs telles que $\lambda_1 > \varepsilon(\lambda_2+k)+2$, $\lambda_2 > 4l_p$, $\gamma_1 > \varepsilon(\gamma_2+k)+2$ et $\gamma_2 > \lambda_1 + 2$,
- les intervalles $\Lambda =]2^{\lambda_1} - 2^{\lambda_2}, 2^{\lambda_1} + 2^{\lambda_2}[$ et $\Gamma =]2^{\gamma_1} - 2^{\gamma_2}, 2^{\gamma_1} + 2^{\gamma_2}[$,
- $\mathcal{H}$ une fonction de hachage résistante aux collisions $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^k$.

Dans la première phase du protocole, le manager de groupe (qu'on note $\mathcal{MG}$) établit les paramètres publics et calcule sa clé privée.

1. $\mathcal{MG}$ choisit des premiers secrets p' et q', de taille l_p, tels que $p = 2p' + 1$ et $q = 2q' + 1$ soient premiers. Il calcule le module $n = pq$.
2. $\mathcal{MG}$ choisit les éléments aléatoires a, a_0, g, $h \in_{\mathrm{R}} \mathrm{QR}(n)$.
3. $\mathcal{MG}$ choisit un secret aléatoire $x \in_{\mathrm{R}} \mathbb{Z}^*_{p'q'}$ et pose $y = g^x \bmod n$.

La clé publique du groupe est $pk_{\mathrm{G}} = (n, a, a_0, y, g, h)$ et la clé secrète de $\mathcal{MG}$ est $sk_{\mathrm{MG}} = (p', q', x)$.

4.3.1.2 Enregistrement d'un utilisateur : Join

Afin de faire partie d'un groupe, tout nouvel utilisateur doit s'enregistrer auprès du manager du groupe, à l'aide du protocole Join. À l'issue de l'exécution de ce protocole, il a obtenu un certificat d'appartenance au groupe, ainsi qu'un secret connu de lui seul, qui lui permettra de signer anonymement.

Les échanges entre l'utilisateur $\mathcal{U}$ et le manager de groupe $\mathcal{MG}$ sont décrits dans la figure 4.1.

1. Tout d'abord $\mathcal{U}$ choisit une valeur secrète $\tilde{x}_i$ et envoie à $\mathcal{MG}$ un engagement sur x_i en prouvant que celui-ci est bien construit dans la preuve pok_1.
2. $\mathcal{MG}$ renvoie deux aléas à $\mathcal{U}$ et celui-ci construit son secret x_i et envoie à $\mathcal{MG}$ la valeur $\mathrm{C}_2 = a^{x_i} \bmod n$ avec une preuve de consistance pok_2.
3. Ce dernier peut maintenant calculer un signature de cet engagement et renvoie à $\mathcal{U}$ son certificat.

Lors de ces échanges, $\mathcal{U}$ produit deux preuves de connaissance :

1. dans pok_1, $\mathcal{U}$ prouve la connaissance de la représentation de C_1 en bases g et h. $pok_1 = pok(\mu, \nu : \mathrm{C}_1 = g^{\mu} h^{\nu})$.
2. dans pok_2, $\mathcal{U}$ prouve à GM que :
   - le logarithme discret de C_2 en base a est dans l'intervalle Λ,
   - il connaît les entiers u, v et w tels que :
     - u appartient à l'intervalle $]-2^{\lambda_2}, 2^{\lambda_2}[$,
     - u est égal au logarithme discret de $\mathrm{C}_2/a^{2^{\lambda_1}}$ en base a,
     - $\mathrm{C}_1^{\alpha_i} g^{\beta_i}$ est égal à $g^u (g^{2^{\lambda_2}})^v h^w$.

   $pok_2 = pok(u, v, w : u \in]-2^{\lambda_2}, 2^{\lambda_2}[\wedge a^u = \mathrm{C}_2/a^{2^{\lambda_1}} \wedge \mathrm{C}_1^{\alpha_i} g^{\beta_i} = g^u (g^{2^{\lambda_2}})^v h^w)$.

L'utilisateur est maintenant enregistré. Son certificat est le couple $[\mathrm{A}_i, e_i]$ et le manager ajoute le triple $\{\mathrm{A}_i, e_i, \mathrm{trans}_i\}$ dans une liste de membres enregistrés.

4.3.1.3 Signature d'un message : Sign

Muni de son certificat, le membre $\mathcal{U}$ peut maintenant signer n'importe quel message de son choix $m \in \{0,1\}^*$. Pour ce faire $\mathcal{U}$ exécute le protocole suivant :

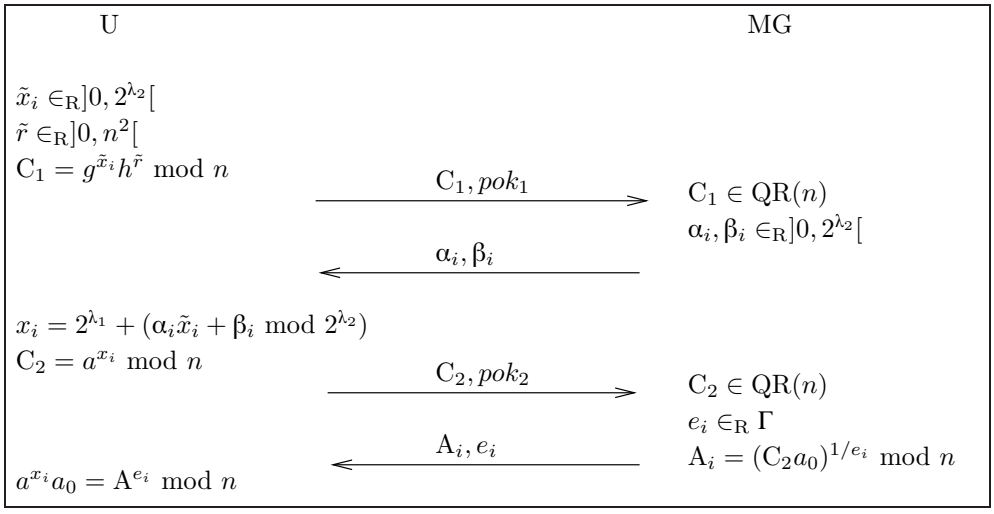

FIG. 4.1 – Protocole Join du schéma ACJT

1. $\mathcal{U}$ génère une valeur aléatoire $w \in_R \{0,1\}^{2l_p}$ et calcule

$$T_1 = A_i y^w \bmod n, \qquad T_2 = g^w \bmod n, \qquad T_3 = g^{e_i} h^w \bmod n.$$

2. $\mathcal{U}$ choisit $r_1 \in_R \pm\{0,1\}^{\varepsilon(\gamma_2+k)}$, $r_2 \in_R \pm\{0,1\}^{\varepsilon(\lambda_2+k)}$, $r_3 \in_R \pm\{0,1\}^{\varepsilon(\gamma_1+2l_p+k+1)}$ et $r_4 \in_R \pm\{0,1\}^{\varepsilon(2l_p+k)}$ et calcule :
   (a) $d_1 = T_1^{r_1}/(a^{r_2} y^{r_3}) \bmod n$, $d_2 = T_2^{r_1}/g^{r_3} \bmod n$, $d_3 = g^{r_4} \bmod n$ et $d_4 = g^{r_1} h^{r_4} \bmod n$;
   (b) $c = \mathcal{H}(g||h||y||a_0||a||T_1||T_2||T_3||d_1||d_2||d_3||d_4||m)$;
   (c) $s_1 = r_1 - c(e_i - 2^{\gamma_1})$, $s_2 = r_2 - c(x_i - 2^{\lambda_1})$, $s_3 = r_3 - c e_i w$ et $s_4 = r_4 - cw$.
3. $\mathcal{U}$ renvoie $(c, s_1, s_2, s_3, s_4, T_1, T_2, T_3)$.

La signature de connaissance que produit l'utilisateur peut être vue comme une signature de connaissance de son certificat.

4.3.1.4 Vérification d'une signature : Verify

Le destinataire vérifie la validité de la signature $(c, s_1, s_2, s_3, s_4, T_1, T_2, T_3)$ du message m de la manière suivante :

1. il calcule

$$c' = \mathcal{H}(g||h||y||a_0||a||T_1||T_2||T_3||a_0^c T_1^{s_1 - c2^{\gamma_1}}/(a^{s_2 - c2^{\lambda_1}} y^{s_3}) \bmod n \\ ||T_2^{s_1 - c2^{\gamma_1}}/g^{s_3} \bmod n||T_2^c g^{s_4} \bmod n||T_3^c g^{s_1 - c2^{\gamma_1}} h^{s_4} \bmod n||m)$$

2. la signature est acceptée si et seulement si $c = c'$ et si $s_1 \in_R \pm\{0,1\}^{\varepsilon(\gamma_2+k)+1}$, $s_2 \in_R \pm\{0,1\}^{\varepsilon(\lambda_2+k)+1}$, $s_3 \in_R \pm\{0,1\}^{\varepsilon(\gamma_1+2l_p+k+1)+1}$ et $s_4 \in_R \pm\{0,1\}^{\varepsilon(2l_p+k)+1}$

4.3.1.5 Révocation d'anonymat : Open

En cas de conflit, le manager de groupe est capable de lever l'anonymat d'une signature.

1. $\mathcal{MG}$ vérifie la validité de la signature à l'aide du protocole Verify.
2. Il retrouve A_i en calculant $\mathrm{A}_i = \mathrm{T}_1/\mathrm{T}_2^x \bmod n$.
3. Il prouve que $\log_g y = \log_{\mathrm{T}_2}(\mathrm{T}_1/\mathrm{A}_i \bmod n)$.

4.3.1.6 Sécurité

La sécurité du schéma repose sur le modèle de sécurité présenté dans [ACJT00] et est basée sur les deux théorèmes suivants.

Théorème 10 (Résistance aux coalitions - [ACJT00])**.** *Sous l'hypothèse RSA fort, un certificat de groupe* $[\mathrm{A}_i, e_i]$ *ne peut être généré que par le manager de groupe, tant que le nombre de certificats émis par le manager est polynomialement borné.*

Théorème 11 (Preuves de connaissance à divulgation nulle de connaissance - [ACJT00])**.** *Sous l'hypothèse RSA fort, le protocole interactif sous-jacent au schéma de signature de groupe est une preuve de connaissance à divulgation statistiquement nulle de connaissance d'un certificat et une preuve de connaissance à divulgation nulle de connaissance d'une clé secrète de membre.*

Nous renvoyons le lecteur à [ACJT00] pour le détail des preuves de sécurité.

4.3.1.7 Amélioration du schéma

Dans ce schéma, le manager de groupe joue les deux rôles de manager de membres et de manager de révocation. Ces deux parties peuvent être divisée en deux entités de manière directe pour renforcer la sécurité du schéma.

4.3.2 BBS

Dans leur article *Short Group Signatures* [BBS04], Boneh, Boyen and Shacham s'attachent à construire des signatures de groupe courtes : leur signature est approximativement de la même taille qu'une signature RSA pour une sécurité équivalente. La sécurité de leur schéma repose sur l'hypothèse Diffie-Hellman fort et l'hypothèse linéaire décisionnelle (*Decision Linear assumption*) qu'ils introduisent dans l'article. La sécurité de leur schéma est montrée dans le modèle BMW.

L'agencement de ce schéma est un peu différent du modèle précédent. En effet, le manager de groupe ici est considéré comme l'autorité de révocation et n'a pas pour rôle d'enregistrer les nouveaux arrivants. En effet, lorsqu'une personne désire entrer dans le groupe, elle va s'identifier auprès du manager qui va lui donner son certificat de membre de groupe, par conséquent l'autorité connaît tous les certificats des membres du groupe. De plus, c'est elle aussi qui choisit la clé secrète associée à la clé publique du groupe et elle reste la seule à en avoir connaissance. Il n'y a donc pas à proprement parler de protocole Join dans la version simple de ce schéma.

4.3.2.1 Création des paramètres : Setup

Les paramètres du système sont donnés par :
- $\mathbb{G}_1$ et $\mathbb{G}_2$ deux groupes cycliques d'ordre premier p,
- ψ un isomorphisme calculable de $\mathbb{G}_2$ dans $\mathbb{G}_1$,
- e un fonction bilinéaire admissible de G_1 dans G_2,
- une fonction de hachage $\mathcal{H} : \{0,1\} \to \mathbb{Z}_p$.

Dans la suite, on considère que le nombre de membres dans le groupe est fixé et on note n ce nombre.

Les clés sont construites en suivant la procédure suivante :

1. choisir aléatoirement selon une distribution uniforme, un générateur g_2 dans G_2 et poser $g_1 = \psi(g_2)$;
2. choisir $h \xleftarrow{\mathrm{R}} \mathbb{G}_1 \setminus \{1_{\mathbb{G}}\}$ et ξ_1, $\xi_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ et u et $v \in \mathbb{G}_1$, tels que $u^{\xi_1} = v^{\xi_2} = h$. Les valeurs u et v sont en fait des clés publiques pour un double chiffrement ElGamal (*cf.* section 3.2). Les clés privées sont alors données par ξ_1 et ξ_2 ;
3. choisir $\gamma \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ et poser $w = g_2^{\gamma}$;
4. pour chaque utilisateur i, $1 \leq i \leq n$ générer un couple (A_i, x_i) en choisissant $x_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ et en posant $\mathrm{A}_i \leftarrow g_1^{1/(\gamma + x_i)}$. Les valeurs (A_i, x_i) ainsi que l'identité de l'utilisateur sont enregistrées dans une liste.

La clé publique du groupe est $pk_{\mathrm{G}} = (g_1, g_2, h, u, v, w)$. La clé privée du manager de groupe est $sk_{\mathcal{MG}} = (\xi_1, \xi_2)$. Chaque utilisateur reçoit son certificat de membre du groupe $sk_{\mathcal{U}} = (\mathrm{A}_i, x_i)$.

4.3.2.2 Signature d'un message : Sign

Afin de signer un message $m \in \{0,1\}^*$ de son choix, l'utilisateur a besoin de calculer une signature de connaissance afin de prouver la connaissance de son certificat sans le révéler. Nous décrivons d'abord la preuve interactive entre l'utilisateur $\mathcal{U}$ et le destinataire (qui joue ici le rôle de vérificateur), puis nous la transformons en signature de connaissance.

1. $\mathcal{U}$ choisit des exposants $\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ et calcule un chiffrement linéaire de A :

$$\mathrm{T}_1 \leftarrow u^{\alpha} \qquad \mathrm{T}_2 \leftarrow v^{\beta} \qquad \mathrm{T}_3 = \mathrm{A}h^{\alpha+\beta}.$$

   Il calcule aussi deux valeurs $\delta_1 \leftarrow x\alpha$ et $\delta_2 \leftarrow x\beta \in \mathbb{Z}_p^*$.

2. Le but de $\mathcal{U}$ est alors de prouver au destinataire qu'il connaît les valeurs $(\alpha, \beta, x, \delta_1, \delta_2)$ qui satisfont les cinq relations

$$\begin{aligned} u^{\alpha} &= \mathrm{T}_1 & v^{\beta} &= \mathrm{T}_2 \\ e(\mathrm{T}_3, g_2)^x . e(h,w)^{-\alpha-\beta} . e(h,g_2)^{-\delta_1-\delta_2} &= e(g_1,g_2)/e(\mathrm{T}_3,w) \\ \mathrm{T}_1^x u^{-\delta_1} &= 1 & \mathrm{T}_2^x v^{-\delta_2} &= 1 \end{aligned}$$

   Il procède de la façon suivante :

- $\mathcal{U}$ choisit aléatoirement des valeurs $r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2}$ dans $\mathbb{Z}_p^*$. Il calcule ensuite les valeurs suivantes :

$$\begin{aligned} \mathrm{R}_1 &\leftarrow u^{r_\alpha}, & \mathrm{R}_2 &\leftarrow v^{r_\beta}, \\ \mathrm{R}_3 &\leftarrow e(\mathrm{T}_3, g_2)^{r_x}.e(h,w)^{-r_\alpha - r_\beta}.e(h,g_2)^{-r_{\delta_1}-\delta_2}, \\ \mathrm{R}_4 &\leftarrow \mathrm{T}_1^{r_x}.u^{-r_{\delta_1}}, & \mathrm{R}_5 &\leftarrow \mathrm{T}_2^{r_x}.v^{-r_{\delta_2}}. \end{aligned}$$

- $\mathcal{U}$ envoie les valeurs $(\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3, \mathrm{R}_4, \mathrm{R}_5)$ au vérificateur.
- Le destinataire (qui est le vérificateur dans ce schéma) envoie un challenge c, qui est une valeur choisie aléatoirement dans la distribution uniforme sur $\mathbb{Z}_p$.
- $\mathcal{U}$ calcule les valeurs suivantes :

$$s_\alpha \leftarrow r_\alpha + c\alpha, \quad s_\beta \leftarrow r_\beta + c\beta, \quad s_x \leftarrow r_x + cx, \quad s_{\delta_1} \leftarrow r_{\delta_1} + c\delta_1, \quad s_{\delta_2} \leftarrow r_{\delta_2} + c\delta_2.$$

et les envoie au vérificateur.

- Le destinataire vérifie les cinq équations suivantes :

$$u^{s_\alpha} \stackrel{?}{=} \mathrm{T}_1^c.\mathrm{R}_1, \tag{4.1}$$

$$v^{s_\beta} \stackrel{?}{=} \mathrm{T}_2^c.\mathrm{R}_2, \tag{4.2}$$

$$e(\mathrm{T}_3, g_2)^{s_x}.e(h,w)^{-s_\alpha - s_\beta}.e(h,g_2)^{-s_{\delta_1}-s_{\delta_2}} \stackrel{?}{=} \left(\frac{e(g_1,g_2)}{e(\mathrm{T}_3,w)}\right)^c.r_3, \tag{4.3}$$

$$\mathrm{T}_1^{s_x}.u^{-s_{\delta_1}} \stackrel{?}{=} \mathrm{R}_4, \tag{4.4}$$

$$\mathrm{T}_2^{s_x}.v^{-s_{\delta_1}} \stackrel{?}{=} \mathrm{R}_5. \tag{4.5}$$

3. Afin d'obtenir une signature du message m, la preuve décrite précédemment est convertie en signature de connaissance :
   - $\mathcal{U}$ calcule les valeurs $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3, \mathrm{R}_4, \mathrm{R}_5$
   - Il calcule le challenge c à l'aide de la fonction de hachage :

$$c \leftarrow \mathcal{H}(m, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3, \mathrm{R}_4, \mathrm{R}_5) \in \mathbb{Z}_p$$

   - A l'aide de c, il calcule les valeurs $s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}$
4. La signature est alors donnée par $\sigma \leftarrow (\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$.

4.3.2.3 Vérification d'une signature : Verify

Étant donné un message m, une signature σ et la clé publique du groupe pk_G, la vérification s'effectue de la manière suivante :

1. Le destinataire utilise les équations 4.1 à 4.5 pour retrouver R_1, R_2, R_3, R_4, R_5 :

$$\tilde{\mathrm{R}}_1 \leftarrow u^{s_\alpha}.\mathrm{T}_1^{-c} \qquad \tilde{\mathrm{R}}_2 \leftarrow .\mathrm{T}_2^{-c} \qquad \tilde{\mathrm{R}}_4 \leftarrow \mathrm{T}_1^{s_x}.u^{-s_{\delta_1}} \qquad \tilde{\mathrm{R}}_5 \leftarrow \mathrm{T}_2^{s_x}.v^{-s\delta_2} \tag{4.6}$$

$$\tilde{\mathrm{R}_3} \leftarrow e(\mathrm{T}_3, g_2)^{s_x}.e(h,w)^{-s_\alpha - s_\beta}.e(hng_2)^{-s_{\delta_1}-s_{\delta_2}}.\big(e(\mathrm{T}_3,w)/e(g_1,g_2)\big)^c \tag{4.7}$$

2. Il vérifie ensuite qu'il retrouve bien le challenge c :

$$c \stackrel{?}{=} \mathcal{H}(m, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \tilde{\mathrm{R}}_1, \tilde{\mathrm{R}}_2, \tilde{\mathrm{R}}_3, \tilde{\mathrm{R}}_4, \tilde{\mathrm{R}}_5)$$

3. Il accepte la signature si la vérification est correcte et la rejette autrement.

4.3.2.4 Révocation d'anonymat : Open

Le manager de groupe qui possède les clés privées de déchiffrement (ξ_1, ξ_2) est le seul à pouvoir ouvrir les signatures.

1. $\mathcal{MG}$ reçoit une signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ et un message m.
2. Il vérifie la validité de la signature.
3. Si celle-ci est valide, il retrouve la valeur A à l'aide du chiffrement linéaire contenu dans les valeurs T_1, T_2, T_3 :

$$A \leftarrow T_3/(T_1^{\xi_1}.T_2^{\xi_2}).$$

4. Il regarde alors dans sa liste à quelle identité cette valeur A correspond.

4.3.2.5 Sécurité

La sécurité du schéma est montrée en utilisant le modèle BMW et est donnée par les théorèmes suivants

Théorème 12 (Consistance - [BBS04])**.** *Le schéma de signature de groupe BBS est consistant.*

Théorème 13 (Anonymat - [BBS04])**.** *Si le chiffrement linéaire est (t', ε')-sémantiquement sûr sur $\mathbb{G}_1$, alors le schéma de signature de groupe est $(t, q_\mathcal{H}, \varepsilon)$-CPA-anonyme, avec $\varepsilon = \varepsilon'$ et $t = t' - q_\mathcal{H}O(1)$ où $q_\mathcal{H}$ est le nombre de requêtes faites à la fonction de hachage par l'adversaire.*

Théorème 14 (Traçabilité - [BBS04])**.** *Si le problème Diffie-Hellman fort est (q, t', ε')-sur alors le schéma de signature de groupe est (q, t', ε') entièrement traçable, avec $n = q - 1, \varepsilon = 4n\sqrt{2\varepsilon' q_\mathcal{H}} + n/p$ et $t = O(1).t'$ où $q_\mathcal{H}$ est le nombre de requêtes à la fonction de hachage lancées par l'adversaire, q_s est le nombre de requêtes de signatures, et n, le nombre de membres dans le groupe.*

Nous renvoyons le lecteur à [BBS04] pour la description des preuves.

4.3.2.6 Amélioration du schéma

Dans cette version du schéma, les clés sont distribuées par une autorité lors de la phase de mise en place des paramètres. En conséquence l'utilisateur doit faire confiance à cette autorité car celle-ci à tout pouvoir de créer une signature en son nom. Afin d'empêcher un tel cas de figure, Boneh *et al.* décrivent une procédure de Join qui permet à l'utilisateur de participer à la création de son certificat. A la fin de ce protocole, l'utilisateur est en possession de son certificat, qui est de la forme (A, x, y) tels que $A^{\gamma+x} h_1^y = g_1$, où h_1 est un paramètre public et y est une valeur secrète choisie par l'utilisateur et connue de lui seul. La preuve de connaissance utilisée pour la signature se modifie aisément afin de prouver la connaissance d'un tel triplet.

Cet ajout renforce la sécurité du schéma, et même s'il rallonge quelque peu la signature, celle-ci reste courte comparée à l'existant.

Chapitre 5

Signatures aveugles

Dans le chapitre précédent, nous avons présenté une signature permettant à l'utilisateur de signer tout en protégeant son anonymat. En 1982, Chaum a introduit une nouvelle signature qui, elle aussi, est utilisée à des fins d'anonymat en cryptographie : la signature aveugle. Contrairement à la signature de groupe, cette signature n'a pas pour objectif de masquer l'identité du signataire, mais de masquer le contenu du message aux yeux de ce dernier. Plus précisément, l'utilisateur obtient une signature sur un message de son choix, en interagissant avec un signataire, sans que celui-ci ait connaissance du message. Le signataire ne doit, à aucun moment, être capable de relier les valeurs qu'il échange avec les utilisateurs aux signatures finales obtenues.

Les signatures aveugles trouvent de nombreuses applications en cryptographie, comme le vote ou la monnaie électronique. Nous les présentons, dans ce chapitre, ainsi que certaines variantes. Nous nous intéressons plus particulièrement à leur sécurité et présentons une faille de sécurité s'appliquant à un certain type de schéma de signature aveugle. Cette faille, ainsi que le moyen de la corriger, ont été présentés avec Jacques Traoré à la conférence *Pairing 2007*.

Sommaire

5.1 Définition et état de l'art

Les signatures aveugles constituent un outil cryptographique très largement répandu (voir par exemple la bibliographie de [Wan07]). S'il n'en n'existe cependant pas de définition universelle (notamment en raison de leurs nombreuses variantes) leur fonctionnement général repose, néanmoins, sur le même principe.

Comme nous l'avons dit, une signature aveugle est un procédé permettant à un utilisateur de demander à un signataire de signer un message dont le contenu lui reste inconnu. Afin que cette propriété perdure après la divulgation du message signé, il est nécessaire que le signataire ne puisse pas relier les signatures finales aux interactions qu'il a eues avec les différents utilisateurs. C'est pourquoi, un protocole de signature aveugle se déroule en trois étapes.

Tout d'abord, l'utilisateur masque (« aveugle ») son message avant de l'envoyer au signataire. Cette procédure peut se réaliser de diverses manières avec ou sans l'implication du signataire. Ensuite, le signataire signe le message aveuglé qu'il reçoit et envoie cette signature à l'utilisateur. Pour finir, l'utilisateur « désaveugle » la signature de telle sorte qu'elle corresponde à son message initial et que le signataire ne puisse pas la tracer.

5.1.1 Usages

Cette manière d'obtenir une signature trouve des applications dans des domaines variés. Deux cependant retiennent l'attention : la monnaie électronique et le vote électronique. Nous détaillerons plus longuement l'utilisation des signatures aveugles dans la monnaie électronique en partie IV.

Dans le vote traditionnel, l'électeur doit respecter un certain nombre d'étapes. Tout d'abord, il sélectionne son vote et cache son bulletin dans une enveloppe. Ensuite il émarge et dépose son vote dans l'urne. La signature aveugle permet de remplir les conditions nécessaires au déroulement légal d'un vote. L'utilisateur choisit son vote qui représente ici son message. Il l'aveugle ensuite en utilisant la première étape d'une signature aveugle. Puis il s'authentifie auprès d'une autorité qui représente le signataire. Ceci lui permet de prouver qu'il est un électeur inscrit et d'émarger. L'autorité signe le message aveuglé et l'utilisateur le désaveugle. Il peut maintenant déposer son vote et la signature dans l'urne. Ainsi, les assesseurs ont l'assurance que son vote a été authentifié et est valide si la signature est correcte, et l'autorité n'est pas capable de relier les votes aux électeurs.

Une telle solution a été proposée par Fujioka, Okamoto et Ohta dans [FOO92]. Les constructions proposées par la suite et utilisant les signatures aveugles sont, en fait, des variantes de cette proposition. Plusieurs projets (comme le projet e-Poll ou VOTOPIA) ont permis de les expérimenter lors de récentes élections. Malgré de nombreuses réticences , le vote électronique est un sujet de recherche en plein développement.

5.1.2 Sécurité

Les signatures aveugles ont pour but de protéger la confidentialité des messages des utilisateurs vis à vis du signataire. Le signataire ne doit donc pas être capable d'apprendre d'information significative sur le message qu'il est en train de signer, ni même de relier une signature à l'une des interactions qu'il a eues avec les utilisateurs. Cette première propriété, appelée indistinguabilité ou aveuglement, protège les utilisateurs contre les signataires malhonnêtes. Mais le signataire doit, lui aussi, avoir la garantie que des utilisateurs malhonnêtes ne peuvent pas produire de signature sans passer par son intermédiaire. Cette deuxième propriété est appelée la $(l, l+1)$-inforgeabilité ou la non-falsification supplémentaire.

On peut de manière informelle décrire ces deux propriétés de la manière suivante :

- l'*indistinguabilité* ou *aveuglement* : supposons que le signataire ait participé à la production des signatures σ et σ' de deux messages m et m' (qu'il ne connaît donc pas), au cours des interactions I et I$'$. Alors, à la vue de (m, σ), le signataire ne peut pas distinguer si ce message signé a été produit lors de l'interaction I ou I$'$.
- la $(l, l+1)$-*inforgeabilité* ou *non-falsification supplémentaire* : il ne doit pas être possible pour un utilisateur de produire $l+1$ signatures après l interactions avec le signataire.

Nous donnons dans la section 5.2 des définitions plus précises et un modèle de sécurité plus détaillé pour ces propriétés.

5.1.3 Construction

Nous nous intéressons maintenant aux algorithmes nécessaires à la construction d'une signature aveugle. Un schéma de signature aveugle implique deux types de participants :

- l'*utilisateur* $\mathcal{U}$ qui désire obtenir une signature d'un message,
- le *signataire* $\mathcal{S}$ qui a pour rôle de signer le message.

Nous avons vu qu'une signature aveugle se construit en trois étapes. Il faut rajouter à celles-ci une procédure de mise en place des clés et une de vérification de la validité de la signature. Ceci nous donne les cinq algorithmes d'un schéma de signature aveugle. Ces algorithmes sont décrits dans la définition suivante.

Définition 81 (Construction d'un schéma de signature aveugle)**.** *Un schéma de signature aveugle se construit à l'aide des procédures suivantes :*

- Setup*, algorithme probabiliste de génération des clés et des paramètre publics. Il prend en entrée un paramètre de sécurité k et retourne les paramètres du système, la clé privée du signataire $sk_{\mathcal{S}}$, la clé publique du signataire $pk_{\mathcal{S}}$.*
- Sign*, protocole de signature entre l'utilisateur et le signataire. Ce protocole se divise en trois sous-algorithmes :*
  - Blind*. Étant donné un message m, un aléa r, renvoie un message aveuglé m' de m.*
  - BSign*. Étant donné un message aveuglé m' et la clé privée du signataire $sk_{\mathcal{S}}$, renvoie une signature aveugle σ'.*

- Unblind. *Étant donné une signature* σ' *et l'aléa* r *utilisé dans le fonction* Blind *renvoie une signature* σ *du message* m.

- Verify*, algorithme de vérification de signature, qui prend en entrée un message* m*, une signature* σ *et la clé publique du signataire* $pk_{\mathcal{S}}$*. Il renvoie 1 si la signature est correcte, 0 sinon.*

Afin de mieux comprendre comment se déroule une procédure de signature aveugle, nous décrivons ici deux schémas classiques de signature aveugle.

5.1.4 Schémas

5.1.4.1 Signature aveugle RSA

Le premier schéma est basé sur la signature RSA. Cette transformation de schéma de signature classique en signature aveugle a été proposée par Chaum dans son premier article sur les signatures aveugles [Cha82].

- Setup : soient
  - p, $q \in \mathbb{N}$ deux entiers premiers,
  - $\mathrm{N} = pq$ un module RSA,
  - e premier avec $\phi(\mathrm{N}) = (p-1)(q-1)$,
  - $d = e^{-1} \bmod \phi(\mathrm{N})$
  - $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_{\mathrm{N}}$

  La clé publique du signataire est donnée par (N, e), sa clé privée par d.

$$(d, (\mathrm{N}, e)) \leftarrow (sk_{\mathcal{S}}, pk_{\mathcal{S}}).$$

- Sign : la phase de signature se déroule en trois étapes.
  - Blind : l'utilisateur $\mathcal{U}$ aveugle son message à l'aide d'un aléa r :
    - ◇ il choisit $r \in \mathbb{Z}_{\mathrm{N}}^*$,
    - ◇ il calcule $h' = \mathrm{H}(m) r^e \bmod \mathrm{N}$,
    - ◇ il envoie h' à $\mathcal{S}$.
  - BSign : le signataire $\mathcal{S}$ peut maintenant signer la version aveuglée du message :
    - ◇ il calcule $\sigma' = h'^d \bmod \mathrm{N} = (\mathrm{H}(m) r^e)^d \bmod \mathrm{N} = \mathrm{H}(m)^d r \bmod \mathrm{N}$,
    - ◇ il envoie σ' à $\mathcal{U}$
  - Unblind : $\mathcal{U}$ obtient sa signature en calculant $\sigma = \sigma' r^{-1} \bmod \mathrm{N} = \mathrm{H}(m)^d$. Il envoie (σ, m) à son destinataire.
- Verify : le destinataire vérifie que $\sigma^e = \mathrm{H}(m) \bmod \mathrm{N}$.

5.1.4.2 Signature aveugle de Schnorr

Le deuxième schéma est une transformation du protocole d'authentification de Schnorr en schéma de signature aveugle. Le schéma est détaillé plus précisément à la figure 5.3.

- Setup : soit p un nombre premier tel que $p-1$ a un grand facteur premier q et g un élément de $\mathbb{Z}_p$ d'ordre q. La clé privée du signataire est une valeur $x \in \mathbb{Z}_q$ et sa clé publique est $h = g^x \bmod p$.
- Sign : la phase de signature se déroule en trois étapes.

- Blind
  - ◇ Le signataire $\mathcal{S}$ choisit une valeur ω aléatoirement dans $[0, q-1]$.
  - ◇ Il calcule ensuite $a = g^{\omega} \mod p$ et l'envoie à l'utilisateur $\mathcal{U}$.
  - ◇ Celui ci choisit aléatoirement deux valeurs dans $[0, q-1]$ et aveugle son message à l'aide de ces valeurs et de la valeur a qu'il a reçu de $\mathcal{S}$. Il envoie la valeur c ainsi calculée à $\mathcal{S}$.
- BSign : le signataire signe la version aveuglée du message et renvoie sa signature $r = \omega - cx \mod q$.
- Unblind : l'utilisateur vérifie la validité de la réponse de $\mathcal{S}$ et calcule ensuite la signature finale sous la forme d'un triplet $\sigma = (c', r', a')$.

- Verify : le destinataire vérifie que $g^{r'} h^{c'} = a' \mod p$.

5.1.5 État de l'art

Chaum, en 1982, avait introduit les signatures aveugles [Cha82] afin d'élaborer un moyen de paiement non traçable. L'usager pouvait ainsi dépenser son argent, sans que la banque auprès de laquelle il l'a retiré ne puisse savoir chez quel commerçant. Chaum a présenté, ensuite, plusieurs améliorations de son schéma [Cha83], [Cha87] et en 1988, Fiat et Naor l'ont rejoint afin de construire un système de monnaie électronique plus sûr [CFN88]. Plus tard, Stadler, Piveteau et Camenisch introduisent en 1995 [SPC95] une nouvelle signature dérivée de la signature aveugle, la signature aveugle à anonymat révocable, sur laquelle nous reviendrons de façon très détaillée dans la partie III.

En 1992, Okamoto a présenté une transformation générique, permettant de passer d'un schéma d'authentification à un schéma de signature aveugle [Oka92]. C'est ainsi qu'a été construit le schéma de signature aveugle de Schnorr que nous avons présenté précédemment. Une amélioration de cette construction a été présentée par Okamoto [Oka95] par la suite. Nous donnons une description de ce schéma à la figure 5.1.

Dans ces premiers articles, la sécurité des schémas n'était pas toujours très détaillée. Les premières constructions prouvées sûres sont dues à Pointcheval et Stern qui, dans [PS96a], donnent un modèle formel pour les signatures aveugles et prouvent la sécurité du schéma de signature aveugle d'Okamoto-Schnorr (*cf* figure 5.1). La sécurité est prouvée dans le modèle de l'oracle aléatoire tant que le nombre l de signatures demandé par l'adversaire dans le jeu de la non-falsification est poly-logarithmiquement borné.

L'année suivante, Juels, Luby et Ostrovsky ont montré comment obtenir un schéma de signature sûr lorsque l'attaquant est autorisé à lancer plusieurs exécutions en parallèle, de manière concurrente [JLO97]. De plus, dans ce modèle, l'attaquant peut abandonner certaines exécutions en cours de route, sans qu'elles soient comptabilisées comme des signatures. Cette construction repose sur l'existence de familles de fonctions à trappe, mais reste théorique et peu applicable en pratique.

Ces nouveaux modèles de sécurité ont permis d'une part de prouver la sécurité de certains schémas, comme l'ont fait Pointcheval et Stern [PS96a], [PS00], mais aussi de construire de nouvelles attaques. Schnorr montre dans [Sch01] comment réussir une forge supplémentaire et ainsi casser la propriété de non-falsification supplémentaire du schéma de signature aveugle d'Okamoto-Schnorr lorsque l'attaquant est autorisé à lancer

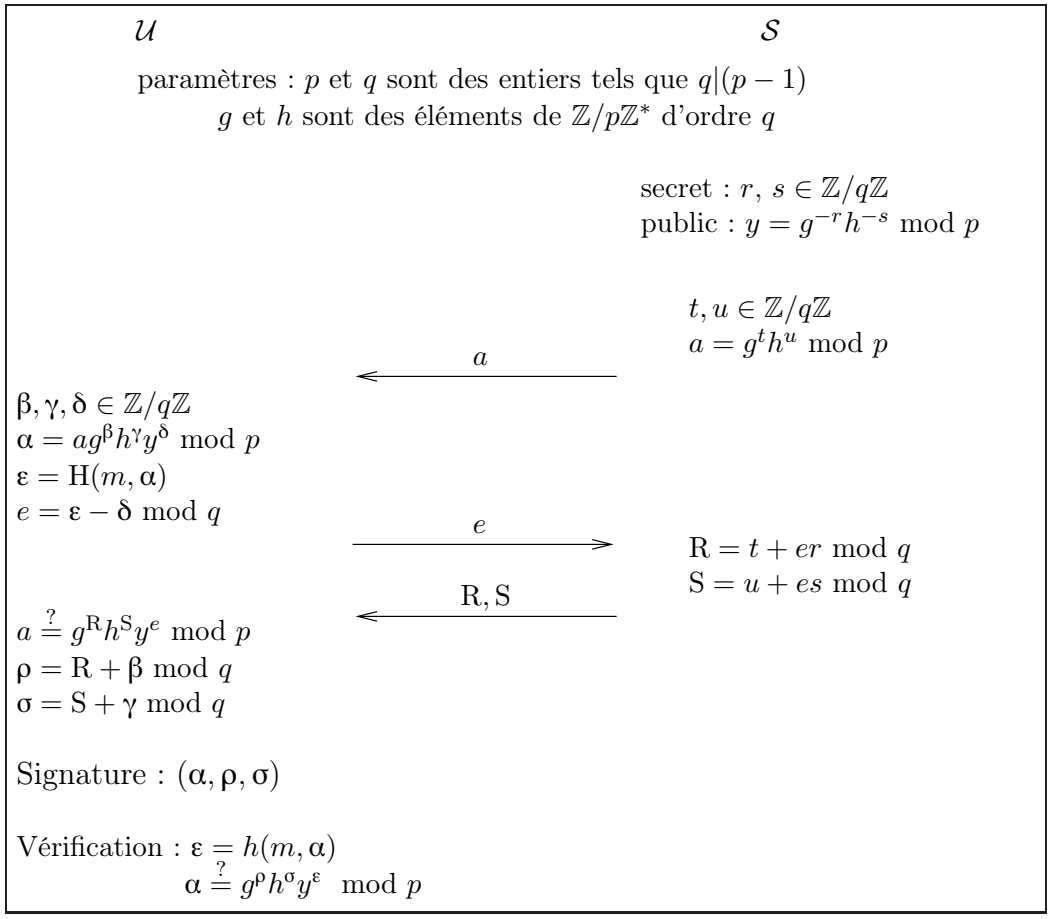

FIG. 5.1 – Signature aveugle d'Okamoto-Schnorr

un nombre poly-logarithmique (en le paramètre de sécurité) d'exécutions du protocole en parallèle. Les preuves sont faites dans le modèle de l'oracle aléatoire et celui du groupe générique. Cette attaque est appelée l'*attaque parallèle générique*.

Avec l'explosion de la cryptographie basée sur l'identité, sont apparus les premiers schémas de signature aveugle basés sur l'identité. Le premier est dû à Zhang et Kim [ZK02]. L'article propose une construction de signature aveugle et de signature en anneau[1]. Leur schéma utilise des applications bilinéaires, ce qui leur permet d'améliorer l'efficacité calculatoire de la construction. Malheureusement, l'attaque générique parallèle de Schnorr [Sch01] s'applique aussi à leur schéma et ils laissent ouvert la question de trouver une preuve formelle contre la falsification supplémentaire existentielle introduite par Pointcheval et Stern [PS00].

Les applications bilinéaires ont par la suite donné lieu à de nombreux autres schémas de signature aveugle. Le schéma de Boldyreva [Bol03] en est un exemple. Cette

[1]Les signatures en anneau constituent un autre procédé cryptographique pour obtenir des signatures anonymes que nous détaillons dans le chapitre 6.

construction utilise l'hypothèse Diffie-Hellman de groupe. Ceci permet de construire un schéma extrêmement simple et efficace. De plus, il est prouvé sûr contre la falsification supplémentaire dans les attaques à messages choisis, sous l'hypothèse calculatoire Diffie-Hellman à "cible choisie" [Bol03]. Nous donnons dans la figure 5.2 une description de ce schéma.

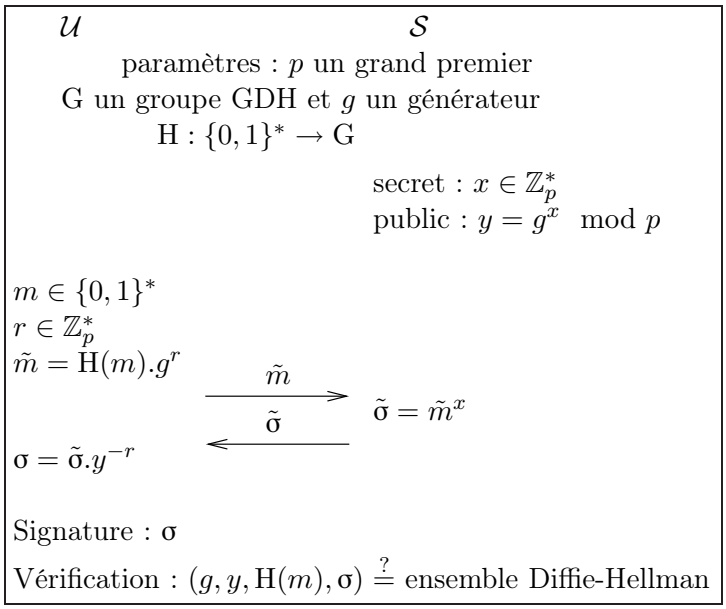

FIG. 5.2 – Signature aveugle de Boldyreva dans les groupes GDH

Si l'utilisation des applications bilinéaires a permis d'augmenter l'efficacité des schémas, cela a aussi donné naissance à des constructions prouvées sûres sous des hypothèses plus fortes et dans des modèles non-standards. C'est pourquoi les auteurs se sont attachés à proposer des constructions prouvées sûres dans le modèle standard. La première est due à Camenisch et Lysyankaya, dans [CL04]. Ils présentent un procédé à la fois efficace et prouvé sûr sans utiliser le modèle de l'oracle aléatoire. La sécurité de ce schéma repose sur le RSA fort et l'hypothèse décisionnelle du résidu composite. Mais il reste néanmoins moins efficace que ceux prouvés sûrs dans le modèle de l'oracle aléatoire, en raison du nombre important de preuves de connaissance que l'utilisateur doit effectuer.

Okamoto, dans [Oka06], présente quant à lui un schéma prouvé sûr dans le modèle standard et dont l'efficacité est comparable à celles des constructions dans le modèle de l'oracle aléatoire. La sécurité repose sur une hypothèse plus forte que l'hypothèse Diffie-Hellman fort qu'il appelle l'hypothèse *Diffie-Hellman fort à deux variables* et le schéma ne résiste pas aux attaques concurrentes classiques (il résiste cependant aux attaques concurrentes à *profondeur constante*, c'est-à-dire quand le nombre d'exécutions lancées en parallèle est polynomialement borné). Okamato propose aussi une version résistante aux attaques concurrentes classiques mais dont la sécurité est prouvée seulement si les paramètres du système respectent certaines contraintes. Kiayias et Zhou [KZ06] proposent eux aussi un schéma de signature aveugle concurrent dans le modèle standard.

Ils prouvent l'indistinguabilité sous les hypothèses LRSW[LRSW99] et Diffie-Hellman linéaire décisionnel. L'inforgeabilité est prouvée sous l'hypothèse LRSW seule. Leur définition d'indistinguabilité est plus faible que celle de [Oka06], mais ils montrent comment renforcer cette définition afin d'obtenir une sécurité équivalente. Comme pour le schéma d'Okamoto [Oka06], ceci implique que l'inforgeabilité repose aussi sur l'hypothèse DCR.

Plus récemment, Fischlin a proposé à Crypto'06 un schéma de signature aveugle composable [Fis06], dont il prouve la sécurité dans le modèle de la composabilité universelle. Ce schéma est utilisé comme brique de base par Hazat, Katz, Koo et Lindell dans [HKKL07] afin d'obtenir un schéma de signature aveugle résistant aux attaques concurrentes et dont la sécurité est prouvée sans oracle aléatoire ni hypothèse sur la mise en place des paramètres du système.

Nous avons donné ici une présentation des schémas de signature aveugle. Mais il existe de nombreuses variantes à ce procédé. Elles utilisent toutes l'anonymat à des fins différentes mais restent basées sur le même principe, consistant à obtenir une signature de la part d'une tierce personne, sans que celle-ci ne puisse avoir connaissance du message. Nous donnons ici quelques exemples de ces variantes :

- les *signatures partiellement aveugles*, introduites par Paillès et Traoré. Elles permettent aux deux parties de s'entendre d'abord sur une partie du message à signer. Ceci permet notamment de donner une valeur à la pièce dans le contexte de la monnaie électronique ou une date de validité [AO00], [Oka06].
- les *signatures aveugles proxy* (*proxy blind signatures*) qui combinent les propriétés des signatures aveugles et des signatures proxy [DX06], [ZSNL03], [AL03].
- les *signatures aveugles de groupe* (*group blind signatures*) et *signatures aveugles en anneau* (*blind ring signatures*,) qui permettent de combiner l'anonymat d'une signature de groupe ou d'anneau et celui d'une signature aveugle [LR98], [HL06].
- les *signatures aveugles à seuil* (*treshold blind signatures*) dans lesquelles la signature est obtenue grâce à la coopération d'un nombre de signataire fixé par le seuil [KKL01].
- les *signatures aveugles restrictives* (*restrictive blind signatures*), qui restreignent le choix des messages à signer pour l'utilisateur [Bra95], [MB02], [CZMS06].

...

5.2 Sécurité des signatures aveugles

En raison des applications très sensibles que peuvent avoir les signatures aveugles, s'assurer de leur sécurité est essentiel. De ce fait, les schémas proposés n'ont cessé de tenter d'augmenter la sécurité des signatures aveugles. Dans cette section, nous définissons plus en détail les propriétés de sécurité. Dans la section suivante, nous montrerons comment un signataire malhonnête peut, dans certains schémas, casser l'une de ces propriétés, en présentant une attaque contre l'indistinguabilité des signatures aveugles. Cette attaque a été présentée à la conférence *Pairing 2007* dans un travail commun avec Jacques Traoré.

Comme nous l'avons dit, la sécurité des signatures aveugles repose sur deux propriétés fondamentales : la non-falsification supplémentaire et l'indistinguabilité. Les pre-

mières définitions formelles ont été données par Pointcheval et Stern [PS96a] et Juels *et al.* [JLO97]. Nous utilisons leurs définitions afin de construire un modèle formel pour les signatures aveugles.

5.2.1 Falsification

A travers cette propriété, le signataire a la garantie qu'aucun utilisateur ne peut émettre de signature vérifiable à l'aide de sa clé publique s'il n'a pas interagi avec celui-ci. Cette propriété doit être vérifiée même si l'utilisateur a déjà demandé plusieurs signatures.

Pointcheval et Stern dans [PS96a] et [PS00] se sont tout particulièrement intéressés à cette propriété et ont donné plusieurs définitions selon les moyens de l'adversaire. Ces définitions sont basées sur le même principe. L'adversaire est autorisé à demander un certain nombre de signatures de manière tout à fait légale à un signataire de son choix. Son but est de produire une signature de plus que le nombre de requêtes qu'il a faites. C'est pourquoi cette propriété porte le nom de non-falsification supplémentaire.

La première définition de Pointcheval et Stern est la plus générale possible. Il s'agit du cas où l'adversaire n'a aucune contrainte sur le nombre de signatures qu'il peut obtenir légalement avant de renvoyer sa forge.

Définition 82 (La $(l, l+1)$-falsification)**.** *Soit un entier l quelconque. Si l'attaquant obtient $l+1$ signatures valides après au plus l interactions avec le signataire, alors il réussit une $(l, l+1)$-falsification.*

La définition suivante réduit la puissance de l'attaquant au cas où l'entier l est polynomialement borné. Il s'agit en fait d'une $(l, l+1)$-falsification pour un entier l polynomialement borné.

Définition 83 (La falsification supplémentaire)**.** *Soit un entier l, polynomial en le paramètre de sécurité k. Si l'attaquant obtient $l+1$ signatures valides après au plus l interactions avec le signataire, alors il réussit une falsification supplémentaire.*

La dernière définition est la plus restrictive. Dans ce cas, l'adversaire est autorité à interagir l fois avec un signataire, mais l est poly-logarithmiquement borné. Comme précédemment, il s'agit d'une version plus faible de la propriété de $(l, l+1)$-falsification.

Définition 84 (La falsification supplémentaire forte)**.** *Soit un entier l, poly-logarithmiquement borné. Si l'attaquant obtient $l+1$ signatures valides après au plus l interactions avec le signataire, alors il réussit une falsification supplémentaire forte.*

Le nombre de requêtes que l'adversaire peut initier (c'est-à-dire le nombre de signatures que l'adversaire est autorisé à demander à un signataire) détermine donc sa puissance.

De plus, plusieurs types de scénarios d'attaques peuvent être envisagés. Tout d'abord, l'attaquant peut décider d'interagir séquentiellement avec le signataire mais il peut aussi se placer dans un modèle plus fort en effectuant des attaques concurrentes. De même, l'adversaire peut aussi corrompre un seul utilisateur ou alors un groupe et lancer des attaques par coalitions.

La définition donnée par Juels *et al.* est similaire à la précédente, mais détaille plus précisément l'expérience que mène l'adversaire lors de son attaque.

Définition 85 (Non-falsification de Juels *et al.*)**.** *L'adversaire $\mathcal{A}$ exécute l'expérience suivante ($\mathcal{A}$ contrôle les utilisateurs, mais le signataire est honnête) :*

- *Étape 1 :* $(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{Setup}(1^k)$
- *Étape 2 : $\mathcal{A}$ s'engage dans des exécutions adaptatives et parallèles de protocoles interactifs avec un nombre polynomialement borné de copies du signataire et peut lancer un nombre polynomial d'exécutions. $\mathcal{A}$ décide de lui même quand il arrête ses exécutions. On note l, le nombre d'exécutions qui ont abouti pour le signataire.*
- *Étape 3 : $\mathcal{A}$ retourne l'ensemble $\{(m_1, \sigma_1), \ldots (m_j, \sigma_j)\}$ et tous les couples $1 \leq i \leq j$ sont acceptés par* $\mathsf{Verify}(m_i, \sigma_i, pk_{\mathcal{S}})$.

Alors, la probabilité que $j > l$ est au plus $\frac{1}{k^c}$, où c est une constante et k un entier assez grand.

Ces définitions ne donnent pas de précisions quant aux couples message/signature donnés par l'adversaire en sortie. On peut distinguer deux situations qui impliquent des propriétés de sécurité plus ou moins fortes. En effet, on peut attendre de l'adversaire que sa sortie soit un ensemble $((m_1, \sigma_1), \ldots, (m_{l+1}, \sigma_{l+1}))$ de signatures valides, telles que $m_i \neq m_j$ pour tout $1 \leq i < j \leq l+1$, mais on peut aussi définir une notion de falsification plus forte en demandant à l'adversaire de produire $l+1$ couples message/signature tels que $(m_i, \sigma_i) \neq (m_j, \sigma_j)$. En général, seule la première définition est retenue et suffit à prouver la sécurité pratique des schémas de signature aveugle ([Oka06], [KZ06], [CL04] [BNPS03]). Fischlin [Fis06] est l'un des rares à utiliser la notion forte de la non-falsification dans ses preuves. Il précise d'ailleurs que se ramener à la définition plus faible simplifie la construction de son schéma.

5.2.2 Indistinguabilité

Lorsqu'il s'engage dans une procédure de signature, l'utilisateur doit avoir la garantie que la signature qu'il obtient à l'issue de la transaction ne peut pas être reconnue par le signataire et, s'il demande plusieurs signatures, que le signataire ne pourra pas les relier entre elles. Cet anonymat est garanti par la propriété d'indistinguabilité ou d'aveuglement (*blindness*).

La première définition formelle de cette propriété a été donnée par Juels *et al.* [JLO97].

Définition 86 (Indistingabilité)**.** *Soit $b \in \{0,1\}$ un bit aléatoire inconnu de $\mathcal{A}$. Afin de mener son attaque contre l'indistinguabilité d'un schéma de signature aveugle, $\mathcal{A}$ s'engage dans l'expérience suivante :*

- *Étape 1 :* $(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{Setup}(1^k)$
- *Étape 2 :* $\{m_0, m_1\} \leftarrow \mathcal{A}(1^k, pk_{\mathrm{S}}, sk_{\mathcal{S}})$
- *Étape 3 : on note $\{m_b, m_{1-b}\}$ les deux messages m_0, m_1, ordonnés selon la valeur du bit b inconnu de $\mathcal{A}$. $\mathcal{A}(1^k, pk_{\mathcal{S}}, sk_{\mathcal{S}}, m_0, m_1)$ s'engage dans deux exécutions parallèles du protocole de signature, la première avec $\mathcal{U}(pk_{\mathcal{S}}, m_b)$ et la seconde avec $\mathcal{U}(pk_{\mathcal{S}}, m_{1-b})$*

- *Étape 4 : si le premier utilisateur renvoie* $\sigma(m_b)$ *et le second* $\sigma(m_{1-b})$ *(c'est-à-dire qu'aucun des deux n'échoue dans l'exécution du protocole) alors* $\mathcal{A}$ *reçoit les deux signatures* $\{\sigma_b, \sigma_{1-b}\}$
- *Étape 5 :* $\mathcal{A}$ *renvoie un bit* $\tilde{b}$

Alors, la probabilité prise sur le choix de b, des aléas de la génération des clés et des aléas de $\mathcal{A}$ *que* $\tilde{b} = b$ *est au plus* $\frac{1}{2} + \frac{1}{k^c}$ *où c est une constante.*

Dans cette attaque, l'adversaire est capable de corrompre un signataire et donc de répondre à des demandes de signature de la part d'utilisateurs honnêtes. A un moment de son choix, il sélectionne deux messages et deux utilisateurs honnêtes et s'engage dans deux protocoles de signature sans savoir quel message est envoyé à chacun des utilisateurs. Il reçoit en réponse les deux signatures dans un ordre aléatoire. Le but de son attaque est alors de déterminer la provenance de chaque signature en renvoyant un bit $\tilde{b}$ correspondant à l'ordonnancement des messages.

Il est possible d'améliorer cette définition en laissant à l'adversaire la possibilité de choisir la clé privée du signataire, ce qui n'est pas fait dans la définition de Juels *et al.* Ceci n'affaiblit nullement la sécurité de la signature. En réalité, il est seulement demandé à l'adversaire de divulguer une clé publique de signature et il peut calculer celle-ci sans même utiliser l'algorithme de génération de clé. Cette amélioration de la définition n'est apparue que récemment dans les modèles de sécurité [Oka06], [Fis06], [HKKL07].

5.3 Une nouvelle attaque sur les signatures aveugles

Dans la définition de l'indistinguabilité, nous avons vu que l'adversaire peut lui même choisir la clé publique du schéma. Mais il est implicitement sous-entendu qu'il la construit en respectant les spécificités du schéma. Une question naturelle est alors de se demander ce qui se passe si l'attaquant modifie volontairement sa clé publique et quelles peuvent en être les conséquences sur l'indistinguabilité du schéma.

L'attaque suivante, présentée avec Jacques Traoré à *Pairing 2007*, montre comment il est possible de casser facilement l'indistinguabilité de certains schémas de signature aveugle si aucun contrôle n'est fait au moment de la mise en place des clés du signataire.

Ce type d'attaque a déjà été étudié pour les protocoles d'échange de clé, principalement ceux basés sur le protocole ElGamal [LL97]. La modification de la clé publique permet alors à l'adversaire de retrouver la clé secrète du membre avec qui il est en train de construire une clé de session. Ce comportement de l'adversaire n'avait jusque là pas été étudié pour les signatures aveugles. Notre attaque est plus précisément ciblée sur les schémas de signature aveugle basés sur l'authentification de Schnorr [Sch91], comme les constructions de [Oka92], (que nous avons décrit figure 5.1), ou les schémas de Zhang *et al.* ([ZK02], [FSNS03]).

Nous décrivons cette attaque en nous basant sur la transformation du schéma d'authentification de Schnorr en signature aveugle que nous avons décrite précédemment. Le schéma est rappelé à la figure 5.3. Les paramètres publics sont choisis de la manière suivante : soit p un nombre premier tiré aléatoirement tel que $p-1$ a un grand facteur premier q. La clé privée du signataire est une valeur $x \in \mathbb{Z}_q^*$ et sa clé publique est $h = g^x \bmod p$ où g est d'ordre q dans $\mathbb{Z}_p$. L'utilisateur $\mathcal{U}$ demande au signataire $\mathcal{S}$ une signature

sur un message m de son choix. H est une fonction de hachage permettant à l'utilisateur de masquer son message avant de l'envoyer au signataire.

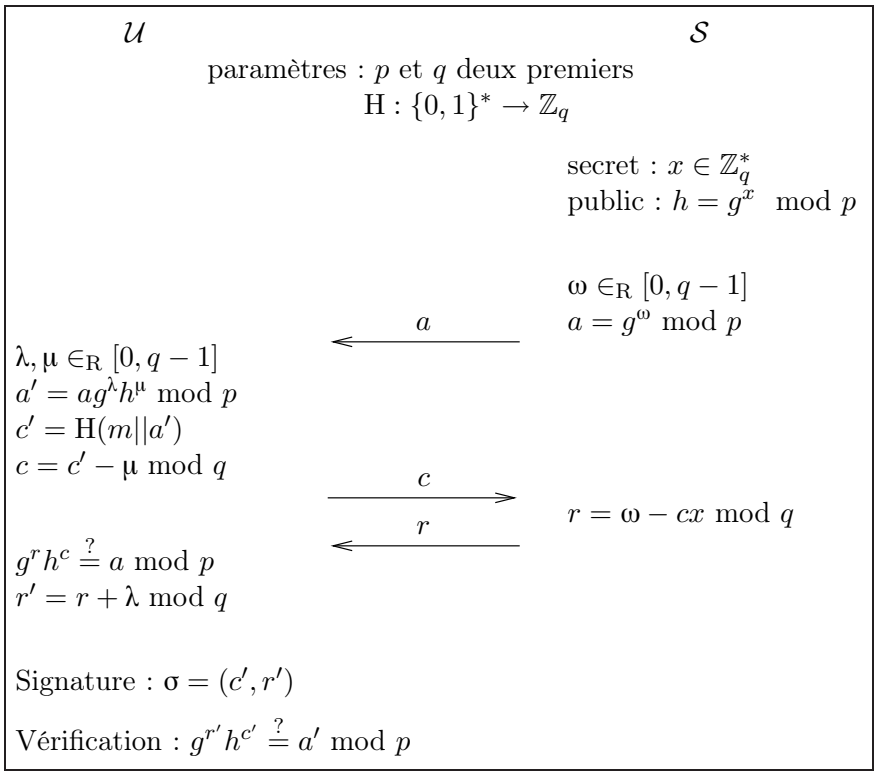

FIG. 5.3 – Signature aveugle de Schnorr

Pour mener à bien son attaque, l'adversaire se place dans le rôle du signataire. Son but est de distinguer deux couples message/signature après qu'il les a signés, et par là, casser l'indistinguabilité du schéma. Nous considérons dans notre description un attaquant $\mathcal{A}$ et deux utilisateurs honnêtes $\mathcal{U}_0$ et $\mathcal{U}_1$.

$\mathcal{A}$ commence par construire les clés de signature. Pour ce faire, il choisit $x \in \mathbb{Z}_q^*$ et $\beta \in \mathbb{Z}_p$ d'ordre petit modulo p. Il calcule ensuite sa clé publique h en incluant la valeur β : $h = \beta g^x \mod p$.

On suppose maintenant que l'utilisateur $\mathcal{U}_0$ demande à $\mathcal{A}$ une signature sur un message m_0 de son choix. $\mathcal{A}$ sait seulement qu'il est en train d'interagir avec l'utilisateur $\mathcal{U}_0$ mais ne peut pas savoir quel message il est en train de signer. L'exécution du protocole est détaillée dans la figure 5.4. Lorsqu'il calcule la valeur a_0, l'adversaire la multiplie par la valeur β qu'il a choisie pour construire sa clé publique. La suite du protocole se déroule de manière normale à condition que l'utilisateur accepte les données du signataire. En effet, lorsque $\mathcal{A}$ renvoie r_0, $\mathcal{U}_0$ vérifie d'abord la validité de la valeur. Il calcule :

$$\begin{aligned} g^{r_0} h^{c_0} &= g^{\omega_0 - c_0 x} h^{c_0} \mod p \\ &= g^{\omega_0} \beta^{c_0} \mod q \end{aligned}$$

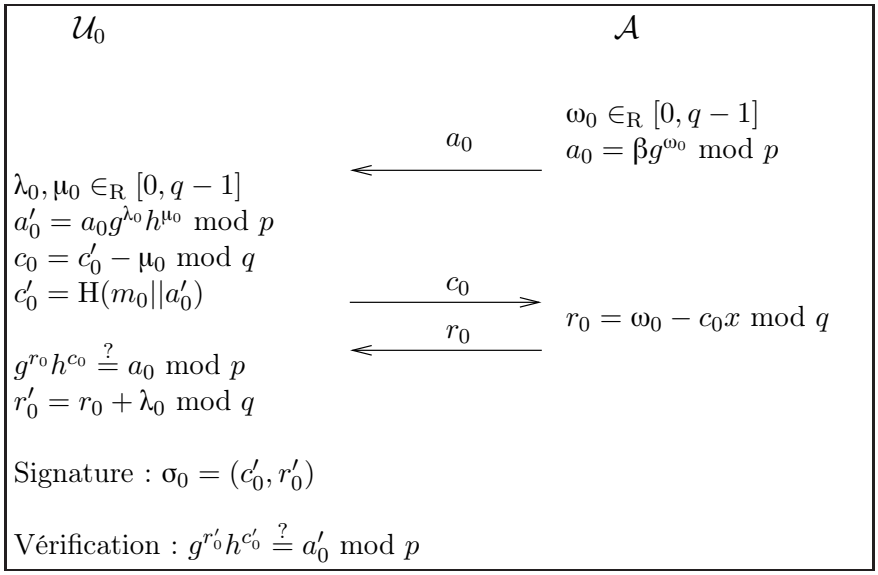

FIG. 5.4 – Signature aveugle de Schnorr avec signataire malhonnête

Ceci est égal à la valeur attendue a_0, si et seulement si, $c_0 \equiv 1 \bmod \text{ord}(\beta)$. Le protocole s'exécute donc jusqu'à la fin avec probabilité $1/\text{ord}(\beta)$.

De même, pour vérifier la validité de la signature, le vérificateur calcule :

$$\begin{aligned} g^{r'_0} h^{c'_0} &= g^{r_0+\lambda_0} h^{c_0+\mu_0} \bmod p \\ &= g^{\omega_0+\lambda} \beta^{c_0} h^{\mu_0} \bmod q \end{aligned}$$

L'équation est correcte si et seulement si $c \equiv 1 \bmod \text{ord}(\beta)$. La signature est donc forcément valide si l'utilisateur a accepté le protocole avec le signataire.

Au final, la signature σ_0 de $\mathcal{U}_0$ est acceptée avec probabilité $1/\text{ord}(\beta)$.

Lorsque l'utilisateur $\mathcal{U}_1$ demande à son tour une signature, l'adversaire exécute le protocole normalement, c'est à dire qu'il calcule $a_1 = g^{\omega_1} \bmod p$ comme dans la figure 5.3.

De même, pour accepter la réponse de l'attaquant, $\mathcal{U}_1$ vérifie la validité de la valeur c_1 :

$$\begin{aligned} g^{r'_1} h^{c'_1} &= g^{r_1+\lambda_1} h^{c_1+\mu_1} \bmod p \\ &= g^{\omega_1+\lambda} \beta^{c_1} h^{\mu_1} \bmod q \end{aligned}$$

En faisant les mêmes calculs de vérification de validité, on obtient que la signature σ_1 de $\mathcal{U}_1$ est acceptée si $c \equiv 0 \bmod \text{ord}(\beta)$, c'est-à-dire avec probabilité $1/\text{ord}(\beta)$.

L'adversaire $\mathcal{A}$ est maintenant à même de distinguer les signatures σ_0 et σ_1 et peut les relier aux utilisateurs $\mathcal{U}_0$ et $\mathcal{U}_1$. Pour cela, il calcule la valeur $a_i'^q \bmod p$, pour $i \in \{0, 1\}$. Si cette valeur vaut 1, il sait que la signature provient de l'utilisateur $\mathcal{U}_1$. Sinon, il s'agit de la signature demandée par $\mathcal{U}_0$. Sa probabilité totale de succès est donc de $1/\text{ord}(\beta)^2$. L'ordre de β étant choisi suffisamment petit, cette probabilité est non négligeable et notre adversaire a cassé l'indistinguabilité du schéma.

Afin d'éviter toute attaque de ce type, l'utilisateur doit vérifier la consistance de la clé publique du signataire lorsqu'il la reçoit. Dans notre cas, il doit vérifier que $h^q \equiv 1 \mod p$.

Chapitre 6

Signatures d'anneau et authentification anonyme

Dans ce chapitre, nous nous intéressons à une autre brique cryptographique permettant à un utilisateur de signer anonymement un message de son choix : la signature d'anneau. Introduites en 2001 par Rivest, Shamir et Tauman [RST01][1], ces signatures sont très proches des signatures de groupe dans leur concept, mais leur construction particulière permet de nouvelles applications. Nous présentons aussi, dans ce chapitre, un état de l'art des signatures d'anneau et détaillons une nouvelle construction, dans un anneau composé de deux membres. Cette signature est basée sur le schéma d'identification HLS [HLS05] présenté au workshop WEWoRC[2] dans un travail commun avec David Lefranc et Hervé Sibert. Nous montrons aussi, comment cette signature peut-être utilisée comme preuve du « OU ».

Sommaire

[1] En 1994, Cramer, Damgård et Schoenmakers ont introduit un concept très proche [CDS94] sous une dénomination différente mais sans l'appliquer aux signatures.

[2] Western Europe Workshop on Research in Cryptology.

6.1 Définition et état de l'art

Les signatures d'anneau, comme les signatures de groupe, permettent à un utilisateur de signer anonymement au sein d'un ensemble de personnes, de telle sorte que le destinataire ne puisse identifier le membre à l'origine de la signature (il sait de quel groupe/anneau provient la signature, mais pas qui est le signataire).

Comme nous l'avons vu au chapitre 4, avant de pouvoir produire une signature de groupe, tout nouveau membre doit s'enregistrer auprès du manager du groupe. Dans les signatures d'anneau, il n'y a pas de telle phase. Lorsqu'un utilisateur veut signer un message, il construit lui même son anneau, en choisissant librement les membres qui l'intéressent. Il peut même changer d'anneau pour toute nouvelle signature. La seule condition requise pour être membre d'un anneau est d'avoir publié une clé publique de signature (une clé RSA par exemple).

Contrairement aux signatures de groupe, il n'existe pas de procédure de révocation pour les signatures d'anneau. Elles fournissent donc un anonymat inconditionnel pour le signataire et même un juge doté d'une puissance de calcul non bornée ne peut pas retrouver l'identité du signataire[3].

6.1.1 Usages

L'application donnée dans l'article [RST01] de cette nouvelle notion, est la délation. Supposons qu'un membre d'un ministère veuille divulguer une information tout en gardant son anonymat. Il va constituer son anneau en prenant sa clé publique ainsi que celles d'autres membres du ministère et signer son message en utilisant un protocole de signature d'anneau. Il envoie ensuite son message au journaliste de son choix. Le journaliste aura la preuve que le message provient du personnel de ce ministère, puisque toutes les clés publiques appartiennent à des membres du ministère, mais ne saura pas duquel.

Les signatures d'anneau peuvent aussi être utilisées à des fins plus nobles. Comme elles fournissent un anonymat inconditionnel, ces signatures permettent de protéger des données extrêmement sensibles. Supposons par exemple que le signataire ait utilisé des signatures basées sur RSA dans sa construction. Il a alors la garantie que son anonymat sera conservé, quand bien même RSA viendrait à être cassé.

Une autre application est donnée par Rivest *et al.* dans [RST01]. Supposons que deux compagnies A et B souhaitent échanger des propositions de contrat de manière à ce que ni A, ni B, ne puisse convaincre une tierce personne de l'identité (A ou B) de l'auteur d'une proposition. On considère alors l'anneau composé seulement de A et B. Si la compagnie A signe toutes ses propositions à l'aide d'une signature d'anneau B ne pourra convaincre personne que A est à l'origine de la signature, car B aurait pu elle-même produire cette signature. Ainsi, la signature d'anneau peut-elle être utilisée comme signature à vérificateur désigné.

[3]Les signatures de groupe, elles, ne proposent qu'un anonymat calculatoire.

6.1.2 Sécurité

Les propriétés de sécurité souhaitées pour un schéma de signature d'anneau sont très proches de celles d'un schéma de signature de groupe. Tout d'abord, il ne doit pas être possible de construire de nouvelles signatures d'anneau dont on ne serait pas membre, c'est-à-dire qu'un schéma de signature d'anneau doit être *non-falsifiable*. De plus, la signature ne doit donner aucune information sur le membre qui a réellement produit cette signature, c'est-à-dire que le schéma doit avoir la propriété d'*anonymat*. En revanche, il n'existe pas d'équivalent à la propriété de non-diffamation, étant donné qu'une signature d'anneau n'est pas révocable.

6.1.3 Construction

Dans leur article [RST01], Rivest, Shamir et Tauman définissent les signatures d'anneau à l'aide de deux algorithmes, l'un pour signer le message, l'autre pour vérifier la validité de la signature. Il n'existe pas à proprement parler de phase de Setup dans une telle signature. En effet, le signataire construit son anneau à partir de membres possédant déjà des clés de signature. Le signataire utilise les paramètres de ces schémas de signature pour signer son message et par conséquent, n'a pas besoin de re-fixer de paramètres. La description d'un schéma de signature d'anneau comporte cependant un algorithme de génération de clé propre à chaque membre. Nous utilisons ici la définition donnée par Bender, Katz et Morselli dans [BKM06].

Un anneau R est noté $\mathrm{R} = (pk_1, \ldots, pk_n)$ et $\mathrm{R}[i] = pk_i$ et on dit que $pk \in \mathrm{R}$ s'il existe un indice i tel que $\mathrm{R}[i] = pk$. On suppose aussi que les clés sont toujours ordonnées lexicographiquement.

Définition 87 (Construction d'un schéma de signature d'anneau)**.** *Un schéma de signature d'anneau se construit à l'aide des procédures suivantes :*

- $\mathsf{Gen}(1^k)$*, algorithme probabiliste de génération de clés. Il prend en entrée un paramètre de sécurité* k *et renvoie une clé publique* pk *et une clé secrète* sk*.*
- $\mathsf{RSign}_{s,sk}(m, \mathrm{R})$*, algorithme probabiliste qui produit une signature d'anneau* σ *sur le message* m*, en prenant en entrée l'anneau* $\mathrm{R} = (pk_1, \ldots, pk_n)$*. On suppose que :*
  - $(\mathrm{R}[s], sk)$ *est une paire clé publique/privée valide, renvoyée par* Gen*,*
  - *l'anneau est composé d'au moins deux clés publiques distinctes,*
  - *toutes les clés publiques de l'anneau sont différentes.*
- $\mathsf{RVerify}_{\mathrm{R}}(m, \sigma)$*, algorithme déterministe vérifiant la validité de la signature* σ *sur le message* m*. Il renvoie* 1 *si la signature est valide,* 0 *sinon.*

6.1.4 Variantes des signatures d'anneau

Comme nous l'avons vu, la signature d'anneau ne requiert pas de procédure d'initialisation particulière. De plus, elle permet de signer sans obtenir de coopération de la part d'autres membres et fournit un anonymat inconditionnel pour le signataire puisqu'il n'existe pas de procédure de révocation. Cette absence de révocation peut, pour des personnes mal-intentionnées, entraîner des utilisations abusives (notons que la première application donnée par Rivest, Shamir et Tauman peut déjà être considérée comme une application abusive).

Suite à la construction de Rivest *et al.* , les signatures d'anneau qui ont été proposées ont pour la plupart d'entre elles, cherché à limiter la puissance du signataire. Cela à conduit à de nombreuses variantes qui peuvent être classées en différentes catégories. Nous présentons ici, les plus significatives d'entre elles.

6.1.4.1 Signature d'anneau à seuil

Cette signature prend en entrée un seuil t et chaque signature comprend la preuve qu'au moins t membres de l'anneau authentifient le message. Ceci garantit donc une certaine protection contre les délations isolées. Les signatures à seuil ont été étudiées, principalement dans le modèle de l'oracle aléatoire. C'est le cas notamment de [BSS02] dont la sécurité du schéma est prouvée sous l'hypothèse RSA.

6.1.4.2 Signature d'anneau à accès général

Les différents acteurs possibles d'une signature d'anneau sont regroupés par ensembles. Les membres d'un ensemble peuvent choisir librement n'importe quelle famille d'ensembles (incluant leur propre ensemble) et prouver que tous les membres d'un de ces ensembles ont coopéré au calcul de la signature, sans dévoiler lequel. Ce type de signature est une généralisation des signatures d'anneau à seuil. Dans [CDS94], Damgard *et al.* avaient déjà envisagé un tel scénario et montré (utilisant une terminologie différente) qu'une preuve de connaissance combinée à l'heuristique de Fiat-Shamir [FS86] permettaient d'obtenir des signatures qu'on pourrait appeler *signatures d'anneau à accès monotone*, dans le sens où le choix des ensembles est plus contrôlé que dans une signature à accès général. Naor dans [Nao02] présente un schéma de signature d'anneau à accès général qui peut dans certains cas être transformé en schéma de signature d'anneau à seuil. Ce schéma est interactif et sa sécurité repose sur l'existence de schéma de signature sûrs. D'autres travaux, comme [HS04], ont considéré les signatures d'anneau à accès général.

6.1.4.3 Signature d'anneau vérifiable

Le signataire peut, s'il désire lever son anonymat, apporter une preuve formelle, vérifiable par le destinataire, qu'il est bien l'auteur de la signature. Cette notion a été introduite par Rivest, Shamir et Tauman dans le premier article sur les signatures d'anneau [RST01] et formalisée par Lv et Wang [LW03].

6.1.4.4 Signature d'anneau reliable

Ce type de signature permet à quiconque de déterminer si deux signatures d'anneau ont été produites par la même personne. Liu, Wei et Wong proposent un tel schéma dans [LWW04]. Ce schéma a aussi la particularité d'être modifiable en schéma de signature d'anneau à seuil [TWC+04].

6.1.4.5 Signature d'anneau responsable

Cette notion, assez proche des signatures d'anneau reliables, permet à n'importe qui de vérifier que le signataire appartient bien à un ensemble possible de signataires (cet ensemble de signataires n'étant pas fixe). De plus, le signataire véritable peut être identifié par une autorité de confiance. Cette propriété a été introduite par Xu et Yung [XY04]. Ils proposent une transformation générique de schéma de signature d'anneau, en schéma de signature d'anneau responsable.

6.1.4.6 Signature d'anneau déniable

Toujours par soucis de limiter les abus sur les signatures d'anneau, Naor, dans [Nao02], définit la notion de *signature en anneau déniable*. Cette notion permet à un membre d'un anneau de convaincre un vérificateur qu'un message m a bien été authentifié par un des membres de l'anneau, tout en préservant son anonymat. Mais le vérificateur ne peut pas à son tour convaincre une tierce personne de l'authenticité du message. La construction est basée sur une interaction entre un prouveur et un vérificateur et est donc interactive. Dans [SM03], Susilo et Mu construisent une signature d'anneau déniable non-interactive, afin de rendre ce type de signature plus facilement utilisable.

D'autres variantes des signatures d'anneau se sont attachées à simplifier les constructions afin de rendre ces signatures plus facilement applicables.

6.1.4.7 Signature d'anneau séparable

Pour construire son anneau, l'utilisateur doit choisir des membres possédant une clé publique de signature. Dans les premiers schémas de signature d'anneau, cette clé devait être de même type pour tous les membres, ce qui limite considérablement l'utilisation de ces signatures. Abe, Okhubo et Suzuki ont introduit dans [AOS02], les signatures d'anneau séparables. Elles permettent d'utiliser pour la construction de l'anneau, des personnes n'ayant pas forcément des clés de signature pour les mêmes types de schéma. Le schéma d'Abe *et al.* permet de construire un anneau dont les membres possèdent des clés de schémas de signature basés sur RSA ou sur le problème de Logarithme Discret. Ce schéma a été étendu aux signatures d'anneau à seuil par Liu *et al.* dans [LWW03].

6.1.4.8 Signature d'anneau courte

Dans la plupart des schémas, la taille de la signature dépend de la taille de l'anneau (la taille est au moins linéaire en la taille du groupe). Dodis *et al.* construisent dans [DKNS04] une signature d'anneau dont la taille est indépendante du nombre de participants. Leur schéma est prouvé sûr dans le modèle de l'oracle aléatoire, sous l'hypothèse RSA fort.

6.2 Sécurité des signatures d'anneau

L'absence de juge et de manager permet de construire un modèle de sécurité plus simple que pour les signatures de groupe. Un schéma de signature d'anneau doit tout d'abord être consistant et vérifier ensuite les propriétés d'anonymat et de résistance aux falsifications universelles.

Dans cette section, nous nous intéressons aux propriétés des signatures d'anneau en général. Nous ne considérons donc pas les propriétés introduites par les variantes. Les définitions de sécurité les plus formelles sont dues à Bender, Katz et Morselli [BKM06]. En effet, les définitions précédentes, dues à Bresson [Bre02] ne prenaient pas en compte le cas d'un adversaire capable de choisir lui même les clés publiques des membres d'un anneau. Toutes les clés étaient considérées comme honnêtement générées. Cette restriction forte ne peut pas garantir la sécurité d'un schéma de signature d'anneau pour une utilisation pratique : étant donné que le membre qui veut signer construit lui même son anneau, il peut, involontairement, choisir un membre corrompu par un adversaire. Dans ce cas, rien ne garantit la sécurité du schéma.

La sécurité d'un schéma de signature d'anneau est donné par sa consistance et, comme nous l'avons vu, par les propriétés d'anonymat et de non-falsification.

Consistance : comme pour les signatures de groupe, la *consistance* des signatures d'anneau garantit la validité des protocoles. Elle décrit donc ce qui se passe lorsque les joueurs sont honnêtes, sans la présence d'adversaire.

Non-falsification : la notion de *non-falsification* pour les signatures d'anneau est très proche de celle de traçabilité pour les signatures de groupe. En effet, elle garantit qu'un adversaire ne peut pas produire de signature d'anneau s'il n'est pas l'un des membres spécifiés pour la signature. La définition de [BKM06] prend en compte le fait que l'adversaire puisse corrompre certains membres de l'anneau impliqués dans la signature.

Anonymat : Un adversaire $\mathcal{A}$ ne doit pas être capable de déterminer l'auteur d'une signature d'anneau donnée. La définition d'anonymat pour les signatures d'anneau considère le cas où seuls deux membres de l'anneau concerné ne sont pas corrompus. L'adversaire doit alors décider qui, de ces deux participants, est à l'origine de la signature. Il est nécessaire d'avoir deux membres non corrompus, faute de quoi l'adversaire sait directement que le seul membre non corrompu est le signataire.

6.3 Un schéma d'authentification dans les groupes GDH

Dans cette section, nous présentons un nouveau schéma d'identification [HLS05] dont la sécurité repose sur le problème du "Gap" Diffie-Hellman (cf. définition dans le paragraphe 1.4.3). Ce protocole qui résulte d'un travail commun avec David Lefranc et Hervé Sibert a été présenté au workshop WEWoRC en 2005.

La construction est basée sur le protocole d'identification de Schnorr [Sch89] et utilise des applications bilinéaires. Elle permet donc à un prouveur de prouver à un vérificateur qu'il connaît un secret sans le dévoiler.

6.3.1 Description du schéma

6.3.1.1 Paramètres publics

Soit un entier k, le paramètre de sécurité. Soient $\mathbb{G}$ un groupe *gap* Diffie-Hellman, d'ordre un grand premier p et noté multiplicativement, $\mathbb{G}_1$ un groupe cyclique d'ordre q et une application bilinéaire admissible $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$. On note g un générateur de $\mathbb{G}$. On peut alors définir l'application

$$\begin{aligned} \psi : \quad \mathbb{G} &\to \mathbb{G}_1 \\ \alpha &\mapsto e(g, \alpha) \end{aligned}$$

Le prouveur $\mathcal{P}$ choisit $a \in \mathbb{Z}_q^*$ et calcule sa clé privée $sk_{\mathcal{P}} = g^a$. Sa clé publique est $pk_{\mathcal{P}} = e(g, g^a)$ et les paramètres publics $params$ =$(g$,$e(g,g)$, $e(g,g)^a)$. Dans ce protocole, $\mathcal{P}$ prouve, pour s'authentifier, la connaissance de son secret g^a tel que $pk_{\mathcal{P}} = e(g, g^a)$.

6.3.1.2 Protocole

Dans sa construction théorique, le protocole d'authentification consiste en l itérations d'un échange à trois passes entre le prouver $\mathcal{P}$ et le vérificateur $\mathcal{V}$. Pour des raisons d'efficacité, on considère que l vaut 1.

Nous décrivons, dans la figure 6.1, une exécution du protocole.

1. Tout d'abord, $\mathcal{P}$ choisit un aléa r et calcule la valeur $\mathrm{W} = e(g, g)^r$ qu'il envoie au vérificateur.
2. Celui-ci choisit le challenge c auquel devra répondre $\mathcal{P}$ et lui envoie.
3. $\mathcal{P}$ est maintenant capable de calculer sa réponse $\mathrm{Y} = g^r \times sk_{\mathcal{P}}^c$.
4. Le vérificateur authentifie le prouveur en vérifiant si $e(g, \mathrm{Y}) = \mathrm{W} \times pk_{\mathcal{P}}^c$.

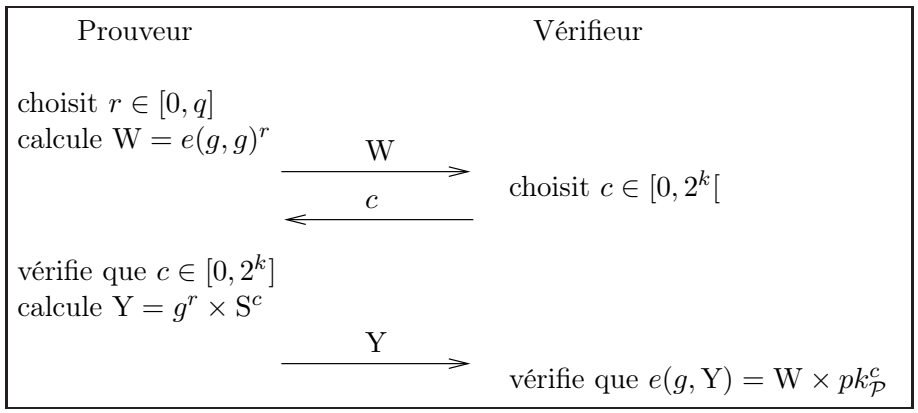

FIG. 6.1 – Protocole d'authentification HLS

6.3.2 Sécurité du protocole

Afin de prouver que notre protocole est un schéma d'identification à divulgation nulle de connaissance, nous avons besoin de vérifier qu'il est consistant et qu'il vérifie les propriétés de significativité et de divulgation nulle de connaissance (*cf.* section 1.4.3).

Théorème 15. *Si le prouveur et le vérificateur respectent les spécifications du protocole, alors le protocole HLS est consistant.*

Démonstration.

$$\begin{aligned} e(g, \mathrm{Y}) &= e(g, g^r (g^{ab})^c) \\ &= e(g, g^{r+abc}) \\ &= e(g, g)^{r+abc} \\ &= e(g, g)^r \times e(g, g)^{abc} \\ &= e(g, g)^r \times (e(g, g)^{ab})^c \\ &= \mathrm{W} \times pk_{\mathcal{P}}^c \end{aligned}$$

□

Nous montrons maintenant que notre protocole est significatif, c'est-à-dire qu'un prouveur accepté avec une probabilité non négligeable peut être utilisé pour retrouver la clé secrète du prouveur et par là, résoudre le problème GDH.

Théorème 16. *Le protocole HLS est significatif si le nombre d'exécutions l est polynomial en $|\mathbb{G}|$ et si $\log_2 |\mathbb{G}| = o(l \times k)$.*

Démonstration. Tout d'abord, on peut remarquer qu'un attaquant $\tilde{\mathcal{P}}$ contre la significativité de notre schéma a une probabilité non nulle de réussite. En effet, à chaque exécution, il peut essayer de deviner la valeur du challenge c que lui renvoie le vérificateur. Ensuite, il choisit aléatoirement la valeur Y et calcule W comme étant $\mathrm{W} = e(g, \mathrm{Y}) pk_{\mathcal{P}}^{-c}$. Les trois valeurs $\mathrm{W}, c, \mathrm{Y}$ définissent un triplet valide, c'est-à-dire que l'équation $e(g, \mathrm{Y}) = \mathrm{W} \times pk_{\mathcal{P}}^c$ est vérifiée. La probabilité d'un tel adversaire est au moins égale à $1/2^k$ pour chaque exécution. Si l'on considère l exécutions du protocole, sa probabilité de succès totale est au moins $1/2^{kl}$.

En notant $\mathrm{A}cc(\omega_{\tilde{\mathrm{P}}}, c_1, \ldots, c_l)$ l'événement correspondant à l'acceptation de $\tilde{\mathcal{P}}$ on obtient :

$$\Pr_{\omega_{\tilde{\mathcal{P}}}, c_1, \ldots, c_l} = \big(\mathrm{A}cc(\omega_{\tilde{\mathrm{P}}}, c_1, \ldots, c_l)\big) \geq 1/2^{kl}$$

Il nous reste à montrer maintenant que cette probabilité est au plus égale à $1/2^{kl}$. Pour ce faire, nous raisonnons par l'absurde et supposons qu'il existe un attaquant $\tilde{\mathcal{P}}$ dont le succès est tel que :

$$\Pr_{\omega_{\tilde{\mathcal{P}}}, c_1, \ldots, c_l} = \big(\mathrm{A}cc(\omega_{\tilde{\mathrm{P}}}, c_1, \ldots, c_l)\big) \geq \varepsilon'$$

où $\varepsilon' = 1/2^{lk} + \varepsilon$, où ε est une quantité non négligeable.

$i \leftarrow 0$
répéter
  $i \leftarrow i + 1$
  Choisir un ruban aléatoire $\omega_{\tilde{\mathcal{P}}}$
  Choisir $(c_1, \ldots, c_l)$ au hasard dans $[0, 2^k]^l$
jusqu' à $\mathrm{A}cc(\omega_{\tilde{\mathrm{P}}}, c_1, \ldots, c_2)$
$j \leftarrow 0$
répéter
  $j \leftarrow j + 1$
  Choisir $(c'_1, \ldots, c'_l) \neq (c_1, \ldots, c_l)$ au hasard dans $[0, 2^k]^l$
jusqu' à $\mathrm{A}cc(\omega_{tilde\mathrm{P}}, c_1, \ldots, c_2)$ ou $j = i$
si $\mathrm{A}cc(\omega_{tilde\mathrm{P}}, c_1, \ldots, c_2)$
alors
  soit t tel que $c_t \neq c'_t$
  retourner $\big((c_t, y_t)$ et $(c'_t, y'_t)\big)$
sinon
  retourner ECHEC
fin de si

TAB. 6.1 – Extracteur de Schnorr

La suite de notre démonstration est très proche de la démonstration de Schnorr [Sch91] qui utilise un extracteur $\mathcal{E}$. Nous rappelons dans le tableau 6.1, le fonctionnement de cet extracteur $\mathcal{E}$, utilisé pour retrouver la clé secrète du prouveur.

Pour prouver notre théorème, nous avons aussi besoin du lemme de séparation 4 (*cf.* section 2.2.3). Nous appliquons ce théorème avec ε', $\alpha = \varepsilon/2$ et Ω l'ensemble des rubans tels que :

$$\Pr_{c_1,\ldots,c_l}\{\mathrm{S}ucc(\tilde{\mathcal{P}}(\omega), c_1, \ldots, c_l)\} = 1/2^{lk} + \varepsilon/2.$$

La probabilité qu'ω appartienne à Ω étant au moins de $\varepsilon/2$, il faut en moyenne $2/\varepsilon$ tentatives avant d'obtenir un ω appartenant à Ω.

Si $\omega \in \Omega$, alors la probabilité de succès prise sur l'ensemble à l éléments $(c_1, \ldots, c_l)$ est égale à $1/2^{lk} + \varepsilon/2$. Cette probabilité étant strictement supérieure à $1/2^{lk}$, il existe forcément deux réponses correctes correspondant à un i.

Par conséquent, si on exécute l'algorithme $\mathcal{E}$, $2/\varepsilon$ fois, on obtient un succès avec probabilité $1/2^{lk} + \varepsilon/2$ et deux couples (c_1, Y_1) et (c_2, Y_2) pour deux valeurs c_1 et c_2 différentes. Le temps total d'exécution de l'algorithme est alors de $l \times 2^k \times \tau \times (2/\varepsilon)$, avec τ le temps d'exécution d'un tour du protocole.

Ces deux couples de valeurs (c_1, Y_1) et (c_2, Y_2) vont nous permettre d'extraire la clé secrète du prouveur.

Tout d'abord, on constate l'égalité suivante :

$$e(g, \mathrm{Y}_1)/pk_{\mathcal{P}}^{c_1} = e(g, \mathrm{Y}_2)/pk_{\mathcal{P}}^{c_2}. \tag{6.1}$$

On peut supposer que $c_2 < c_1$ et que donc $0 < c_1 - c_2 < q$. Comme q est un nombre premier, l'algorithme d'Euclide étendu nous donne α et β tels que $\alpha(c_1 - c_2) + \beta q = 1$. On réécrit alors l'équation 6.1 :

$$e(g, (\mathrm{Y}_1 \mathrm{Y}_2^{-1})^{\alpha}) = e(g, g)^{ab} = e(g, g^{ab}).$$

Par conséquent,

$$(\mathrm{Y}_1 \mathrm{Y}_2^{-1})^{\alpha} = g^{ab}.$$

Nous avons ainsi obtenu la clé privée $sk_{\mathcal{P}}$ du prouveur.

La probabilité de succès de l'extracteur $\mathcal{E}$ est supérieure à $\frac{\varepsilon^2}{6\varepsilon'^2}$ et son temps de fonctionnement est $2l\tau/\varepsilon'$. Ce temps est polynomial en $|\mathbb{G}|$. La probabilité de succès le l'attaquant, qui vaut $1/2^{lk}$, est négligeable si $\log_2 |\mathbb{G}| = o(l \times k)$. □

Il nous reste à prouver maintenant que notre schéma est à divulgation nulle de connaissance.

Théorème 17. *Le protocole HLS est à divulgation nulle de connaissance si $l \times 2^k$ est polynomial en $|\mathbb{G}|$.*

Démonstration. Afin de prouver la divulgation nulle de connaissance de notre schéma, nous devons montrer qu'il existe un algorithme qui simule en temps polynomial les communications entre un prouver $\mathcal{P}$ et un simulateur non nécessairement honnête $\tilde{\mathcal{V}}$.

Comme pour la significativité de notre schéma, cette preuve est très proche de celle du schéma d'authentification de Schnorr. Nous allons comparer les distributions de probabilité entre des communications réelles et simulées.

Dans une première partie, nous nous intéressons à la distribution des communications réelles entre $\mathcal{P}$ et $\tilde{\mathcal{V}}$ et on suppose que les challenges c choisis par $\tilde{\mathcal{V}}$ sont sélectionnés selon une fonction $\mathrm{C}(\mathrm{W}, \omega_{\tilde{\mathcal{V}}}, hist)$ où $hist$ est l'historique des communications précédentes. $\tilde{\mathcal{V}}$ peut donc dans notre preuve choisir ses challenges c en fonction des exécutions passées. Étant donnés $(\mathrm{W}_0, c_0, \mathrm{Y}_0) \in \mathbb{G}_1 \times [0, 2^k] \times \mathbb{G}$, la distribution des communications $(\mathrm{W}, \mathrm{Y}, c)$ est donnée par :

$$\begin{aligned}
& \Pr_{\omega_{\mathcal{P}}, \omega_{\tilde{\mathcal{V}}}} \big((\mathrm{W}, c, \mathrm{Y}) = (\mathrm{W}_0, c_0, \mathrm{Y}_{\mathrm{O}})\big) \\
= & \sum_{r \in [0, q[} \frac{1}{q} \times \Pr_{\omega_{\tilde{\mathcal{V}}}} \big(e(g, \mathrm{Y}_0) pk_{\mathcal{P}}^{-c_0} = \mathrm{W}_0 \wedge g^r = \mathrm{Y}_0 \mathrm{S}^{-c_0} \wedge c_0 = \mathrm{C}(\mathrm{W}_0, \omega_{\tilde{\mathcal{V}}}, hist)\big) \\
= & \frac{1}{q} \times \Pr_{\omega_{\tilde{\mathcal{V}}}} \big((e(g, \mathrm{Y}_{\mathrm{O}}) pk_{\mathcal{P}}^{-c_0} = \mathrm{W}_0 \wedge \mathrm{C}(\mathrm{W}_0, \omega_{\tilde{\mathcal{V}}}, hist)\big)
\end{aligned}$$

Nous obtenons alors la probabilité de distribution suivante :

$$\Pr_{\omega_{\mathcal{P}}, \omega_{\tilde{\mathcal{V}}}} \big((\mathrm{W}, c, \mathrm{Y}) = (\mathrm{W}_0, c_0, \mathrm{Y}_{\mathrm{O}})\big) =$$
$$\begin{cases} \frac{1}{q} \times \Pr_{\omega_{\tilde{\mathcal{V}}}} \big(c_0 = (\mathrm{C}(\mathrm{W}_0, \omega_{\tilde{\mathcal{V}}}, hist)\big) & \text{si } \mathrm{W}_0 = e(g, \mathrm{Y}_0) pk_{\mathcal{P}}^{-c_0} \\ 0 & \text{sinon} \end{cases}$$

Nous étudions maintenant cette même distribution, lorsque l'algorithme est modélisé par une PPTM interactive M avec un ruban aléatoire ω_{M} simulant de telles communications. Afin de simuler parfaitement les communications, M choisit d'abord aléatoirement $\bar{c}$ dans $[0, 2^k]$ et $\bar{\mathrm{Y}}$ dans $\mathbb{G}$. Finalement, M calcule $\bar{\mathrm{W}} = e(g, \bar{\mathrm{Y}})pk_{\mathcal{P}}^{-\bar{c}}$.

Dans notre premier calcul, $\tilde{\mathcal{V}}$ ne choisit pas les challenges c de manière aléatoire. Nous devons donc tester si la valeur $\bar{c}$ choisie aléatoirement par M peut être envoyée à $\tilde{\mathcal{V}}$ à la réception de $\bar{\mathrm{W}}$. Après que M ait créé son triplet $(\bar{\mathrm{W}}, \bar{c}, \bar{\mathrm{Y}})$, il envoie la valeur $\bar{\mathrm{M}}$ à $\tilde{\mathcal{V}}$ et $\tilde{\mathcal{V}}$ renvoie une valeur $c = \mathrm{C}(\bar{\mathrm{W}}, \omega_{\tilde{\mathcal{V}}}, hist)$. Il nous faut maintenant calculer la probabilité que la valeur c choisie par $\tilde{\mathcal{V}}$ soit égale à la valeur $\bar{c}$ choisie par M. Nous notons cette probabilité

$$\Pr_{\bar{\mathrm{Y}}\in\mathbb{G},\bar{c}\in[0,2^k[,\omega_{\tilde{\mathcal{V}}}}(\bar{c} = \big(\mathrm{C}(e(g, \bar{\mathrm{Y}}pk_{\mathcal{P}}^{-\bar{c}}), \omega_{\tilde{\mathcal{V}}}, hist)\big).$$

Elle est égale à :

$$\begin{aligned}
& \frac{1}{q} \times \frac{1}{2^k} \times \sum_{\bar{c}\in[0,2^k[} \sum_{\bar{\mathrm{Y}}\in\mathbb{G}} \Pr_{\omega_{\tilde{\mathcal{V}}}}(\bar{c} = \big(\mathrm{C}(e(g, \bar{\mathrm{Y}}pk_{\mathcal{P}}^{-\bar{c}}), \omega_{\tilde{\mathcal{V}}}, hist)\big) \\
= \quad & \frac{1}{q} \times \frac{1}{2^k} \times \sum_{\bar{c}\in[0,2^k[} \sum_{\bar{\mathrm{Y}}\in\mathbb{G}} \Pr_{\omega_{\tilde{\mathcal{V}}}}(\bar{c} = \big(\mathrm{C}(e(g, \bar{\mathrm{Y}}), \omega_{\tilde{\mathcal{V}}}, hist)\big) \\
= \quad & \frac{1}{q} \times \frac{1}{2^k} \times \sum_{\bar{\mathrm{Y}}\in\mathbb{G}} \sum_{\bar{c}\in[0,2^k[} \Pr_{\omega_{\tilde{\mathcal{V}}}}(\bar{c} = \big(\mathrm{C}(e(g, \bar{\mathrm{Y}}), \omega_{\tilde{\mathcal{V}}}, hist)\big) \\
= \quad & \frac{1}{q} \times \frac{1}{2^k} \times \sum_{\bar{\mathrm{Y}}\in\mathbb{G}} 1 \\
= \quad & 1/2^k
\end{aligned}$$

Par conséquent, la valeur choisie par $\tilde{\mathcal{V}}$ est égale à $\bar{c}$ avec probabilité $1/2^k$. En cas de non-égalité, M recommence la procédure en construisant un nouveau triplet.

Il faut que M calcule en moyenne 2^k triplets pour obtenir l'égalité entre c et $\bar{c}$. Comme notre protocole se déroule en l tours, il faut itérer ce procédé l fois. La complexité en temps de M est alors $\mathcal{O}(l \times 2^k)$, ce qui est polynomial en $|\mathbb{G}|$ si $l \times 2^k$ est polynomial en $|\mathbb{G}|$.

La distribution des triplets $(\bar{\mathrm{W}}, \bar{c}, \bar{\mathrm{Y}})$, donnée par $\Pr_{\omega_{\mathrm{M}},\omega_{\tilde{\mathcal{V}}}}\big((\bar{\mathrm{W}}, \bar{c}, \bar{\mathrm{Y}}) = (\mathrm{W}_0, c_0, \mathrm{Y}_0)\big)$ s'écrit de la manière suivante :

$$\begin{aligned}
& \Pr_{\omega_{\mathrm{M}},\omega_{\tilde{\mathcal{V}}}}\big((\bar{\mathrm{W}}, \bar{c}, \bar{\mathrm{Y}}) = (\mathrm{W}_0, c_0, \mathrm{Y}_0)\big) \\
= \quad & \Pr_{\bar{\mathrm{Y}}\in\mathbb{G},\bar{c}\in[0,2^k[,\omega_{\tilde{\mathcal{V}}}}\big(e(g, \bar{\mathrm{Y}})pk_{\mathcal{P}}^{-\bar{c}} = \mathrm{W}_0 \wedge \bar{c} = c_0 \wedge \bar{\mathrm{Y}} = \mathrm{Y}_0 \big| \bar{c} = \mathrm{C}(e(g, \bar{\mathrm{Y}})pk_{\mathcal{P}}^{-\bar{c}}, \omega_{\tilde{\mathcal{V}}}, hist)\big) \\
= \quad & \frac{\Pr_{\bar{\mathrm{Y}}\in\mathbb{G},\bar{c}\in[0,2^k[,\omega_{\tilde{\mathcal{V}}}}\big(e(g, \bar{\mathrm{Y}})pk_{\mathcal{P}}^{-\bar{c}} = \mathrm{W}_0 \wedge \bar{c} = c_0 \wedge \bar{\mathrm{Y}} = \mathrm{Y}_0\big)}{\Pr_{\bar{\mathrm{Y}}\in\mathbb{G},\bar{c}\in[0,2^k[,\omega_{\tilde{\mathcal{V}}}}\big(\bar{c} = \mathrm{C}(e(g, \bar{\mathrm{Y}})pk_{\mathcal{P}}^{-\bar{c}}, \omega_{\tilde{\mathcal{V}}}, hist)\big)}
\end{aligned}$$

Le numérateur de l'équation étant égal à :

$$\frac{1}{2^k} \times \frac{1}{q} \times \Pr_{\omega_{\tilde{\mathcal{V}}}}\big(e(g, \mathrm{Y}_0)pk_{\mathcal{P}}^{-c_0} = \mathrm{W}_0 \wedge c_0 = \mathrm{C}(\mathrm{W}_0, \omega_{\tilde{\mathcal{V}}}, hist)\big)$$

nous obtenons finalement

$$\Pr_{\omega_{\mathcal{P}},\omega_{\tilde{\mathcal{V}}}}\left((\mathrm{W},c,\mathrm{Y})=(\mathrm{W}_0,c_0,\mathrm{Y}_\mathrm{O})\right)=$$
$$\begin{cases}\frac{1}{q}\times\Pr_{\omega_{\tilde{\mathcal{V}}}}\left(c_0=(\mathrm{C}(\mathrm{W}_0,\omega_{\tilde{\mathcal{V}}},hist)\right) & \text{si } \mathrm{W}_0=e(g,\mathrm{Y}_0)pk_{\mathcal{P}}^{-c_0}\\ 0 & \text{sinon}\end{cases}$$

Les distributions de probabilité des communications réelles et simulées sont identiques et par conséquent, le schéma HLS est parfaitement à divulgation nulle de connaissance. □

6.4 Conversion en preuve du OU et signature d'anneau

De par sa construction, le schéma précédent peut être transformé facilement en schéma d'identification d'une personne parmi deux, ce que l'on peut considérer comme une preuve du OU.

La clé privée du prouveur dans le schéma est de la forme g^a et la clé publique associée est $e(g,g)^a$. Supposons maintenant que deux utilisateurs possèdent chacun un couple $(s,\, g^s)$ où s est une clé privée et g^s la clé publique associée. Nous pouvons alors, à l'aide de notre schéma identifier une personne parmi deux, sans que le vérificateur ne puisse décider qui des deux personnes est réellement en train de s'authentifier.

Par exemple, considérons l'utilisateur Alice dont les clés sont $(a,\, g^a)$ et l'utilisateur Bob, dont les clés sont $(b,\, g^b)$. Plaçons nous dans le cas où Alice souhaite s'authentifier anonymement (le protocole se déroule de manière symétrique si Bob souhaite s'authentifier). Elle connaît la clé publique de Bob et est capable à l'aide de sa clé secrète a de calculer une nouvelle clé privée g^{ab}. Ce calcule peut aussi être fait interactivement entre Bob et Alice à l'aide du protocole d'échange de clé Diffie-Hellman [DH76]. Les paramètres publics de l'ensemble $\{\mathrm{A}lice,\mathrm{B}ob\}$ sont alors essentiellement les mêmes que ceux de notre schéma. L'utilisation de l'application bilinéaire e permet de compléter les paramètres et on obtient $params=(g,\, g^a,\, g^b,\, e(g,g),\, pk_{\mathcal{P}})$, où $pk_{\mathcal{P}}=e(g,g)^{ab}$ se calcule aisément à partir des deux clés publiques d'Alice et Bob.

On remarque aussi que la clé privée associée à ces paramètres est de la même forme que la clé privée du schéma précédent, $sk_{\mathcal{P}}=g^{ab}$ et cette clé secrète peut être calculée uniquement par Alice et Bob. Ainsi, après avoir calculé cette clé $sk_{\mathcal{P}}$, Alice et Bob peuvent s'identifier auprès du vérificateur de notre schéma d'authentification, sans que celui-ci puisse déterminer qui de Alice ou Bob, est en train d'exécuter le protocole. Le schéma devient ainsi une preuve d'identification à divulgation nulle de connaissance d'un prouveur $\mathcal{P}$ appartenant à l'ensemble $\{\mathrm{A}lice,\mathrm{B}ob\}$.

6.4.1 Description du schéma

Nous décrivons maintenant le déroulement du protocole, en supposant que le prouveur $\mathcal{P}$ est soit Alice soit Bob.

1. Dans une première phase d'initialisation, le prouveur $\mathcal{P}$ (Alice ou Bob) calcule la clé secrète $sk_{\mathcal{P}}$,

2. $\mathcal{P}$ annonce ensuite au vérificateur l'ensemble $\{\mathrm{A}lice, \mathrm{B}ob\}$ dont il fait partie,
3. le vérificateur reçoit les clés publiques des deux membres (g^a, g^b) et peut ainsi calculer les paramètres publics $params = (g,\ g^a,\ g^b,\ e(g,g),\ pk_{\mathcal{P}})$, où $pk_{\mathcal{P}} = e(g,g)^{ab}$,
4. le protocole se déroule ensuite de la même manière que pour l'authentification d'une personne décrit dans la figure 6.1. Tout d'abord, $\mathcal{P}$ calcule son engagement $\mathrm{W} = e(g,g)^r$ et l'envoie au vérificateur. Celui-ci lui renvoie son challenge c et $\mathcal{P}$ renvoie sa réponse $\mathrm{Y} = g^r \times sk_{\mathcal{P}}^c$. Le vérificateur accepte $\mathcal{P}$ comme étant Alice ou Bob en vérifiant que $e(g,\mathrm{Y}) = \mathrm{W} \times pk_{\mathcal{P}}^c$.

Ce protocole se déroule de manière identique que ce soit Alice ou Bob qui cherche à s'authentifier. Seuls Alice et Bob sont capables de prouver s'ils sont à l'origine d'une authentification, après que celle-ci a eu lieu. Si Alice, par exemple, veut prouver qu'elle est bien à l'origine d'une identification, elle calcule la valeur $y = r + ac$ où a est sa clé secrète, c le challenge envoyé par le vérificateur pendant la dite session et r son aléa. Alice envoie la valeur au vérificateur et celui-ci teste si

$$e(g,g)^y = \mathrm{W} \times pk'^c_{\mathcal{P}}$$

où W est la première valeur envoyée par Alice durant la session et $pk'_{\mathcal{P}} = e(g,g^a)$. En cas d'égalité, le vérificateur sait de manière sûre que l'authentification provient bien d'Alice.

Notre schéma permet donc d'authentifier anonymement une personne parmi deux, de manière efficace. Ceci peut aussi être vu comme une preuve du OU, dans le sens où Alice (ou Bob), prouve qu'elle connaît le logarithme discret de la valeur g^a ou g^b aux yeux du vérificateur.

Ce schéma d'identification d'une personne parmi deux peut ensuite être transformé en schéma de signature d'anneau, où l'anneau est formé de deux entités, en utilisant l'heuristique de Fiat-Shamir [FS86]. Les groupes $\mathbb{G}$ et $\mathbb{G}_1$ sont choisis comme précédemment et H est une fonction de hachage telle que $\mathrm{H} : (\{0,1\}^* \times \mathbb{G} \times \mathbb{G}_1 \times \mathbb{G}_1 \to [0, 2^k]$. Nous utilisons toujours la même application bilinéaire e.

Supposons qu'Alice décide de signer un message $m \in \{0,1\}^*$ de son choix en utilisant l'anneau $\{\mathrm{A}lice, \mathrm{B}ob\}$. Dans une phase d'initialisation, Alice (*resp.* Bob) choisit sa clé privée a (*resp.* b), calcule sa clé publique $pk_{\mathrm{A}} = g^a$ (*resp.* $pk_{\mathrm{B}} = g^b$) et la publie. Afin de signer son message, Alice suit la procédure suivante :

1. Elle calcule la clé $sk = pk_{\mathrm{B}}^a$.
2. Elle choisit $r \in_{\mathrm{R}} \mathbb{Z}_q$ et calcule $\mathrm{W} = e(g,g)^r$.
3. À l'aide de la fonction de hachage, Alice calcule le challenge $c = \mathrm{H}(m, \mathrm{W}, g^a, g^b)$.
4. Elle calcule ensuite $\mathrm{Y} = g^r \times sk^c$.
5. La signature σ de m est le couple (W, Y).

Pour vérifier la signature, le destinataire a besoin des paramètres publics (pk_{A}, pk_{B}, g, e). En recevant (m, σ), le destinataire calcule $pk_{\mathcal{P}} = e(g^a, g^b)$ (cette valeur peut être ajoutée aux paramètres publics, comme pour le schéma d'identification). Le destinataire

calcule la valeur $c = \mathrm{H}(m, \mathrm{W}, g^a, g^b)$ et vérifie que la valeur Y appartient bien à $\mathbb{Z}_p$. La validité de la signature est donnée par la même équation que précédemment :

$$e(g, \mathrm{Y}) = \mathrm{W} \times pk_{\mathcal{P}}^{c}$$

La sécurité de ce schéma contre des attaques à message connus adaptatives se montre en utilisant le *lemme de bifurcation* (*cf.* section 2.2.3).

6.4.2 Efficacité et utilisations

Le schéma d'identification que nous avons proposé est, à notre connaissance, le seul schéma d'identification basé sur l'utilisation des couplages, possédant la propriété de divulgation nulle de connaissance.

Shao, Lu et Cao ont proposé dans [SLC04], un protocole basé sur le schéma de signature de Boneh et Boyen [BB04], mais celui-ci ne peut être prouvé à divulgation nulle de connaissance, même face à un vérificateur honnête. En effet, il n'est pas possible de construire d'algorithme simulant les conversations sans connaître la clé secrète. Kim et Kim ont aussi proposé un protocole d'identification utilisant les couplages [KK02a]. Ce schéma possédant une faille de sécurité, il a été réparé une première fois par Yao, Wang et Wang [YWW03], puis par Kim et Kim [KK02b] eux-mêmes. Cependant, ces nouvelles constructions ne sont pas non plus satisfaisantes et la propriété de divulgation nulle de connaissance face à des vérificateurs honnêtes semble difficile à atteindre.

Du point de vue de l'efficacité, le schéma présenté ici est au moins aussi performant que celui de Shao, Lu et Cao et nettement plus efficace que ceux de Kim et Kim, et Yao, Wang et Wang. La comparaison de l'efficacité est faite sur une passe du protocole. Nos preuves de sécurité restent valides, même en cas d'une seule itération tant que $1/2^k$ reste négligeable.

Le schéma de signature d'anneau qui résulte de cette identification est lui aussi très efficace et très simple d'utilisation. Il est cependant difficile de le comparer à l'existant car il ne concerne que des anneaux composés de deux personnes.

Troisième partie

Signatures aveugles à anonymat révocable

Chapitre 7

Définitions et état de l'art

Nous avons étudié, dans la partie précédente, plusieurs types de signatures utilisées à des fins d'anonymat en cryptographie. S'il est important pour diverses applications, de garantir à l'utilisateur que certaines données qu'il transmet (son identité ou son message par exemple) seront protégées, il est aussi nécessaire de se prémunir contre les utilisateurs mal intentionnés. Dans le cas des signatures de groupe, la mise en place d'une autorité de révocation permet à un juge mandaté de lever l'anonymat sur une signature suspecte et ainsi de retrouver la personne à l'origine de cette signature.

Dans le cas des signatures aveugles, il est, par définition, impossible de relier une signature à un utilisateur ou une exécution du protocole à une signature. Comme nous l'avons vu, les signatures aveugles trouvent des applications dans des domaines très sensibles comme le vote électronique ou la monnaie électronique. Afin d'en améliorer la sécurité il peut être nécessaire d'inclure une levée d'anonymat pour protéger les participants, que ce soit les utilisateurs ou les signataires, contre des utilisations frauduleuses de tels systèmes.

À *Eurocrypt'95*, Stadler, Piveteau et Camenisch ont donné une solution à la levée d'anonymat dans les signatures aveugles [SPC95], en introduisant les signatures aveugles à anonymat révocable.

Ce chapitre définit cette variante des signatures aveugles. Nous présentons un état de l'art des travaux effectués depuis [SPC95]. Nous nous intéressons plus particulièrement à l'article d'Abe et Ohkubo [AO01] ainsi qu'à sa sécurité et mettons en avant une faille dans l'une des preuves de sécurité. Cette faille a été décrite dans [HT07] et présentée à la conférence *Pairing 2007*.

Sommaire

7.1 Définition

Une signature aveugle, comme nous l'avons décrite dans le chapitre 5, permet à un utilisateur de faire signer le message de son choix par un signataire, sans que ce dernier ait connaissance de son contenu. Afin de satisfaire la propriété d'indistinguabilité, il ne doit pas être possible, connaissant un message signé, de retrouver l'utilisateur qui a demandé cette signature. Il ne doit pas non plus être possible, à la vue des données échangées lors d'une exécution du protocole de signature, de retrouver la signature finale émise. Si cet anonymat peut-être utilisé à des fins tout à fait légitimes et dans des contextes spécifiques (comme le vote électronique ou la monnaie électronique), il est très vite apparu que les signatures aveugles pouvaient aussi servir à des personnes malintentionnées. Dans [vSN92], von Solms et Naccache ont montré comment utiliser ces signatures pour blanchir de l'argent.

En cas de doute sur une signature, il est donc important de pouvoir retrouver l'identité de l'utilisateur qui en est à l'origine. De même, afin de pouvoir tracer toutes les signatures émises par une personne donnée (quand elle est suspectée de fraude, par exemple), il est important de pouvoir relier les transcriptions des exécutions aux signatures.

C'est dans cette optique que Stadler, Piveteau et Camenisch ont présenté à *Eurocrypt'95* une nouvelle variante des signatures aveugles, permettant de lever l'anonymat : les signatures aveugles à anonymat révocable (aussi appelées signatures aveugles équitables). Nous utiliserons par la suite, l'abréviation FBS : *Fair Blind Signatures* pour désigner ces signatures.

Cette signature fait appel aux mêmes participants qu'une signature aveugle, à savoir des utilisateurs et un signataire. Afin de lever l'anonymat Stadler *et al.* ont introduit une autorité de confiance, appelée l'autorité de révocation. Cette autorité est capable de lever l'anonymat de deux manières différentes, définissant ainsi deux notions de traçabilité :

- le *traçage de signature* : étant donnée une transcription d'une exécution du protocole entre un utilisateur authentifié et le signataire, c'est-à-dire les données échangées pendant cette exécution, l'autorité de révocation est capable d'identifier la signature résultante.
- le *traçage de session ou d'identité* : étant donnée une signature, l'autorité de révocation est capable de retrouver l'exécution du protocole qui a conduit à cette signature ou, selon la description du protocole, de retrouver l'identité même de l'utilisateur.

L'autorité de révocation peut être placée sous le contrôle d'un juge. Dans ce cas, la levée d'anonymat n'est considérée comme valide que si l'autorité apporte au juge une preuve de ses affirmations.

7.1.1 Sécurité

Les FBS étant des signatures aveugles, elles doivent garantir au moins les propriétés de sécurité de celles-ci. Elles doivent donc être indistinguables et inforgeables. Cependant, la levée d'anonymat nécessite de se prémunir contre de nouvelles attaques. D'une part, il doit être impossible de produire une signature n'étant liée à aucune identité existante et d'autre part, toute session acceptée par le signataire et l'utilisateur doit être reliable à une unique signature. De plus, un adversaire ne doit pas pouvoir attribuer à un utilisateur honnête une signature ou une session à laquelle ce dernier n'aurait pas participé.

Nous verrons plus en détail, dans le chapitre 8, les définitions précises des ces propriétés.

7.1.2 Usages

Les signatures aveugles à anonymat révocable sont généralement utilisées dans les mêmes applications que les signatures aveugles, à savoir le vote ou la monnaie électroniques. En effet, elles offrent les mêmes garanties de sécurité tout en assurant une protection contre les fraudes ou les défaillances du système.

Dans le cadre de la monnaie électronique par exemple, une telle signature permettrait de tracer l'argent retiré par un utilisateur suspecté par les autorités. De même, il serait possible de retrouver l'utilisateur à l'origine d'une transaction suspecte comme dans le cas d'un blanchiment d'argent.

Ces signatures peuvent aussi être utilisées pour les enchères en ligne ou les appels d'offre anonymes de la manière suivante : supposons qu'un utilisateur souhaite enchérir pour l'acquisition d'un bien sans que son identité soit révélée aux autres enchérisseurs s'il ne remporte pas l'enchère. Il va demander à un signataire (qui serait en quelque sorte un commissaire-priseur) de valider son enchère en la faisant signer de manière aveugle. Il peut ensuite publier le montant ainsi authentifié afin que les autres enchérisseurs voient l'évolution des enchères. Quand la période des enchères est finie, l'autorité qui les régit est capable de retrouver le meilleur enchérisseur grâce à sa signature. Ainsi, seule l'identité de l'acquéreur est rendue publique.

7.1.3 Algorithmes de signatures aveugles à anonymat révocable

La construction d'une FBS suit le même déroulement général que celle d'une signature aveugle. Il est cependant nécessaire d'ajouter de nouveaux algorithmes pour la levée d'anonymat. Tout d'abord, la procédure de Setup comprend un nouvel algorithme pour construire les clés de l'autorité de révocation, et l'utilisateur doit, pour obtenir ses clés, posséder une identité publique, notée $\mathrm{Id}_{\mathcal{U}}$. Cette identité peut lui tenir lieu de clé publique.

Le système comprend ensuite une nouvelle procédure pour la révocation en elle-même. L'autorité de révocation a à sa disposition deux algorithmes pour lever l'anonymat :

- l'algorithme $\mathsf{R}_{\mathsf{sig}}$ qui permet de tracer une signature. Il prend en entrée le transcript d'une session et la clé privée de l'autorité, et renvoie un identifiant qui permet de

rattacher la session à une signature, ainsi qu'une preuve que la levée s'est effectuée honnêtement.

- l'algorithme $\mathsf{R_{id}}$ qui permet de tracer l'identité. Il prend en entrée un couple message/signature et la clé privée de l'autorité, et il renvoie un identifiant qui permet de relier la signature à une identité, ainsi qu'une preuve que la levée s'est effectuée honnêtement.

Les identifiants et les preuves sont ensuite transmis au juge qui, grâce à des algorithmes de comparaison, est capable de retrouver l'utilisateur ou la signature visé et de vérifier que la preuve est correcte ; autrement dit, il vérifie que l'autorité a effectué correctement la levée d'anonymat.

De manière plus formelle, les algorithmes sont décrits de la manière suivante :

Définition 88 (Construction des signatures aveugles à anonymat révocable)**.** *Un schéma de signature aveugle à anonymat révocable se construit à l'aide des procédures suivantes :*

- Setup*, phase de mise en place des paramètres du système, des paramètres publics et de construction des clés pour les différents participants. Les clés sont données par les algorithmes suivants :*
  - $\mathsf{G}_{\mathcal{S}}$*, algorithme de génération de clés pour le signataire. Il prend en entrée un paramètre de sécurité k et renvoie une clé privée $sk_{\mathcal{S}}$ et la clé publique associée $pk_{\mathcal{S}}$ pour le signataire.*
    $(sk_{\mathcal{S}}, pk_{\mathcal{S}}) \leftarrow \mathcal{G}_{\mathcal{S}}(1^k)$.
  - G_{RA}*, algorithme de génération de clés pour l'autorité de révocation. Il prend en entrée le paramètre de sécurité k et renvoie une clé privée sk_{RA} et la clé publique associée pk_{RA} pour l'autorité de révocation.*
    $(sk_{\mathrm{RA}}, pk_{\mathrm{RA}}) \leftarrow \mathcal{G}_{\mathrm{R}a}(1^k)$.
  - $\mathsf{G}_{\mathcal{U}}$*, algorithme de génération de clés pour un utilisateur. Il prend en entrée le paramètre de sécurité k, et l'identité $\mathrm{Id}_{\mathcal{U}}$ de l'utilisateur. Il renvoie une clé privée $sk_{\mathrm{Id}_{\mathcal{U}}}$ et la clé publique associée $pk_{\mathrm{Id}_{\mathcal{U}}}$ pour l'utilisateur.*
    $(sk_{\mathrm{Id}_{\mathcal{U}}}, pk_{\mathrm{Id}_{\mathcal{U}}}) \leftarrow \mathcal{G}_{\mathcal{U}}(1^k, \mathrm{Id}_{\mathcal{U}})$.
- BSign*, protocole interactif de signature aveugle entre le signataire $\mathcal{S}$ et l'utilisateur $\mathcal{U}$, représentés par des machines de Turing interactives. Le signataire prend en entrée sa clé privée $sk_{\mathcal{S}}$ et l'utilisateur le message m de son choix et son identité $\mathrm{Id}_{\mathcal{U}}$. Si les deux parties acceptent le protocole, le signataire garde un transcript $trans_{\mathrm{Id}_{\mathcal{U}}}$ du protocole et $\mathcal{U}$ est en possession d'une signature aveugle σ de son message m. Sinon, le protocole échoue.*
  $((m, \sigma), trans) \leftarrow \mathsf{Bsign}(\mathcal{U}(m, \mathrm{Id}_{\mathcal{U}}), \mathcal{S}(sk_{\mathcal{S}}))$.
- Verify*, algorithme de vérification de validité. Il prend en entrée un message m et une signature σ, et renvoie 1 si la signature est valide, 0 sinon.*

- Revoc*, procédure permettant à l'autorité de révocation et au juge de lever l'anonymat de deux manières différentes. Cette procédure est composée de quatre algorithmes :*
  - $\mathsf{R_{sig}}$*, algorithme de révocation de signature. Il prend en entrée un transcript $trans$ d'une exécution du protocole, la clé privée de l'autorité de révocation sk_{RA} et renvoie un identifiant de signature I_{sig} ainsi qu'une preuve de validité π_1.*

$(\mathrm{I}_{sig}, \pi_1) \leftarrow \mathsf{R}_{\mathsf{sig}}(trans, sk_{\mathrm{RA}})$.

- R_{id}, *algorithme de révocation d'identité. Il prend en entrée un couple message/signature* (m, σ) *valide, la clé privée de l'autorité de révocation* sk_{RA} *et renvoie un identifiant de session* I_{id} *ainsi qu'une preuve de validité* π_2.
  $(\mathrm{I}_{id}, \pi_2) \leftarrow \mathsf{R}_{\mathsf{id}}(m, \sigma, sk_{\mathrm{RA}})$.
- $\mathsf{Match}_{\mathsf{sig}}$, *algorithme de comparaison qui prend en entrée un identifiant de signature* I_{sig}, *un couple message/signature* (m, σ) *et une preuve* π_1. *Il renvoie* 1 *si l'identifiant correspond au couple message/signature et si la preuve est valide. Sinon, il renvoie 0.*
- $\mathsf{Match}_{\mathsf{id}}$, *algorithme de comparaison qui prend en entrée un identifiant de session* I_{id}, *un transcript de session* $trans$ *(ou une identité) et une preuve* π_2. *Il renvoie* 1 *si l'identifiant correspond à la session (ou à l'identité) et si la preuve est valide. Sinon, il renvoie 0.*

On spécifie alors un schéma de signature aveugle à anonymat révocable comme l'ensemble : FBS = $(\mathsf{G}_{\mathcal{S}}, \mathsf{G}_{\mathrm{RA}}, \mathsf{G}_{\mathcal{U}}, \mathsf{BSign}, \mathsf{Verify}, \mathsf{R}_{\mathsf{sig}}, \mathsf{R}_{\mathsf{id}}, \mathsf{Match}_{\mathsf{sig}}, \mathsf{Match}_{\mathsf{id}})$.

7.2 État de l'art

Dans [SPC95], Stadler *et al.* ont proposé la première définition de signature aveugle à anonymat révocable. Celle-ci ne fait pas intervenir de juge et les signatures sont divisées en deux types, selon la révocation autorisée. Le traçage de signature donne une signature de type I, le traçage d'identité une signature de type II. Les trois schémas qu'ils présentent ne proposent pas nécessairement les deux révocations. Le premier protocole s'appuie sur la méthode du *cut and choose* introduite par Chaum *et al.* pour la monnaie électronique [CFN88]. Le schéma résultant est en fait inefficace. C'est pourquoi, ils proposent un autre schéma utilisant la signature de Fiat-Shamir [FS86] et le concept d'*oblivious transfer* introduit dans [EGL85]. Ce schéma n'autorisant qu'une révocation de type I, Stadler *et al.* présentent une troisième signature offrant la double révocation. Dans leur article, les auteurs ne se sont pas penchés sur la sécurité des schémas. D'ailleurs, le deuxième schéma sera prouvé non sûr [Tra97]. La même année, Brickell, Gemmel et Kravitz ont eux aussi proposé un protocole de signature aveugle à anonymat révocable [BGK95], mais il s'avère que leur construction est inefficace.

Par la suite, d'autres schémas de signature aveugle à anonymat révocable ont été présentés dans le cadre d'applications précises, comme la monnaie électronique ([CMS96], [FTY96], [dST98], [GT03]) ou le vote électronique [CGT06].

Le schéma de FBS le plus significatif est celui d'Abe et Ohkubo décrit dans [AO01]. Cet article s'attache à définir les principales propriétés de sécurité des schémas FBS et propose un schéma prétendu sûr sous l'hypothèse Diffie-Hellman décisionnelle et l'hypothèse du logarithme discret, dans le modèle de l'oracle aléatoire.

[AO01] a été le premier article à formaliser les propriétés de sécurité des signatures aveugles à anonymat révocable et à détailler les preuves de sécurité. Cependant, Abe et Ohkubo ne parviennent pas à prouver la sécurité polynomiale qu'ils prétendent dans leur théorème. Nous allons dans la suite de ce chapitre décrire ce schéma et en étudier sa

sécurité en mettant en avant une faille dans la preuve de l'inforgeabilité. Cette attaque résulte d'un travail commun avec Jacques Traoré et a été présentée à la conférence *Pairing 2007*.

7.2.1 Schéma d'Abe et Ohkubo

Setup

Soient p et q deux nombres premiers tels que $q|p-1$ et soient g et h, deux générateurs d'un sous-groupe d'ordre q dans $\mathbb{Z}_p^*$. Le signataire et l'autorité de révocation font appel à leur algorithme de génération de clés respectifs :

$\mathsf{G}_{\mathcal{S}}$ **:** le signataire choisit trois fonctions de hachage
- $\mathrm{H}_1 : \{0,1\}^* \rightarrow \langle g \rangle$,
- $\mathrm{H}_2 : \{0,1\} \rightarrow \{0,1\}^{|q|}$,
- $\mathrm{H}_3 : \{0,1\} \rightarrow \{0,1\}^{|q|}$.

Sa clé publique est alors $pk_{\mathcal{S}} = (p, q, g, h, y, z)$ et sa clé privée $sk_{\mathcal{S}} = x$ telle que :
- $x \in \mathbb{Z}_q$,
- $y = g^x \mod p$,
- $z = \mathrm{H}_1(p, q, g, h, y)$.

Dans la suite, la réduction modulo p sera implicite.

G_{RA} **:** l'autorité de révocation génère ses clés privées et publiques : $sk_{\mathrm{RA}} = (x_t, dk)$ et $pk_{\mathrm{RA}} = (y_t, ek)$ où $x_t \in \mathbb{Z}_q^*$, $y_t = g^{x_t}$ et ek et dk sont des clés pour un schéma de chiffrement vérifiable noté $\mathcal{E}_{ek}$ (algorithme de chiffrement) et $\mathcal{D}_{dk}$ (algorithme de déchiffrement).

Dans cette construction, l'utilisateur utilise un nouvel identifiant pour chaque signature et ne possède pas de clés propres.

Les paramètres publics sont : $params = (p, q, g, h, y, z, \mathrm{H}_1, \mathrm{H}_2, \mathrm{H}_3, y_t, ek)$.

BSign

Nous décrivons maintenant le schéma, en incluant les éléments nécessaires à la révocation d'anonymat. Ce schéma est basé sur le schéma de signature aveugle proposé par Abe dans [Abe01]. Une première étape est ajoutée à ce schéma afin que l'utilisateur puisse s'engager sur un secret qui lui permettra d'être tracé. La suite du protocole reste semblable. Nous donnons ci-dessous une description rapide, le détail des calculs se trouvant dans la figure 7.1.

1. L'utilisateur $\mathcal{U}$ choisit un facteur d'aveuglement $\gamma \in \mathbb{Z}_q^*$ et calcule $z_u = z^{1/\gamma}$ et $\xi = g^{\gamma}$. Il chiffre ensuite la valeur γ en E à l'aide du chiffrement vérifiable choisi et prouve les relations entre z_u, ξ, E dans une preuve de connaissance notée P. Ces valeurs seront nécessaires pour tracer la signature finale par la suite.
2. Le signataire $\mathcal{S}$ vérifie la preuve P. Il génère ensuite v et calcule $z_1 = y_t^v$ et $z_2 = z_u/z_1$. Il prouve ensuite que z_1 est bien formé à l'aide d'une preuve de connaissance à divulgation nulle de connaissance face à un vérificateur honnête.
3. Le signataire et l'utilisateur s'engagent ensuite dans une preuve interactive de connaissance à témoin indistinguable. Pour le signataire, le protocole est une

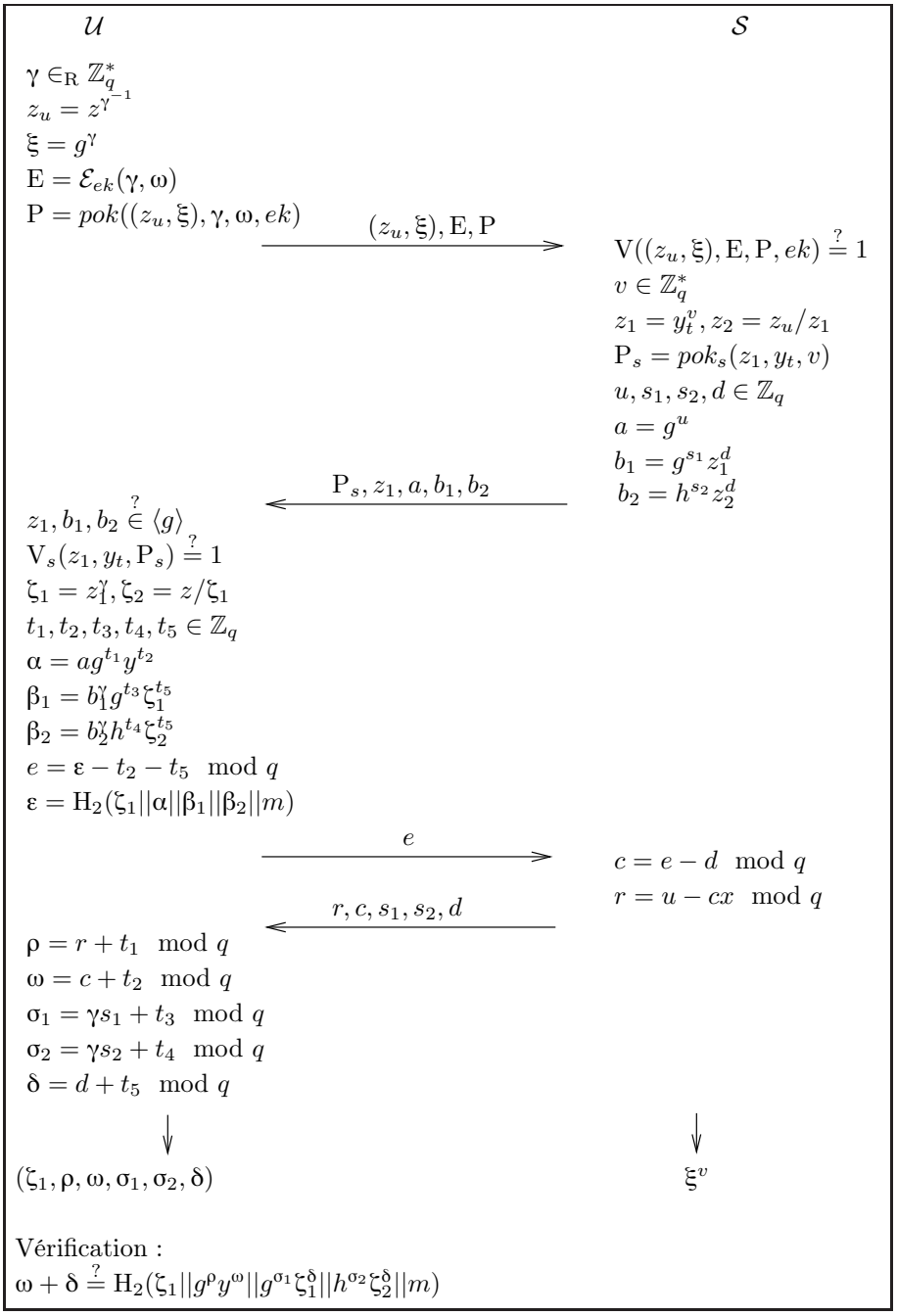

FIG. 7.1 – Signature aveugle à anonymat révocable d'Abe et Ohkubo

preuve de connaissance de

$$\log_g y \vee (\log_g z_1 \wedge \log_h (z_u / z_1)).$$

Le signataire convertit cette preuve en une preuve de connaissance de

$$\log_g y \vee (\log_g \zeta_1 \wedge \log_h(z_u/\zeta_1))$$

en élevant (z, z_1) à la puissance $\gamma : (z_1, z_u) \xrightarrow{\gamma} (\zeta_1, z)$ et en l'« aveuglant » grâce à la technique de *diversion* de [OO89b]. La preuve peut finalement être transformée en signature en utilisant l'heuristique de Fiat-Shamir.

4. Le signataire enregistre la valeur ξ^v comme l'identifiant de la session.
5. L'utilisateur calcule sa signature $\Sigma = (\zeta_1, \rho, \omega, \sigma_1, \sigma_2, \delta)$ pour son message m.

Verify

Le couple message/signature (Σ, m) est valide s'il vérifie

$$\omega + \delta = \mathrm{H}_2(\zeta_1 || g^\rho y^\omega || g^\sigma \zeta_1^\delta || h^{\sigma_2}(z/\zeta_1)^\delta || m) \mod q.$$

Revoc

Selon la révocation voulue, l'autorité va procéder de manière différente et utiliser soit l'algorithme de traçage de signature $\mathsf{R_{sig}}$ soit de traçage d'identité $\mathsf{R_{id}}$. La description du protocole ne fait pas intervenir de juge. L'autorité de révocation est la seule personne qui entre en jeu lorsqu'une levée d'anonymat est demandée. Elle ne produit donc pas de preuve sur le déroulement de sa levée d'anonymat. Il n'y a donc pas d'algorithme de comparaison $\mathsf{Match_{sig}}$ et $\mathsf{Match_{id}}$.

1. $\mathsf{R_{sig}}$: l'autorité reçoit $(z_u, \xi, \mathrm{E}, \mathrm{P})$, les valeurs échangées pendant une exécution du protocole. Elle calcule ensuite $\mathrm{I}_{sig} = (\xi^v)^{x_t}$. On remarque que

$$\mathrm{I}_{sig} = (\xi^v)^{x_t} = g^{\gamma v x_t} = y_t^{\gamma v} = \zeta_1.$$

   I_{sig} permet donc d'identifier de manière unique la signature résultant des interactions ayant produit $(z_u, \xi, \mathrm{E}, \mathrm{P})$.

2. $\mathsf{R_{id}}$: l'autorité reçoit un couple message/signature valide et calcule $\mathrm{I}_{id} = \zeta_1^{1/x_t}$. On remarque que

$$\mathrm{I}_{id} = \zeta_1^{1/x_t} = z_1^{\gamma/x_t} = y_t^{v\gamma/x_t} = g^{v\gamma} = \xi^v.$$

   L'autorité de révocation retrouve la valeur ξ^v que le signataire a gardée en fin de protocole.

7.2.2 Sécurité du schéma d'Abe et Ohkubo

Nous allons maintenant nous intéresser à la sécurité du schéma d'Abe et Ohkubo. Nous donnons tout d'abord les propriétés de sécurité telles qu'introduites dans [AO01]. Nous nous attachons tout particulièrement à la preuve d'inforgeabilité et montrons qu'Abe et Ohkubo ne parviennent pas à établir la sécurité qu'ils prétendent.

Définition 89 (Traçage de signature - [AO01])**.** *Soit $\mathcal{U}^*$ un algorithme probabiliste s'exécutant en temps polynomial. Si, après avoir interagi au plus l fois de manière adaptative et concurrente avec un signataire honnête, $\mathcal{U}^*$ renvoie*

- *soit un couple message/signature* Σ_m *valide tel que si on pose* $\mathrm{I}_{sig}^i = \mathsf{R}_{\mathsf{sig}}(view_i, sk_{\mathrm{RA}})$ *alors* $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}^i, \Sigma_m) = 0$ *pour tout* $i = 1, \ldots, l$,
- *soit deux couples message/signature valides et différents* Σ_{m_0} *et* Σ_{m_1} *tels qu'il existe* $i \in \{1, \ldots, l\}$ *vérifiant* $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, \Sigma_{m_0}) = \mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, \Sigma_{m_1}) = 1$, *avec* $\mathrm{I}_{sig} = \mathsf{R}_{\mathsf{sig}}(view_i, sk_{\mathrm{RA}})$,

avec probabilité au moins $1/n^c$ *pour* n *suffisamment grand et* c *une constante, alors le schéma de signature aveugle à anonymat révocable est* traçable pour la signature.

Définition 90 (Traçage de session - [AO01]). *Soit* $\mathcal{U}^*$ *un algorithme probabiliste s'exécutant en temps polynomial. Si, après avoir interagi au plus* l *fois, de manière adaptative et concurrente avec un signataire honnête,* $\mathcal{U}^*$ *renvoie un couple message/signature* Σ_m *valide, satisfaisant*

- *soit* $\mathsf{Match}_{\mathsf{id}}(\mathrm{I}_{id}, view_i) = 0$ *pour tout* $i \in \{1, \ldots, l\}$ *où* $\mathrm{I}_{id} = \mathsf{R}_{\mathsf{id}}(\Sigma_m, sk_{\mathrm{RA}})$,
- *soit* $\mathsf{Match}_{\mathsf{id}}(\mathrm{I}_{id}, view_i) = \mathsf{Match}_{\mathsf{id}}(\mathrm{I}_{id}, view_j) = 1$ *pour deux entiers* i *et* $j \in \{1, \ldots, n\}$ *avec* $i \neq j$,

avec probabilité au moins $1/n^c$ *pour* n *suffisamment grand et* c *une constante, alors le schéma de signature aveugle à anonymat révocable est* traçable pour la session.

Les définitions suivantes concernent l'indistinguabilité et l'inforgeabilité. Il s'agit en fait des propriétés de sécurité requises pour tout schéma de signature aveugle. Ces définitions rejoignent donc celles que nous avons données dans le chapitre 5. Nous les rappelons ici, en utilisant le formalisme et les notations d'Abe et Ohkubo.

Définition 91 (Indistinguabilité - [AO01]). *Soient* $\mathcal{S}^*$ *et* $\mathcal{D}^*$ *deux algorithmes probabilistes fonctionnant en temps polynomial,= jouant le jeu suivant avec les utilisateurs honnêtes* $\mathcal{U}_0$ *et* $\mathcal{U}_1$ *:*

1. $(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k)$, $(pk_{\mathrm{RA}}, sk_{\mathrm{RA}}) \leftarrow \mathsf{G}_{\mathrm{RA}}(1^k, pk_{\mathcal{S}})$,
2. $(m_0, m_1) \leftarrow \mathcal{S}^*(sk_{\mathcal{S}}, pk_{\mathrm{RA}})$,
3. *soit* $b \in_{\mathrm{R}} \{0, 1\}$ *;* m_b *est donné à* $\mathcal{U}_0$ *et* m_{1-b} *est donné à* $\mathcal{U}_1$,
4. $\mathcal{S}^*$ *s'engage dans des protocoles de signature avec* $\mathcal{U}_0$ *et* $\mathcal{U}_1$ *dans un ordre arbitraire,*
5. *la signature* Σ_0 *du message* m_0 *est donnée à* $\mathcal{D}^*$. $\mathcal{D}^*$ *a aussi le droit de recevoir toutes les informations que* $\mathcal{S}^*$ *a reçues au cours de ses interactions,*
6. D^* *renvoie un bit* $b' \in \{0, 1\}$.

   Le schéma de signature est indistinguable *si, pour tout* $\mathcal{S}^*$ *et* $\mathcal{D}^*$ *s'exécutant en temps polynomial, la probabilité que* $b = b'$ *est au plus* $1/2 + 1/k^c$, *pour tout* k *suffisamment grand et une constante* c. *Les probabilités sont prises sur les entrées des rubans aléatoires de* $\mathsf{G}_{\mathcal{S}}$, G_{RA}, $\mathcal{S}^*$, $\mathcal{D}^*$, $\mathcal{U}_0$, $\mathcal{U}_1$ *et* b.

Définition 92 (Non-falsification supplémentaire - [AO01]). *Un schéma de signature aveugle est* $(l, l+1)$-inforgeable, *si, pour tout algorithme probabiliste* $\mathcal{U}^*$ *s'exécutant en temps polynomial,* $\mathcal{U}^*$ *renvoie* $l+1$ *signatures valides, avec probabilité* $1/k^c$ *après au plus* l *interactions avec le signataire* $\mathcal{S}$, *pour tout* k *suffisamment grand et une constante* c. *Les probabilités sont prises sur les entrées des rubans aléatoires de* $\mathsf{G}_{\mathcal{S}}$, $\mathcal{S}$ *et* $\mathcal{U}^*$.

Il s'avère en fait, que la définition 92 est comprise dans la définition de traçabilité. En effet, la révocation telle qu'elle est présentée assure une correspondance unique entre les exécutions du protocole et les signatures émises. Il suffit donc de montrer que les propriétés d'indistinguabilité et de traçabilité sont satisfaites pour prouver la sécurité d'un schéma FBS.

Théorème 18 (Indistinguabilité - [AO01])**.** *Si le schéma de chiffrement vérifiable est sûr et simulable alors le schéma d'Abe et Ohkubo satisfait, sous l'hypothèse DDH et dans le modèle de l'oracle aléatoire, la propriété d'indistinguabilité.*

Nous ne donnons pas ici la preuve complète de l'indistinguabilité, celle-ci est décrite précisément dans [AO01].

Théorème 19 (Traçage de session - [AO01])**.** *Si le schéma de chiffrement vérifiable est sûr et significatif alors le schéma d'Abe et Ohkubo est, sous l'hypothèse DDH et dans le modèle de l'oracle aléatoire, traçable pour la session.*

Cette preuve de traçabilité inclut la preuve d'inforgeabilité du schéma. Dans un premier temps, nous rappelons la preuve donnée par Abe et Ohkubo, puis nous démontrons pourquoi le schéma ne satisfait pas la propriété d'inforgeabilité.

Démonstration. Abe et Ohkubo s'attachent à démontrer qu'il n'est pas possible de produire une signature $\Sigma^* = (\zeta_1, \rho, \omega, \sigma_1, \sigma_2, \delta)$ telle que $\log_{z_u} z \neq \log_{z_1} \zeta_1$ pour tous les couples (z_u, z_1) produits dans les sessions de signature.

La preuve se fait par contraposée. Supposons qu'après au plus q_{H} requêtes à H_2 et l requêtes de signature à $\mathcal{S}$, $\mathcal{U}_0^*$ renvoie une signature $\Sigma^* = (\zeta_1, \rho, \omega, \sigma_1, \sigma_2, \delta)$ telle que $\log_{z_u} z \neq \log_{z_1} \zeta_1$ pour tous les couples (z_u, z_1) produits dans les sessions de signature. On suppose que l et q_{H} sont polynomialement bornés par un paramètre n. On note ε la probabilité de succès de $\mathcal{U}_0^*$ qui est non négligeable (en n).

On tire au hasard $\mathrm{Q} \in \{1, \ldots, q_{\mathrm{H}}\}$. On suppose que $\mathcal{U}_0^*$ réussit son attaque si sa signature finale correspond à la Q-ème requête (autrement dit, $c = \mathrm{H}_2(\zeta || \ldots || m)$ correspond à la Q-ème requête). Dans le cas où $\mathcal{U}_0$ ne se serait pas adressé à l'oracle H_2 pour calculer $c = \mathrm{H}_2(\zeta_1 || \ldots || m)$, sa probabilité de succès serait négligeable. Cela revient à considérer un adversaire $\mathcal{A}$ qui fait une unique requête à H_2. Sa probabilité de succès est alors $\varepsilon_1 \geq \varepsilon_0 / q_{\mathrm{H}}$. L'adversaire $\mathcal{A}$ est utilisé pour construire une machine $\mathcal{M}$ qui résout le problème du logarithme discret. Soit (p, q, g, Y) une instance du problème du logarithme discret. Le but de $\mathcal{M}$ est de trouver X tel que $\mathrm{X} = \log_g \mathrm{Y}$ dans $\mathbb{Z}_q$.

$\mathcal{M}$ choisit lui-même les valeurs p, q et g. Il génère aussi les clés (ek, dk) pour le chiffrement vérifiable. Il choisit aléatoirement $\chi \in \{0, 1\}$ et pose $y := \mathrm{Y}$ si $\chi = 0$, sinon $h := \mathrm{Y}$.

Cas $\chi = 0$: On pose $y = \mathrm{Y}$, on rappelle que dans le schéma, le signataire prouve $\log_g y \vee (\log_g z_1 \wedge \log_h(z_u / z_1))$. Dans le cas $\chi = 0$, $\mathcal{M}$ connaîtra $\log_g z_1$ ainsi que $\log_h(z_u / z_1))$. Il pourra donc simuler parfaitement le signataire du schéma et espérer extraire $\log_g y$. $\mathcal{A}$ est exécuté deux fois avec deux réponses différentes à la Q-ème requête à H_2 et nous appliquons ensuite le *lemme de séparation* (*cf.* Lemme 3).

1. $\mathcal{M}$ pose $y = \mathrm{Y}$.

2. $\mathcal{M}$ choisit $w, w_0, w_1 \in \mathbb{Z}_q^*$ et pose $h := g^w$, $z := \mathrm{H}_1(p||q||g||y) = g^{w_0}$ et $y_t = g^{w_1}$.
3. $\mathcal{M}$ utilise $\mathcal{A}$ et simule la $i^{ème}$ réponse de $\mathcal{S}$ de la manière suivante :
   (a) En recevant $(z_{ui}, \xi_i, \mathrm{E}_i, \mathrm{P}_i)$, $\mathcal{M}$ vérifie la validité de P_i et rejette la requête en cas d'erreur. Sinon, $\mathcal{M}$ déchiffre E_i et reçoit γ_i.
   (b) $\mathcal{M}$ calcule $a_i := g^{r_i} y^{c_i}$ pour $c_i, r_i \in \mathbb{Z}_q$.
   (c) $\mathcal{M}$ calcule $w_{1i} = w_1 v_i \mod q$ et $w_{2i} = (w_0/\gamma_i - w_{1i})/w \mod q$ pour $v_i \in \mathbb{Z}_q^*$. Ensuite, $\mathcal{M}$ pose $z_{1i} = g^{w_{1i}}$ et $z_{2i} = h^{w_{2i}}$.
   (d) $\mathcal{M}$ calcule P_{si} en utilisant le témoin légitime v_i.
   (e) $\mathcal{M}$ calcule $b_{1i} := g^{u_{1i}}$ et $b_{2i} := h^{u_{2i}}$ avec $u_{1i}, u_{2i} \in \mathbb{Z}_q$.
   (f) $\mathcal{M}$ renvoie P_{si}, a_i, b_{1i}, b_{2i} à $\mathcal{A}$.
   (g) En recevant e_i de $\mathcal{A}$, $\mathcal{M}$ calcule $d_i := e_i - c_i \mod q$, $s_{1i} := u_{1i} - d_i w_{1i} \mod q$ et $s_{2i} := u_{2i} - d_i w_{2i} \mod q$.
   (h) $\mathcal{M}$ simule H_2 en retournant $\varepsilon \in \mathbb{Z}_q$.
4. $\mathcal{A}$ renvoie un signature que nous notons $(\zeta_1, \rho, \omega, \sigma_1, \sigma_2, \delta)$, qui correspond à la réponse ε.
5. $\mathcal{M}$ ré-exécute $\mathcal{A}$ avec les mêmes paramètres. Cette fois, $\mathcal{M}$ simule H_2 avec $\varepsilon' \in \mathbb{Z}_d$.
6. $\mathcal{A}$ renvoie un signature, notée $(\zeta_1', \rho', \omega', \sigma_1', \sigma_2', \delta')$, qui correspond à la réponse ε'.
7. Si $\omega \neq \omega'$, $\mathcal{M}$ renvoie $\mathrm{X} := (\rho - \rho')/(\omega' - \omega) \mod q$. Sinon, la simulation échoue.

Cas $\chi = 1$: On pose $h = \mathrm{Y}$ et $z = g^{w_1} h^{w_2}$ avec w_1 et w_2 aléatoires. Le but est alors d'extraire différentes représentations de z afin d'obtenir $\log_g h$. Cette fois, l'oracle de signature est simulé en utilisant $\log_g y$ sauf pour une seule requête. Pour cette $\mathrm{J}^{ème}$ requête, choisie aléatoirement, on utilise w_1 et w_2, c'est-à-dire la représentation en bases g et h de z. $\mathcal{U}_1^*$ est ensuite réexécuté pour pouvoir appliquer le lemme de bifurcation. Dans cette seconde exécution, la valeur d utilisée dans la $\mathrm{J}^{ème}$ session est modifiée. Il est alors possible de répondre pour deux valeurs d différentes dans la $\mathrm{J}^{ème}$ session étant donné que la représentation de z est (w_1, w_2). Maintenant, si δ est modifié par le changement de d, on obtient différentes valeurs pour δ. Il est alors possible d'extraire le logarithme de z en base h et g et les valeurs obtenues sont différentes de w_1 et w_2. $\mathcal{M}$ agit de la façon suivante :

1. $\mathcal{M}$ pose $h = \mathrm{Y}$.
2. $\mathcal{M}$ choisit $x \in \mathbb{Z}_q$ et pose $y := g^x$. Il choisit aussi $w_1, w_2 \in \mathbb{Z}_q$ et pose $z := \mathrm{H}(p||q||g||y) = g^{w_1} h^{w_2}$.
3. $\mathcal{M}$ choisit $\mathrm{J} \in \{1, \ldots l\}$ et aussi v_J et pose $y_t = g^{w_1/v_\mathrm{J}}$.
4. $\mathcal{M}$ utilise $\mathcal{A}$ et simule les requêtes à l'oracle de signature pour la $i^{ème}$ requête de la manière suivante :
   (a) Pour $i \neq \mathrm{J}$, $\mathcal{M}$ suit le protocole en utilisant $x = \log_g y$. H_2 est simulé en retournant un élément aléatoire de $\langle g \rangle$.

(b) Pour $i = \mathrm{J}$, $\mathcal{M}$ s'engage dans le protocole en utilisant x, w_1 et w_2 de la manière suivante :

i. Après avoir reçu $(z_{ui}, \xi_i, \mathrm{E}_i, \mathrm{P}_i)$ de l'utilisateur, $\mathcal{M}$ vérifie si P_i est correct (dans le cas contraire il abandonne la procédure). Il déchiffre ensuite P_i et pose $\gamma_i = \mathrm{E}_i$.

ii. $\mathcal{M}$ pose $z_{i\mathrm{J}} = y_t^{v_\mathrm{J}}$.

iii. $\mathcal{M}$ calcule $a_\mathrm{J} = g^{u_\mathrm{J}}$, $b_{1\mathrm{J}} = g^{u_{1\mathrm{J}}}$, $b_{2\mathrm{J}} = h^{u_{2\mathrm{J}}}$ avec $u_\mathrm{J}, u_{1\mathrm{J}}, u_{2\mathrm{J}} \in \mathbb{Z}_q$.

iv. $\mathcal{M}$ renvoie $(v_\mathrm{J}, a_\mathrm{J}, b_{1\mathrm{J}}, b_{2\mathrm{J}})$ à $\mathcal{A}$.

v. En recevant e_J de $\mathcal{A}$, $\mathcal{M}$ choisit $d_\mathrm{J} \in \mathbb{Z}_q$ et calcule $c_\mathrm{J} := e_\mathrm{J} - d_\mathrm{J} \mod q$, $r_\mathrm{J} := u_\mathrm{J} - c_\mathrm{J} x \mod q$, $s_{1\mathrm{J}} := u_{1\mathrm{J}} - d_\mathrm{J} w_1 \mod q$ et $s_{2\mathrm{J}} := u_{2\mathrm{J}} - d_\mathrm{J} w_2 \mod q$.

vi. $\mathcal{M}$ renvoie $(r_\mathrm{J}, c_\mathrm{J}, s_{1\mathrm{J}}, s_{2\mathrm{J}}, d_\mathrm{J})$ à $\mathcal{A}$.

$\mathcal{M}$ simule H_2 en renvoyant $\varepsilon \in_\mathrm{R} \mathbb{Z}_q$.

(c) $\mathcal{A}$ renvoie une signature $(\zeta_1, \rho, \omega, \sigma_1, \sigma_2, \delta)$ qui correspond à ε.

(d) $\mathcal{M}$ réexécute $\mathcal{A}$ avec les mêmes paramètres et choisit ensuite $\mathrm{I} \in \{0, \ldots, l\}$.
- Si $\mathrm{I} = 0$, $\mathcal{M}$ simule H_2 en renvoyant $\varepsilon' \in \mathbb{Z}_q$, sinon pose $\varepsilon' = \varepsilon$.
- Si $\mathrm{I} \neq 0$ et l'exécution J n'a pas été terminée avant que la réponse de la requête à H_2 soit envoyée alors $\mathcal{M}$ simule l'exécution en utilisant les témoins des deux côtés z et y, en choisissant $d'_\mathrm{J} \in \mathbb{Z}_q$. Sinon, $\mathcal{M}$ simule uniquement le témoin du côté y en choisissant $d'_\mathrm{J} = d_\mathrm{J}$.

(e) $\mathcal{A}$ renvoie une signature $(\zeta_1, \rho', \omega', \sigma'_1, \sigma'_2, \delta')$ qui correspond à ε'.

(f) Si $\delta = \delta'$ alors la simulation échoue. Sinon, $\mathcal{M}$ calcule $w'_1 = (\sigma_1 - \sigma'_1)/(\delta' - \delta') \mod q$, $w'_2 = (\sigma_2 - \sigma'_2)/(\delta - \delta') \mod q$ et renvoie $\mathrm{X} = (w_1 - w'_1)/(w'_2 - w_2) \mod q$.

Probabilité de succès On suppose tout d'abord que toutes les variables choisies par le signataire « simulé » sont déterminées par un ruban aléatoire, donc sont fixées avant même que la simulation commence. Nous nous intéressons à la valeur δ et cherchons à voir comment elle évolue en fonction des changements de ε et de $\{d_{i_{k+1}}, \ldots, d_{i_l}\}$, les valeurs données après que ε eut été donné à $\mathcal{A}$. On remarque aussi que les variables p, q, g, h, y, H_1, H_2, a_i, b_{1i}, b_{2i}, d_i et ε sont indépendantes les unes des autres et toutes les autres valeurs sont déterminées de manière unique par ces variables indépendantes. Nous regroupons toutes ces variables, hormis $\varepsilon, d_{i_{k+1}}, \ldots, d_{i_l}$, dans un ensemble que nous dénoterons Λ. L'ensemble $\{\varepsilon, d_{i_{k+1}}, \ldots, d_{i_l}\}$ sera noté D_ε. Soit D l'ensemble $\mathrm{D}_\varepsilon \backslash \{\varepsilon\}$.

Soit S l'ensemble des couples $(\Lambda, \mathrm{D}_\varepsilon)$ tels que $\mathcal{A}$ réussisse son attaque, autrement dit, $\Pr_{\lambda, \mathrm{D}_\varepsilon}[(\Lambda, \mathrm{D}_\varepsilon) \in \mathrm{S}] \geq \varepsilon_1$. D'après le *lemme de séparation*, un ensemble Λ choisi aléatoirement satisfait $\Pr_{\Lambda, \mathrm{D}_\varepsilon}[(\Lambda, \mathrm{D}_\varepsilon) \in \mathrm{S}] \geq \varepsilon_1/2$ avec probabilité $\varepsilon_1/2$. Une fois Λ fixé, δ est déterminé de manière unique par D_ε. On note $\delta \leftarrow \mathrm{D}_\varepsilon$ l'application qui à $(\Lambda, \mathrm{D}_\varepsilon) \in \mathrm{S}$ associe δ. Si $(\Lambda, \mathrm{D}_\varepsilon) \notin \mathrm{S}$, on utilisera la notation suivante : $\perp \leftarrow \mathrm{D}_\varepsilon$. La fonction ϕ est définie par

$$\phi : \delta \mapsto \Pr_{\mathrm{D}_\varepsilon}[\delta \leftarrow \mathrm{D}_\varepsilon].$$

Soit δ_{max} la valeur qui maximise $\phi(\delta)$. Autrement dit, δ_{max} est la valeur de δ qui a le plus de probabilité d'apparaître dans Σ^*. On pose $\phi_{max} = \phi(\delta_{max})$. Deux cas sont à envisager :

Cas 1 : ϕ_{max} n'est pas négligeable Dans ce cas, pour des valeurs D_ε et D'_ε choisies aléatoirement, l'adversaire renverra sûrement des signatures qui contiennent la valeur δ_{max} avec une probabilité suffisamment grande. Si δ est le même pour des ε différents, alors les valeurs ω sont différentes, étant donné que $\delta + \omega = \varepsilon$. Par conséquent, on obtient $\omega \neq \omega'$ avec une probabilité suffisante. On peut alors extraire le logarithme discret de y, comme vu à l'étape 7 du cas $\chi = 0$.

Cas 2 : ϕ_{max} est négligeable Dans ce cas, δ est susceptible de changer si D_ε est modifié. Abe a montré dans [Abe01] (*cf.* lemme 3 de [Abe01]), que deux valeurs D_ε et D'_ε choisies aléatoirement et dont une seule valeur diffère (par exemple celle à la i-ème position) permettent à $\mathcal{A}$ de renvoyer deux signatures $(\zeta_1, \rho, \omega, \sigma_1, \sigma_2, \delta)$ et $(\zeta_1, \rho', \omega', \sigma'_1, \sigma'_2, \delta')$ avec une probabilité suffisamment grande.

Nous rappelons ici brièvement la preuve du lemme 3. Posons $Id = (0, i_{k+1}, \ldots, i_l)$ et, pour $i \in Id$, soit D_ε^{-i} la suite obtenue en supprimant la valeur d_i de D_ε. On remarque alors que, par définition de ϕ_{max}, $\Pr_{D_\varepsilon}[\delta \leftarrow D_\varepsilon] \leq \phi_{max}$ pour tout δ. On suppose que D_ε est uniformément choisi et δ est défini par $\delta \leftarrow D_\varepsilon$. Alors, d'après le lemme de séparation, il existe $J \in Id$ tel que

$$\Pr_{d_J}[\delta \leftarrow D_\varepsilon^{-J} \cup \{d_J\}] > 1 - \phi_{max}$$

pour D_ε^{-J} choisi aléatoirement, avec probabilité plus petite que ϕ_{max}. La probabilité de choisir le « bon » J est de $\frac{1}{l+1}$ au plus. En prenant la probabilité complémentaire de la précédente, on constate que, pour un D_ε^{-J} aléatoire,

$$\Pr_{d_J}[\delta \nleftarrow D_\varepsilon^{-J} \cup \{d_J\}] \geq \phi_{max}$$

avec probabilité $\geq 1 - \phi_{max}$. On suppose maintenant que D'_ε est construit à partir de D_ε en choisissant $d_J \in_R \mathbb{Z}_q$ et que δ' est défini par $\delta' \leftarrow D'_\varepsilon$. D'après ce qui précède, l'événement $\{\delta \neq \delta\} \wedge \{(\Lambda, D'_\varepsilon) \notin S\}$ se produit avec probabilité non négligeable. D'après le lemme de séparation, des valeurs D_ε^{-J} choisies aléatoirement satisfont

$$\Pr_{d_J}[(\Lambda, D_\varepsilon^{-J} \cup \{d_J\}) \in S] \geq \varepsilon_1/4$$

avec probabilité $\varepsilon_1/4$. Alors, avec probabilité non négligeable, de tels D_ε et D'_ε sont dans S et garantissent $\delta' \neq \delta$.

À partir de ces signatures, on peut extraire w'_1 et w'_2 qui satisfont $\zeta_1 = g^{w'_1}$ et $\zeta/\zeta_1 = h^{w'_2}$. Par hypothèse, $\log_{z_u} z \neq \log_{z_1} \zeta_1$. Par conséquent, $w_1 \neq w'_1$ et $w_2 \neq w'_2$. $\mathcal{M}$ est alors capable de calculer $X = \log_g h = (w_1 - w'_1)/(w'_2 - w_2) \bmod q$. Nous renvoyons à la preuve de [AO01] pour un calcul plus détaillé des probabilités.

□

Théorème 20 (Traçage de signature - [AO01]). *Si le schéma de chiffrement vérifiable est sûr et significatif alors le schéma d'Abe et Ohkubo est, sous l'hypothèse du logarithme discret et dans le modèle de l'oracle aléatoire, traçable pour la signature.*

Démonstration. La preuve est similaire à celle du traçage d'identité et nous renvoyons à [AO01] pour une preuve détaillée. □

7.2.3 Sécurité polynomiale ou poly-logarithmique ?

Comme nous l'avons vu, la preuve du théorème 19 s'appuie sur certains résultats de [Abe01] (notamment le lemme 2 de [Abe01]). Dans cette section, nous expliquons pourquoi la preuve du théorème de [Abe01] est fausse, ce qui, par extension, entraîne que la preuve du théorème 19 est également fausse.

Soit $\mathcal{A}$ un adversaire qui réussit à produire une forge supplémentaire. Durant son attaque, il va recevoir une série de challenges que l'on note $\mathrm{D} = (d_1, \ldots, d_n)$. En relançant la simulation avec des ensembles D et D′ choisis aléatoirement, on arrive à une « collision » qui permet à la machine $\mathcal{M}$ de résoudre le problème du logarithme discret pour une instance donnée. Admettons maintenant, que pour des raisons techniques (dues essentiellement au mode de fonctionnement de la simulation) on ne puisse modifier qu'une seule valeur de D pour générer D′.

Soit d'_i l'élément qui diffère entre D et D′ à la $i^{ème}$ position. $\mathrm{D}' = (d_1, \ldots, d'_i, \ldots, d_n)$ où d'_i est choisi aléatoirement. Dans [Abe01], Abe prétend que de tels D_i et D'_i produisent quand même une collision, ce qui est incorrect. En effet, il peut exister un adversaire qui réussit à produire une forge supplémentaire pour suffisamment de D, tel que, pour tout i et tout d_i, D et D′ ne produiront pas de collisions ; autrement dit, l'adversaire échoue dans son attaque contre la falsification supplémentaire avec D′. Un tel cas ne peut pas se produire quand la probabilité de succès de l'adversaire, qu'on note ε, vaut 1. Mais si ε est inférieur à $1/2$, alors il est fort possible de se trouver dans la situation décrite ci-dessus. Cependant, une preuve qui ne remettrait en cause que l'existence d'un adversaire avec une probabilité de succès écrasante n'est malheureusement pas assez significative pour établir une propriété de sécurité. L'erreur dans la preuve provient de l'évaluation de la probabilité de succès de la réduction (plus précisément dans le cas où ϕ_{max} est considéré comme non négligeable dans la preuve du théorème 19). Abe évalue la probabilité P que l'adversaire renvoie deux signatures valides différentes (avec des valeurs δ différentes) après qu'il eut reçu D et D′. Il prétend que cette probabilité est non négligeable, ce qui lui permet effectivement de résoudre le problème du logarithme discret. Cependant, la seule chose qu'il prouve ici est le fait que des valeurs aléatoires $d_1, d_2, \ldots, d_{i-1}, d_{i+1}, \ldots, d_n$ satisfont

$$\mathrm{P} := \Pr_{d'_i}[\mathrm{D} \text{ et } \mathrm{D}' \text{ donnent deux signatures valides avec des valeurs } \delta \text{ distinctes}] \geq \phi_{max}$$

avec probabilité supérieure à $1 - \phi_{max}$. Ce calcul de probabilité étant fait pour le cas où ϕ_{max} est négligeable, la relation donnée ci-dessus ne permet pas de conclure que P n'est pas négligeable.

Cette erreur dans la preuve a été confirmée par Abe lui-même lors d'échanges personnels [Abe02]. Il nous a aussi informé qu'il n'a pas été capable de trouver un correctif à ce problème. Cependant, étant donné que son schéma [Abe01] est à témoin indistinguable, la preuve de la version complète de [AO00] s'applique aussi à ce schéma, mais elle permet seulement de montrer une sécurité poly-logarithmique.

Chapitre 8

Nouveau modèle de sécurité

Dans le chapitre précédent, nous avons présenté le schéma de référence en matière de signature aveugle à anonymat révocable. Jusqu'à récemment, ce schéma était le seul à disposer de preuves de sécurité complètes. Nous avons montré au chapitre 7 que certaines de ces preuves étaient fausses. La sécurité du schéma d'Abe et Ohkubo n'est donc que conjecturée. La correction de celui-ci s'avérant difficile à réaliser, il nous est apparu intéressant de construire un nouveau schéma respectant les propriétés de sécurité énoncées dans [AO01]. Nous avons alors constaté que le modèle présenté ne permettait pas de prouver toutes les propriétés souhaitées.

Nous avons donc construit un nouveau modèle de sécurité pour les signatures aveugles à anonymat révocable en nous attachant à définir de manière précise les actions menées par l'adversaire. Ce modèle a été présenté avec J. Traoré dans [HT07].

Ce chapitre présente dans la première section les faiblesses et les manques du modèle d'Abe et Ohkubo. Les sections suivantes sont consacrées aux définitions de sécurité.

Sommaire

8.1 Motivation

Abe et Ohkubo ont été les premiers à proposer dans [AO01], un modèle de sécurité détaillé pour les FBS. Ce modèle est celui que nous avons présenté au chapitre précé-

dent. Cependant, le formalisme employé n'est pas entièrement satisfaisant et certaines propriétés de sécurité ne sont pas abordées.

Nous rappelons tout d'abord les critères de sécurité souhaitables pour un FBS et montrons en quoi le modèle de [AO01] ne permet pas de les couvrir entièrement.

Comme nous l'avons décrit au chapitre 7, une signature aveugle à anonymat révocable est avant tout une signature aveugle. C'est pourquoi elle se doit de respecter, au moins, les propriétés d'indistinguabilité et de non-falsification supplémentaire. Les FBS doivent aussi être traçables au sens donné au chapitre 7. Il ne doit donc pas être possible pour un adversaire de produire une signature ne s'ouvrant sur aucune identité connue ou une session non reliée à une signature.

Les définitions 89 et 90 données par Abe et Ohkubo précisent les buts que l'adversaire doit atteindre pour casser la traçabilité d'un schéma, que ce soit pour le traçage de session ou de signature. Cependant, rien n'est dit quant aux moyens dont dispose l'adversaire pour mener son attaque. En particulier, sa connaissance des clés secrètes du signataire et de l'autorité de révocation n'est pas précisée. Par défaut, ceci indique qu'il n'y a pas accès. Il est cependant intéressant de se demander si un schéma reste sûr même quand l'adversaire dispose de l'une de ces clés. En outre, Abe et Ohkubo ne précisent pas ce qu'ils entendent par une vue $view_i$ d'une session. Notamment, il n'est pas précisé si les sessions n'ayant pas abouti sont considérées, elles aussi, comme des vues ou non.

Par ailleurs, il peut être plus intéressant, pour des raisons pratiques, lors du « traçage de session », de remonter directement à l'utilisateur plutôt qu'à un identifiant de session. Dans ce cas de figure, la définition donnée par Abe et Ohkubo n'est plus satisfaisante. En effet, toutes les signatures émises par un utilisateur portant le même identifiant, il devient alors facile, pour un adversaire, de casser leur notion de traçabilité de session.

Les propriétés de sécurités ainsi décrites protègent le signataire face à des utilisateurs malintentionnés. Mais rien n'est mis en place pour protéger l'identité de l'utilisateur. Il est en effet facile d'imaginer qu'un adversaire veuille usurper l'identité d'un utilisateur honnête pour signer à sa place, ou bien qu'il s'arrange pour qu'une session qu'il exécute s'ouvre sur une signature qu'un utilisateur a produite de manière légale. Ces propriétés, que nous appelons propriétés de *non-diffamation*, sont essentielles pour définir totalement la sécurité d'un schéma de signature aveugle à anonymat révocable.

Dans la suite de ce chapitre, nous présentons donc un nouveau modèle de sécurité prenant en compte toutes les attentes d'un schéma de signature aveugle à anonymat révocable, en nous attachant à définir de manière précise les moyens et les buts de l'adversaire.

8.2 Oracles

Afin de proposer le modèle le plus détaillé possible, nous commençons dans cette section par définir les moyens dont dispose un attaquant pour mener à bien ses attaques. Ceux-ci sont représentés sous forme d'oracles auxquels l'adversaire a accès selon l'expérience qu'il conduit. Ils ont pour but de simuler parfaitement le comportement des

participants auxquels l'attaquant est confronté.

Notations

Dans la suite de ce chapitre, nous utiliserons les notations suivantes :

- HU représentera l'ensemble des utilisateurs honnêtes.
- CU représentera l'ensemble des utilisateurs corrompus par l'adversaire.
- Set sera l'ensemble des couples message/signature obtenus lors d'interactions avec des utilisateurs honnêtes. Ces signatures seront appelées par la suite, par abus de langage, signatures honnêtes.
- Trans sera l'ensemble des transcripts des données échangées entre un utilisateur et un signataire lors d'une exécution du protocole.

Définition des Oracles

Afin de réaliser son attaque, l'adversaire a à sa disposition divers oracles lui permettant d'interagir avec les différents acteurs, à savoir les utilisateurs (représentés par leurs identités), le signataire ou l'autorité de révocation. Il peut même dans certains cas jouer totalement le rôle d'un de ces participants.

Tout d'abord, l'attaquant peut enregistrer de nouveaux utilisateurs afin de les utiliser ensuite pour obtenir des signatures. Il peut créer soit des utilisateurs honnêtes (il ne connaît alors que les données publiques des utilisateurs associés), soit des utilisateurs malhonnêtes (il choisit lui-même leurs secrets). Pour ce faire, il a à sa disposition trois oracles :

$\mathsf{AddU}(.)$ Cet oracle permet à l'adversaire d'inscrire de nouveaux utilisateurs. L'adversaire fournit une identité $\mathrm{Id}_{\mathcal{U}}$ à l'oracle. Celle-ci est ajoutée à la liste HU des utilisateurs honnêtes et l'oracle calcule les clés de l'utilisateur. La clé secrète $sk_{\mathrm{Id}_{\mathcal{U}}}$ est conservée par l'oracle, la clé publique $pk_{\mathrm{Id}_{\mathcal{U}}}$ est renvoyée à l'adversaire.
$(pk_{\mathrm{Id}_{\mathcal{U}}}) \leftarrow \mathsf{AddU}(\mathrm{Id}_{\mathcal{U}})$,
$\mathrm{HU} \leftarrow \mathrm{HU} \cup (\mathrm{Id}_{\mathcal{U}}, sk_{\mathrm{Id}_{\mathcal{U}}}, pk_{\mathrm{Id}_{\mathcal{U}}})$.

$\mathsf{CrptU}(.,.,.)$ Grâce à cet oracle, l'adversaire est en mesure de corrompre des utilisateurs. L'adversaire fournit à l'oracle l'identité $\mathrm{Id}_{\mathcal{U}}$ d'un utilisateur. Il fixe la clé publique à $pk_{\mathrm{Id}_{\mathcal{U}}}$ et la clé privée à $sk_{\mathrm{Id}_{\mathcal{U}}}$. Ces paramètres sont ajoutés à la liste CU des utilisateurs corrompus.
$\mathrm{CU} \leftarrow \mathrm{CU} \cup (\mathrm{Id}_{\mathcal{U}}, sk_{\mathrm{Id}_{\mathcal{U}}}, pk_{\mathrm{Id}_{\mathcal{U}}})$.

$\mathsf{USK}(.,.)$ L'adversaire peut appeler cet oracle en lui fournissant une identité $\mathrm{Id}_{\mathcal{U}}$. L'oracle vérifie tout d'abord que $\mathrm{Id}_{\mathcal{U}} \in \mathrm{HU}$ et renvoie ensuite la clé privée $sk_{\mathrm{Id}_{\mathcal{U}}}$. Par la suite l'utilisateur dont l'identité est $\mathrm{Id}_{\mathcal{U}}$ est considéré comme corrompu et est ajouté à la liste CU.
$(sk_{\mathrm{Id}_{\mathcal{U}}}) \leftarrow \mathsf{USK}(\mathrm{Id}_{\mathcal{U}})$,
$\mathrm{HU} \leftarrow \mathrm{HU} \setminus (\mathrm{Id}_{\mathcal{U}}, sk_{\mathrm{Id}_{\mathcal{U}}}, pk_{\mathrm{Id}_{\mathcal{U}}})$,
$\mathrm{CU} \leftarrow \mathrm{CU} \cup (\mathrm{Id}_{\mathcal{U}}, sk_{\mathrm{Id}_{\mathcal{U}}}, pk_{\mathrm{Id}_{\mathcal{U}}})$.

L'adversaire peut aussi décider, au cours de son attaque, d'exécuter le protocole de signature, soit en jouant le rôle d'un utilisateur, soit en jouant le rôle d'un signataire. Il a pour cela, deux oracles à sa disposition :

$\mathsf{User}(.,.,.)$ Cet oracle est utilisé pour simuler une exécution du protocole de signature entre un signataire corrompu et un utilisateur honnête. L'adversaire fournit à l'oracle une identité $\mathrm{I}d_{\mathcal{U}}$ d'un utilisateur honnête, la clé publique associée $pk_{\mathrm{I}d_{\mathcal{U}}}$ et un message m. L'oracle vérifie que $\mathrm{I}d_{\mathcal{U}}$ appartient bien à HU et que la clé publique est correcte. Si le protocole n'échoue pas, l'adversaire reçoit en réponse une transcription $trans_{m,\sigma}$ du protocole, qui est ajoutée à la liste Trans, ainsi que l'ensemble $(m, \sigma, \mathrm{I}d_{\mathcal{U}})$, où σ est une signature aveugle de m. Le triplet $(m, \sigma, \mathrm{I}d_{\mathcal{U}})$ est ajouté à l'ensemble Set.
$(m, \sigma, \mathrm{I}d_{\mathcal{U}}, trans_{m,\sigma}) \leftarrow \mathsf{User}(\mathrm{I}d_{\mathcal{U}}, pk_{\mathrm{I}d_{\mathcal{U}}}, m)$,
Trans $\leftarrow$ Trans $\cup trans_{m,\sigma}$,
Set $\leftarrow$ Set $\cup(m, \sigma, \mathrm{I}d_{\mathcal{U}})$.

$\mathsf{Sign}(.,.)$ Cet oracle est utilisé pour simuler une exécution du protocole de signature entre un signataire honnête et un utilisateur corrompu. L'adversaire, qui joue ici le rôle de l'utilisateur, donne en entrée une identité $\mathrm{I}d_{\mathcal{U}}$ ainsi qu'un message m. Il exécute ensuite le protocole avec l'oracle au nom de cet utilisateur. Si le protocole n'échoue pas, l'adversaire est en mesure de calculer une signature σ de son message m et le transcript du protocole est ajouté à Trans.
$(\sigma, trans_{m,\sigma}) \leftarrow \mathsf{Sign}(\mathrm{I}d_{\mathcal{U}}, m)$,
Trans $\leftarrow$ Trans $\cup trans_{m,\sigma}$.

Tout comme il peut être amené à engager des signatures, l'adversaire peut aussi demander à lever l'anonymat des signatures ou tracer des signatures à l'aide des deux oracles suivants :

$\mathsf{Tsig}(.)$ En fournissant à cet oracle la transcription d'une session, l'adversaire obtient la sortie de l'algorithme $\mathsf{R}_{\mathsf{sig}}$.
$(\mathrm{I}_{sig}, \pi_1) \leftarrow \mathsf{Tsig}(trans_{m,\sigma})$.

$\mathsf{Tid}(.)$ En fournissant à cet oracle un couple message/signature, l'adversaire obtient la sortie de l'algorithme R_{id}.
$(\mathrm{I}_{id}, \pi_2) \leftarrow \mathsf{Tid}(m, \sigma)$.

Le dernier oracle est un « oracle de challenge » qui sera utilisé dans la définition de l'indistinguabilité d'un schéma de FBS. Cet oracle permet à l'adversaire de recevoir un challenge auquel il doit répondre. Le bit b est choisi avant l'appel à l'oracle et est inconnu de l'adversaire.

$\mathsf{Choose}_b(.,.,.)$ L'adversaire donne à l'oracle deux identités $\mathrm{I}d_{\mathcal{U}_0}$ et $\mathrm{I}d_{\mathcal{U}_1}$ et deux messages m_0 et m_1. L'oracle fournit m_b à $\mathrm{I}d_{\mathcal{U}_0}$ et $m_{\bar{b}}$ à $\mathrm{I}d_{\mathcal{U}_1}$ ($\bar{b}$ représente désormais la valeur $1-b$) . L'adversaire reçoit en réponse deux signatures : σ_b pour le message m_b et $\sigma_{\bar{b}}$ du

message $m_{\bar{b}}$ pour l'identité $\mathrm{Id}_{\mathcal{U}_1}$ tant que $\mathrm{Id}_{\mathcal{U}_0}$ et $\mathrm{Id}_{\mathcal{U}_1}$ sont toutes deux rattachées à des utilisateurs honnêtes. L'oracle enregistre les messages et les identités de manière à ce que l'adversaire ne puisse les donner en entrée à un oracle de traçabilité par la suite.
$(\sigma_b, \sigma_{\bar{b}}) \leftarrow \mathsf{Choose}_b(\mathrm{Id}_{\mathcal{U}_0}, \mathrm{Id}_{\mathcal{U}_1}, m_0, m_1)$.

La description de ces oracles nous permet maintenant de définir les expériences que l'adversaire va mener afin de mettre en défaut les propriétés de sécurité que nous souhaitons pour un schéma de signature aveugle à anonymat révocable. Pour chaque propriété, nous décrivons algorithmiquement l'expérience conduite par l'adversaire en précisant ses entrées, les oracles auxquels il peut faire appel ainsi que les conditions pour que son attaque réussisse. Ceci nous permet ensuite de définir son avantage de manière précise.

8.3 Consistance

La consistance est une propriété qui garantit la validité du protocole. En d'autres termes, une signature σ d'un message m émise en respectant le protocole doit toujours être acceptée. En outre, les algorithmes de révocation doivent toujours identifier de manière unique soit un utilisateur, soit une signature. La consistance est formalisée à travers une expérience impliquant un adversaire.

Pour tout schéma de signature aveugle à anonymat révocable, pour tout adversaire $\mathcal{A}$ et tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\mathrm{corr}}_{\mathrm{FBS},\mathcal{A}}(k)$ (*cf.* figure 8.1) et l'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{corr}}_{\mathrm{FBS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{corr}}_{\mathrm{FBS},\mathcal{A}}(k) = 1].$$

On dit que le schéma est *consistant* si $\mathbf{Adv}^{\mathrm{corr}}_{\mathrm{FBS},\mathcal{A}}(k) = 0$ pour tout adversaire $\mathcal{A}$ et tout $k \in \mathbb{N}$.
L'expérience est décrite plus en détail dans la figure 8.1.

Expérience $\mathbf{Exp}^{\mathrm{corr}}_{\mathrm{FBS},\mathcal{A}}(k)$:

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k); (pk_{\mathrm{RA}}, sk_{\mathrm{RA}}) \leftarrow \mathsf{G}_{\mathrm{RA}}$
$\mathrm{HU} \leftarrow \emptyset$; $\mathrm{CU} \leftarrow \emptyset$; $\mathrm{Trans} \leftarrow \emptyset$
$(m, \mathrm{Id}_{\mathcal{U}}) \leftarrow \mathcal{A}(pk_{\mathcal{S}}, pk_{\mathrm{RA}}, \mathsf{AddU})$
`si` $\mathrm{Id}_{\mathcal{U}} \notin \mathrm{HU}$ `alors` renvoie 0 `si` $sk_{\mathrm{Id}_{\mathcal{U}}} = \varepsilon$ `alors` renvoie 0
$\sigma \leftarrow \mathsf{BSsign}(\mathrm{Id}_{\mathcal{U}}, m, sk_{\mathrm{Id}_{\mathcal{U}}})$
`si` $\mathsf{Verify}(pk_{\mathcal{S}}, m, \sigma) = 0$ `alors` renvoie 1
`si` $\forall trans_i \in \mathrm{Trans}$, $(\mathrm{I}^i_{sig}, \pi^i_1) \leftarrow \mathsf{R}_{\mathsf{Sig}}(trans_i, sk_{\mathrm{RA}})$ et $\mathsf{Match}_{\mathsf{Sig}}(\mathrm{I}^i_{sig}, (m, \sigma), \pi^i_1) = 0$
`alors` renvoie 1
$(\mathrm{I}_{id}, \pi_2) \leftarrow \mathsf{R}_{\mathsf{id}}(m, \sigma)$
`si` $\mathrm{I}_{id} \neq \mathrm{Id}_{\mathcal{U}}$ ou $\mathsf{Match}_{\mathsf{id}}(\mathrm{I}_{id}, i, \pi_2) = 0$ `alors` renvoie 1

FIG. 8.1 – Consistance

8.4 Indistinguabilité

L'*indistinguabilité* d'un schéma de signature aveugle à anonymat révocable s'apparente à celle des schémas de signatures aveugles. La seule différence est que l'adversaire à la possibilité de faire appel aux oracles de traçabilité Tsig et Tid. Personne ne doit être en mesure d'extraire des informations importantes à partir d'un couple message/signature ou de relier entre eux deux couples message/signature. Le but de l'adversaire est de déterminer la provenance de deux signatures correspondant à deux messages et deux utilisateurs de son choix donnés à son oracle de challenge.

À tout schéma de signature aveugle à anonymat révocable FBS, à tout adversaire $\mathcal{A}$, à tout bit $b \in \{0,1\}$ et à tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{FBS},\mathcal{A}}(k)$ (*cf.* figure 8.2). L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{blind}}_{\text{FBS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{blind}-1}_{\text{FBS},\mathcal{A}}(k) = 1] - \Pr[\mathbf{Exp}^{\text{blind}-0}_{\text{FBS},\mathcal{A}}(k) = 1].$$

Cet avantage peut aussi s'écrire

$$\mathbf{Adv}^{\text{blind}}_{\text{FBS},\mathcal{A}}(k) = 2|\Pr[\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{FBS},\mathcal{A}}(k) = b] - 1/2|.$$

On dit alors que le schéma est *indistinguable* ou *aveugle* si la fonction $\mathbf{Adv}^{\text{blind}}_{\text{FBS},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

Dans cette attaque, l'adversaire $\mathcal{A}$ a accès à la clé privée du signataire $sk_{\mathcal{S}}$. Il est donc capable d'interagir avec des utilisateurs en jouant le rôle de signataire. Il a aussi la possibilité d'ajouter de nouveaux utilisateurs honnêtes (à l'aide de l'oracle AddU) ou corrompus (avec l'oracle CrptU), ou de les corrompre après coup (avec l'oracle USK). Il a aussi accès à l'oracle de signature User afin de générer des signatures avec les utilisateurs de son choix. Il peut aussi demander aux oracles de traçabilité de lever l'anonymat sur les signatures ou les sessions de son choix. À un moment donné de son attaque (moment qu'il choisit), l'adversaire interroge son oracle de challenge Choose_b en lui envoyant deux identités d'utilisateurs non corrompus $\text{Id}_{\mathcal{U}_0}$ et $\text{Id}_{\mathcal{U}_1}$ ainsi que deux messages m_0 et m_1. Celui-ci renvoie à la fin de son attaque un bit b' qui est sa supposition sur la valeur de b.

Expérience $\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{FBS},\mathcal{A}}(k)$

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k)$
$(pk_{\text{RA}}, sk_{\text{RA}}) \leftarrow \mathsf{G}_{\text{RA}}(1^k)$
$b' \leftarrow \mathcal{A}(pk_{\mathcal{S}}, sk_{\mathcal{S}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}, \mathsf{Choose}_b, \mathsf{User}, \mathsf{Tsig}, \mathsf{Tid})$
renvoie b'

FIG. 8.2 – Indistinguabilité

Dans notre expérience, on peut considérer qu'en réalité, l'adversaire a accès à un nombre polynomial de fois à l'oracle de challenge Choose_b. Mais, comme l'ont expliqué

Bellare, Shi et Zhang dans [BSZ05], l'expérience peut être réduite au cas où l'adversaire ne fait qu'une seule requête à cet oracle.

Lemme 5. *Soient un schéma de signature aveugle à anonymat révocable et un adversaire $\mathcal{B}$ (s'exécutant en temps polynomial) contre l'indistinguabilité du schéma faisant au plus q requêtes à l'oracle* $\mathsf{Choose_b}$. *Alors il est possible de construire un adversaire $\mathcal{A}$ contre l'indistinguabilité du schéma faisant exactement une requête à l'oracle* Choose_b *et*

$$\mathsf{Adv}^{\text{blind}}_{\text{FBS},\mathcal{B}}(k) \leq q.\mathsf{Adv}^{\text{blind}}_{\text{FBS},\mathcal{A}}(k).$$

Preuve La preuve (qui utilise un « argument hybride ») est identique à celle décrite dans [BSZ05] (lemme B.1).

8.5 Traçabilité

La propriété de *traçabilité* est une propriété essentielle des schémas FBS. C'est elle qui garantit aux autorités que toutes les signatures émises seront révocables par l'autorité de révocation mandatée.

La traçabilité d'un schéma FBS s'entend de deux manières différentes. En effet, l'adversaire peut s'attaquer soit au *traçage d'identité* soit au *traçage de signature* et les expériences qu'il va conduire selon ces deux cas sont bien distinctes. C'est pourquoi nous les traitons comme deux propriétés séparées.

8.5.1 Traçage d'identité

En menant cette attaque, le but de l'adversaire est de produire une signature aveugle valide qui ne puisse être reliée à aucune identité d'utilisateur.

À tout schéma de signature aveugle à anonymat révocable FBS, à tout adversaire $\mathcal{A}$ et à tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{IdTrac}}_{\text{FBS},\mathcal{A}}(k)$ (*cf.* figure 8.3). L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{IdTrac}}_{\text{FBS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{IdTrac}}_{\text{FBS},\mathcal{A}}(k) = 1].$$

On dit que le schéma est *traçable pour l'identité* si la fonction $\mathbf{Adv}^{\text{IdTrac}}_{\text{FBS},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

Pour mener son attaque, l'adversaire est capable, comme précédemment, de faire enregistrer des utilisateurs honnêtes ou corrompus et d'obtenir les clés privées des utilisateurs de son choix à l'aide des oracles $\mathsf{AddU}, \mathsf{CrptU}$ et USK. Il n'a pas accès cette fois à la clé privée du signataire et ne peut donc interagir qu'avec un signataire honnête à l'aide de l'oracle Sign. À la fin de son attaque, l'adversaire renvoie un couple message/signature. L'adversaire a réussi son attaque si la signature qu'il produit est valide et vérifie l'une des conditions suivantes :

- l'algorithme de révocation d'identité renvoie une réponse non valide (la valeur I_{id} renvoyée par $\mathsf{R_{id}}$ n'est pas de la forme attendue) ou,

- le juge n'est pas capable de trouver une identité correspondante dans la liste des utilisateurs enregistrés (l'algorithme $\mathsf{Match_{id}}$ renvoie 0).

On suppose en outre, dans cette attaque, que l'autorité de révocation n'est pas entièrement corrompue. L'adversaire a bien accès à la clé privée sk_{RA} et peut donc ouvrir des signatures, mais il ne peut pas jouer le rôle de l'autorité face à un juge. Si l'autorité de révocation est appelée, elle répondra toujours de manière honnête. En effet, une autorité corrompue pourrait refuser tout simplement de tracer une identité, ce qui pourrait faire échouer l'attaque de façon directe.
L'expérience est décrite plus en détail dans la figure 8.3.

Expérience $\mathbf{Exp}^{\mathrm{IdTrac}}_{\mathrm{FBS},\mathcal{A}}(k)$:

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k); (pk_{\mathrm{RA}}, sk_{\mathrm{RA}}) \leftarrow \mathsf{G}_{\mathrm{RA}}(1^k)$
$(m, \sigma) \leftarrow \mathcal{A}(pk_{\mathcal{S}}, pk_{\mathrm{RA}}, sk_{\mathrm{RA}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}, \mathsf{Sign})$
`si` $\mathsf{Verify}(pk_{\mathcal{S}}, m, \sigma) = 0$ `alors` renvoie 0
$\mathsf{R_{id}}(m, \sigma, sk_{\mathrm{RA}}) = (\mathrm{I}_{id}, \pi_2)$
`si` $\mathrm{I}_{id} = \perp$ ou $\mathsf{Match_{id}}(\mathrm{I}_{id}, m, \sigma, \pi_2) = 0$ `alors` renvoie 1 `sinon` renvoie 0

FIG. 8.3 – Traçage d'identité

8.5.2 Traçage de signature

L'attaquant peut attaquer le *traçage de signature* de deux manières différentes :
- en produisant un couple message/signature valide tel que le signataire ne puisse pas retrouver de transcript relié à cette signature,
- en renvoyant deux couples message/signature qui soient reliés à la même transcription.

À tout schéma de signature aveugle à anonymat révocable FBS, à tout adversaire $\mathcal{A}$ et à tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\mathrm{SigTrac}}_{\mathrm{FBS},\mathcal{A}}(k)$ (*cf.* figure 8.4). L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\mathrm{SigTrac}}_{\mathrm{FBS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\mathrm{SigTrac}}_{\mathrm{FBS},\mathcal{A}}(k) = 1].$$

On dit que le schéma est *traçable pour la signature* si la fonction $\mathbf{Adv}^{\mathrm{SigTrac}}_{\mathrm{FBS},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

L'adversaire a accès aux mêmes oracles que pour l'attaque contre la traçabilité d'identité, à savoir $\mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}$ et Sign. Comme défini dans l'oracle Sign, tous les transcripts des exécutions sont mis dans l'ensemble Trans. L'adversaire ayant deux moyens de réussir son attaque, sa sortie finale peut être de deux sortes :
- soit $\mathcal{A}$ renvoie un couple message/signature (m, σ) et on dit que $\mathcal{A}$ a réussi son attaque si, pour tous les transcripts *trans* enregistrés dans la liste Trans et pour toutes les valeurs $(\mathrm{I}_{sig_i}, \pi_{1,i})$ renvoyées par l'autorité RA, l'algorithme

$\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig_i}, m, \sigma, \pi_{1,i})$ renvoie toujours 0 et donc le juge n'acceptera jamais la signature ;

- soit $\mathcal{A}$ renvoie deux couples message/signature (m_1, σ_1) et (m_2, σ_2) et on dit que $\mathcal{A}$ réussit son attaque si l'autorité RA trouve un transcript $trans$ tel que $\mathsf{R}_{\mathsf{sig}}(trans, sk_{\mathrm{RA}}) = (\mathrm{I}_{sig}, \pi_1)$ et l'algorithme $\mathsf{Match}_{\mathsf{sig}}$ renvoie 1 pour les deux signatures, c'est-à-dire $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, m_1, \sigma_1, \pi_1) = \mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, m_2, \sigma_2, \pi_1) = 1$ et le juge acceptera deux signatures pour le même transcript.

Comme pour la traçabilité d'identité, l'autorité de révocation n'est pas entièrement corrompue et doit répondre honnêtement aux requêtes qui lui sont faites.
L'expérience est décrite plus en détail dans la figure 8.4.

Expérience $\mathbf{Exp}^{\mathrm{SigTrac}}_{\mathrm{FBS},\mathcal{A}}(k)$:

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k); (pk_{\mathrm{RA}}, sk_{\mathrm{RA}}) \leftarrow \mathsf{G}_{\mathrm{RA}}$
Trans $\leftarrow \emptyset$
$(m, \sigma) \leftarrow \mathcal{A}(pk_{\mathcal{S}}, pk_{\mathrm{RA}}, sk_{\mathrm{RA}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}, \mathsf{Sign})$
`si` $\mathsf{Verify}(pk_{\mathcal{S}}, m, \sigma) = 0$ `alors` renvoie 0
`si` $\forall trans_i \in \mathrm{Trans}, \mathsf{R}_{\mathsf{sig}}(tras_i, sk_{\mathrm{RA}}) = (\mathrm{I}_{sig_i}, \pi_1)$ et $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig_i}, m, \sigma, \pi_1) = 0$
`alors` renvoie 1 `sinon` renvoie 0
ou
$(m_1, \sigma_1), (m_2, \sigma_2) \leftarrow \mathcal{A}(pk_{\mathcal{S}}, sk_{\mathrm{RA}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}, \mathsf{Sign})$
`si` $\mathsf{Verify}(pk_{\mathcal{S}}, m_1, \sigma_1) = 0$ ou `si` $\mathsf{Verify}(pk_{\mathcal{S}}, m_2, \sigma_2) = 0$
`alors` renvoie 0
`si` $\exists trans_i \in$ Trans tel que $\mathsf{R}_{\mathsf{sig}}(trans_i, sk_{\mathrm{RA}}) = (\mathrm{I}_{sig}, \pi_1)$
et $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, m_1, \sigma_1, \pi_1) = \mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, m_2, \sigma_2, \pi_1) = 1$
`alors` renvoie 1 `sinon` renvoie 0

FIG. 8.4 – Traçage de signature

8.6 Non-diffamation

Dans la section précédente, nous avons vu que toute signature aveugle à anonymat révocable doit être reliée à un identifiant de signature ou de session valide. La propriété de *non-diffamation* est là pour garantir aux utilisateurs honnêtes qu'il n'est pas possible de leur attribuer des faits qu'ils n'ont pas commis. Autrement dit, un adversaire ne peut pas falsifier des signatures ou des transcripts de session qu'il aurait produits malicieusement, de manière à ce qu'ils s'ouvrent sur des utilisateurs honnêtes ou des signatures obtenues honnêtement.

8.6.1 Non-diffamation d'identité

Dans la *non-diffamation d'identité*, le but de l'adversaire est de produire une preuve qu'un utilisateur honnête a obtenu une signature valide alors que cet utilisateur n'a pas

demandé cette signature. Dans cette attaque, on considère que le signataire et l'autorité de révocation sont entièrement corrompus.

À tout schéma de signature aveugle à anonymat révocable FBS, à tout adversaire $\mathcal{A}$ et à tout $k \in \mathbb{N}$ on associe l'expérience $\mathbf{Exp}^{\text{NonIdFram}}_{\text{FBS},\mathcal{A}}(k)$ (*cf.* figure 8.5). L'avantage de $\mathcal{A}$ est donné par

$$\mathbf{Adv}^{\text{NonIdFram}}_{\text{FBS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{NonIdFram}}_{\text{FBS},\mathcal{A}}(k) = 1].$$

On dit que le schéma est *non-diffamable pour l'identité* si la fonction $\mathbf{Adv}^{\text{NonIdFram}}_{\text{FBS},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

$\mathcal{A}$ a accès aux clés privées du signataire et de l'autorité de révocation et il peut jouer le rôle des deux entités vis-à-vis des participants honnêtes. Comme dans les expériences précédentes, l'adversaire peut faire inscrire des utilisateurs honnêtes voire les corrompre ou faire enregistrer des utilisateurs corrompus. Il interagit avec ces utilisateurs en jouant le rôle de signataire. À chaque fois qu'une signature est émise de manière honnête (c'est-à-dire quand l'utilisateur est honnête et accepte la signature), le couple message/signature ainsi que l'identité I_{id} de l'utilisateur sont ajoutés à l'ensemble Set.

On dit que $\mathcal{A}$ réussit son attaque s'il produit une signature σ sur un message m telle que toutes les conditions suivantes soient remplies :

- la signature est valide,
- l'algorithme $\mathsf{R_{id}}(m, \sigma, sk_{\text{RA}})$ renvoie une réponse (I_{id}, π_2) valide,
- le triplet $(m, \sigma, \mathrm{I}_{id})$ n'appartient pas à Set,
- I_{id} est une identité appartenant à HU,
- le juge accepte la preuve π_2.

Dans cette attaque, l'adversaire $\mathcal{A}$ est plus puissant que dans son attaque contre la traçabilité car il peut jouer à la fois le rôle signataire et celui de l'autorité de révocation.

On décrit l'expérience de manière plus formelle dans la figure 8.5.

Expérience $\mathbf{Exp}^{\text{NonIdFram}}_{\text{FBS},\mathcal{A}}(k)$:

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k) \qquad (pk_{\text{RA}}, sk_{\text{RA}}) \leftarrow \mathsf{G}_{\text{RA}}(1^k)$
Set $\leftarrow \emptyset$; HU $\leftarrow \emptyset$; CU $\leftarrow \emptyset$
$(m, \sigma) \leftarrow \mathcal{A}(pk_{\mathcal{S}}, sk_{\mathcal{S}}, sk_{\text{RA}}, pk_{\text{RA}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}, \mathsf{User})$
`si` $\mathsf{Verify}(pk_{\mathcal{S}}, m, \sigma) = 0$ `alors` renvoie 0
$\mathsf{R_{id}}(m, \sigma, sk_{\text{RA}}) = (\mathrm{I}_{id}, \pi_2)$
`si` $(m, \sigma, \mathrm{I}_{id}) \notin$ Set, $\mathrm{I}_{id} \in$ HU et $\mathsf{Match_{id}}(\mathrm{I}_{id}, m, \sigma, \pi_2) = 1$
`alors` renvoie 1 `sinon` renvoie 0

FIG. 8.5 – Non-diffamation d'identité

8.6.2 Non-diffamation de signature

Pour mener à bien une attaque contre la non-diffamation de signature, l'adversaire doit produire une signature qui est reliée à une transcription obtenue lors d'une exécution honnête ; autrement dit, l'adversaire produit une fausse signature attribuée à une transcription réelle. On considère aussi, dans cette attaque, que le signataire et l'autorité de révocation sont tous deux entièrement corrompus.

À tout schéma de signature aveugle à anonymat révocable FBS, à tout adversaire $\mathcal{A}$ et à tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{NonSigFram}}_{\text{FBS},\mathcal{A}}(k)$ (*cf.* figure 8.6). L'avantage de $\mathcal{A}$ est donné par

$$\mathbf{Adv}^{\text{NonSigFram}}_{\text{FBS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{NonSigFram}}_{\text{FBS},\mathcal{A}}(k) = 1].$$

On dit que le schéma est *non-diffamable pour la signature* si la fonction $\mathbf{Adv}^{\text{NonSigFram}}_{\text{FBS},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

$\mathcal{A}$ a accès aux mêmes oracles que pour la non-diffamation d'identité. $\mathcal{A}$ joue donc ici aussi le rôle d'un signataire corrompu face à des utilisateurs honnêtes. A chaque fois que l'adversaire fait appel à l'oracle User qui lui permet d'interagir avec des utilisateurs honnêtes, le transcript des échanges est ajouté à la liste Trans et chaque signature acceptée par les utilisateurs honnêtes est placée dans la liste Set.

Afin de gagner, l'adversaire doit produire une signature σ d'un message m de son choix, telle que toutes les conditions suivantes soient remplies :

- la signature est valide,
- la signature n'appartient pas à l'ensemble Set,
- il existe un transcript dans la liste Trans tel que :
  - l'autorité de révocation, à l'aide de l'algorithme $\mathsf{R_{sig}}(trans, sk_{\text{RA}})$ renvoie les valeurs (I_{sig}, π_1),
  - le juge accepte cet identifiant et la preuve, c'est-à-dire , $\mathsf{M_{sig}}(\text{I}_{sig}, m, \sigma, \pi_1) = 1$.

La signature émise par l'adversaire n'appartient donc pas à la liste des signatures émises par des utilisateurs honnêtes mais est cependant reliée à une exécution du protocole de signature avec un utilisateur honnête.

On décrit l'expérience de manière plus formelle à la figure 8.6.

8.7 Remarque sur la non-falsification supplémentaire

Comme pour l'indistinguabilité, cette propriété est la même que celle requise pour les signatures aveugles. L'adversaire est autorisé à interagir au plus l fois avec un signataire honnête et on dit qu'il met en défaut la *non-falsification supplémentaire* s'il arrive à produire à la fin $l+1$ signatures valides. Nous redéfinissons ici l'expérience en utilisant les notations et les algorithmes des signatures aveugles à anonymat révocable. Nous prenons en compte les algorithmes de traçabilité propres à ces signatures et donnons à l'adversaire accès aux clés de l'autorité de révocation.

À tout schéma de signature aveugle à anonymat révocable FBS, à tout adversaire $\mathcal{A}$ et à tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{Unforg}}_{\text{FBS},\mathcal{A}}(k)$ (*cf.* figure 8.7). L'avantage de

Expérience $\mathbf{Exp}^{\text{NonSigFram}}_{\text{FBS},\mathcal{A}}(k)$:

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k)$ $\quad$ $(pk_{\text{RA}}, sk_{\text{RA}}) \leftarrow \mathsf{G}_{\text{RA}}(1^k)$
Trans $\leftarrow \emptyset$; Set $\leftarrow \emptyset$
$(m, \sigma) \leftarrow \mathcal{A}(pk_{\mathcal{S}}, sk_{\mathcal{S}}, pk_{\text{RA}}, sk_{\text{RA}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}, \mathsf{User})$
`si` $\mathsf{Verify}(pk_{\mathcal{S}}, m, \sigma) = 0$ `alors` renvoie 0
`si` $\mathsf{R_{id}}(m, sk_{\mathcal{RA}}) = \text{I}_{id}$ et $(m, \sigma, \text{I}_{id}) \in$ Set `alors` renvoie 0
`si` $\exists trans \in$ Trans tel que $\mathsf{R_{sig}}(trans, sk_{\text{RA}}) = (\text{I}_{sig}, \pi_1)$
et $\mathsf{Match_{sig}}(\text{I}_{sig}, m, \sigma, \pi_1) = 1$ `alors` renvoie 1 `sinon` renvoie 0

FIG. 8.6 – Non-diffamation de signature

$\mathcal{A}$ est donné par

$$\mathbf{Adv}^{\text{Unforg}}_{\text{FBS},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{Unforg}}_{\text{FBS},\mathcal{A}}(k) = 1].$$

On dit que le schéma est *non-falsifiable* si la fonction $\mathbf{Adv}^{\text{Unforg}}_{\text{FBS},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).
L'expérience est décrite plus en détail dans la figure 8.7.

Expérience $\mathbf{Exp}^{\text{Unforg}}_{\text{FBS},\mathcal{A}}(k)$:

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k)$
$(pk_{\text{RA}}, sk_{\text{RA}}) \leftarrow \mathsf{G}_{\text{RA}}(1^k)$
$((m_1, \sigma_1), \ldots (m_{l+1}, \sigma_{l+1})) \leftarrow \mathcal{A}(pk_{\mathcal{S}}, pk_{\text{RA}}, sk_{\text{RA}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{USK}, \mathsf{Sign})$
`si` $\forall j \in [1, l+1], \mathsf{Verify}(pk_{\mathcal{S}}, m_j, \sigma_j) = 1$
et $(m_i, \sigma_i) \neq (m_j, \sigma_j)$ pour $1 \leq i < j \leq l+1$
et au plus l interactions avec Sign ont été engagées
renvoie 1 `sinon` renvoie 0

FIG. 8.7 – Non-falsification supplémentaire

Il s'avère que cette propriété est englobée dans la propriété de traçabilité de signature que nous avons décrite précédemment. En effet, l'adversaire n'a interagi que l fois avec le signataire, donc l'ensemble Set contient au plus l triplets $(m_i, \sigma_i, \text{Id}_{\mathcal{U}_i})$ pour $i \leq l$ et l'ensemble Trans contient au plus l transcripts $trans_i$ pour $i \leq l$. La signature finale produite (par exemple pour le message m_{l+1}) est notée σ_{l+1} et renvoie à un couple $(\text{I}d, \pi_2) = \mathsf{R_{id}}(m_{l+1}, \sigma_{l+1}, sk_{\text{RA}})$ tel que le triplet $(m_{l+1}, \sigma_{l+1}, \text{I}d)$ n'appartient pas à l'ensemble Set (dans le cas contraire, $\mathcal{A}$ n'a pas produit de falsification supplémentaire). On se trouve alors face aux cas suivants :

- il existe $j \in [1, l+1]$ tel que, pour tout $i \in [1, l]$ on ait $\mathsf{R_{sig}}(trans_i, sk_{\mathcal{RA}}) = (\text{I}_{sig_i}, \pi_{1,i})$ et $\mathsf{Match_{sig}}(\text{I}_{sig_i}, m_j, \sigma_j, \pi_{1,i}) = 0$. L'adversaire a alors cassé la propriété de traçabilité de signature.

- inversement, pour tout $j \in [1, l+1]$, il existe un $i_j \in [1, l]$ tel que $\mathsf{R}_{\mathsf{sig}}(trans_{i_j}, sk_{\mathcal{RA}}) = (\mathrm{I}_{sig_{i_j}}, \pi_{1,i_j})$ et $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig_{i_j}}, m_j, \sigma_j, \pi_{1,i_j}) = 1$. Il existe donc deux entiers j_1 et j_2 tels $i_{j_1} = i_{j_2}$. Donc les deux couples message/signature (m_{j_1}, σ_{j_1}) et (m_{j_2}, σ_{j_2}) correspondent à la même session. Dans ce cas aussi, l'adversaire a cassé la propriété de traçabilité de signature.

Chapitre 9

Nouveau schéma de signature aveugle à anonymat révocable

Nous présentons dans ce chapitre, un nouveau schéma de signature aveugle à anonymat révocable et prouvons sa sécurité dans le modèle introduit au chapitre précédent.

Ce nouveau protocole a été présenté à la conférence *Pairing 07* [HT07]. Sa construction s'inspire du schéma de signature de groupe de Boneh, Boyen et Sacham décrit au chapitre 4 et par conséquent utilise des applications bilinéaires.

Sommaire

9.1 Description du schéma

La signature de groupe BBS permet à un utilisateur d'obtenir un certificat de membre auprès d'un manager de groupe. L'utilisateur prouve ensuite son appartenance au groupe en prouvant sa connaissance d'un certificat valide. Sa signature est alors en réalité une preuve de connaissance (ou « signature de connaissance » en utilisant la terminologie de Camenisch et Stadler [CS97]).

Nous basons notre schéma sur ce même principe. L'utilisateur $\mathcal{U}$ et le signataire $\mathcal{S}$ établissent dans une première étape une signature du message, de la même façon

que l'utilisateur et le manager de groupe construisent un certificat. Par la suite, étant donné que la signature obtenue n'est pas complètement aveugle (le signataire n'a pas connaissance du message, mais de certaines parties de la signature), l'utilisateur va publier une preuve de connaissance de cette signature.

Afin d'autoriser la double révocation, nous ajoutons deux valeurs dans le processus décrit ci-dessus. La première est calculée conjointement par $\mathcal{U}$ et $\mathcal{S}$ durant la première phase et est conservée par le signataire qui pourra la transmettre au juge le cas échéant. La deuxième valeur est calculée par l'utilisateur uniquement et ajoutée à la signature. Elle permettra le traçage d'identité. Nous utilisons le double chiffrement ElGamal pour construire ces valeurs. En effet, comme nous utilisons des applications bilinéaires dans notre construction, ce chiffrement a la propriété d'être IND-CCA2 si on suppose l'hypothèse XDH et si le chiffrement contient une preuve d'égalité des chiffrés [FP01]. Il est tout à fait possible de ne pas utiliser l'hypothèse XDH en remplaçant le double chiffrement ElGamal par un double chiffrement linéaire [BBS04] qui est IND-CCA2 (*cf.* section 3.2). Pour des raisons de commodité, nous avons choisi ici le double chiffrement ElGamal (notamment pour limiter la taille de notre signature).

9.1.1 Mise en place du protocole : Setup

Avant que toute procédure soit initialisée, le signataire $\mathcal{S}$ et l'autorité de révocation reçoivent les paramètres publics du schéma, tels que décrits ci-dessous. Ils reçoivent ensuite leurs clés privées et publiques respectives calculées à l'aide des algorithmes $\mathsf{G}_{\mathcal{S}}$ et G_{RA}.

Paramètres du système Les paramètres du système sont donnés par les valeurs suivantes :

- $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_{\mathrm{T}}$ sont trois groupes d'ordre p premier,
- g_1 est un générateur de $\mathbb{G}_1$ et g_2 un générateur de $\mathbb{G}_2$,
- ψ est un isomorphisme calculable de $\mathbb{G}_2$ dans $\mathbb{G}_1$ tel que $\psi(g_2) = g_1$,
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_{\mathrm{T}}$ est une application bilinéaire admissible,
- h_1, h_2, h_3, h_4 sont des éléments choisis aléatoirement dans $\mathbb{G}_1$ tels que $\log_{h_j} h_i$ pour $i \neq j$ soient inconnus de tous, et $h_2 = g_1^{\chi}$ pour une valeur (gardée secrète) $\chi \in_{\mathrm{R}} \mathbb{Z}_p$,
- l_p est la taille de p,
- H est une fonction de hachage : $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p$,
- Paramètres du chiffrement de Paillier : soient p et q deux entiers premiers aléatoires, distincts et de même taille tels que $\mathrm{pgcd}(\mathsf{pq}, (\mathsf{p}-1)(\mathsf{q}-1)) = 1$. Soient $\mathsf{n} = \mathsf{pq}$ et $\mathsf{g} = (1+\mathsf{n})$. La clé publique est (n, g) et la clé privée est (p, q). On suppose aussi que $|\mathsf{n}| \geq l_p$.

$\mathsf{G}_{\mathcal{S}}$ **:** le signataire utilise l'algorithme $\mathsf{G}_{\mathcal{S}}$ afin d'obtenir sa clé privée et sa clé publique :

- $sk_{\mathcal{S}} := \gamma \in_{\mathrm{R}} \mathbb{Z}_p$,
- $pk_{\mathcal{S}} := g_2^{\gamma} = \Gamma$.

G_{RA} **:** l'autorité de révocation utilise l'algorithme G_{RA} afin d'obtenir sa clé privée et sa clé publique :

- $sk_{\mathrm{RA}} = (\xi_1, \xi_2) \in_{\mathrm{R}} (\mathbb{Z}_p^*)^2$,
- $pk_{\mathrm{RA}} = (u, v) \in \mathbb{G}_1^2$ tels que $u = g_1^{\xi_1}$ et $v = g_1^{\xi_2}$.

$\mathsf{G}_{\mathcal{U}}$ **:** l'utilisateur utilise l'algorithme $\mathsf{G}_{\mathcal{U}}$ afin d'obtenir sa clé privée et sa clé publique qui fera également office d'identifiant (que nous appellerons, par abus de langage, identité) :

- $sk_{\mathrm{Id_U}} = x_u \in_{\mathrm{R}} \mathbb{Z}_p^*$,
- $pk_{\mathrm{Id}_{\mathcal{U}}} = \mathrm{Id}_{\mathcal{U}} = h_2^{x_u}$.

Les paramètres publics sont dès lors les suivants :

$$params = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, e, \psi, p, l_p, g_1, g_2, h_1, h_2, h_3, h_4, \mathsf{g}, \mathsf{n}, pk_{\mathcal{S}}, pk_{\mathrm{RA}}\}.$$

Initialisation et enregistrement de l'utilisateur Lorsqu'un utilisateur souhaite obtenir une signature aveugle pour la première fois, il doit demander tout d'abord les paramètres du système au signataire. À l'aide de ceux-ci il est capable de construire son identité et sa clé privée. Par la suite, à chaque fois qu'il souhaite obtenir une signature, il devra s'authentifier au préalable auprès du signataire en envoyant un chiffrement de Paillier de son identifiant, ainsi qu'une preuve de connaissance de la valeur x_u, sa clé privée. Le signataire devra conserver le chiffrement dans une table d'identités $\mathsf{Tab}_{\mathsf{id}}$. $\mathcal{U}$ n'aura ainsi pas besoin de recalculer un nouvel identifiant à chaque fois, mais il devra toutefois s'authentifier à chaque nouvelle requête de signature.

9.1.2 Signature d'un message : BSign

Nous décrivons maintenant la phase de signature à proprement parler, c'est-à-dire le protocole Bsign. Tout au long de l'exécution du protocole, $\mathcal{S}$ enregistre toutes les communications dans une base Trans en les associant à l'utilisateur $\mathcal{U}$ avec qui il est en train d'interagir. Comme nous l'avons dit, cette phase se déroule en deux étapes.

Première étape du protocole Dans cette étape, $\mathcal{U}$ et $\mathcal{S}$ interagissent afin de construire une signature sur un message m choisi par $\mathcal{U}$.

1. $\mathcal{U}$ commence par choisir deux valeurs aléatoires r et s' (sa participation à un secret partagé) puis calcule un engagement C de son message m et de ces deux valeurs. Il chiffre ensuite à l'aide du chiffrement de Paillier les valeurs m, r et s'.
2. Il calcule ensuite Δ_1', un double chiffrement ElGamal de la valeur $h_1^{s'}$. Ce chiffrement est ensuite utilisé par $\mathcal{S}$ pour construire la valeur nécessaire au traçage de signature.
3. $\mathcal{U}$ envoie à $\mathcal{S}$ toutes les valeurs qu'il a calculées et prouve la connaissance de leur représentation en base (h_1, h_3, h_4) et leur consistance en utilisant une preuve de connaissance que l'on peut convertir en signature de connaissance (à l'aide de l'heuristique de Fiat-Shamir).

4. Le signataire vérifie la validité de la preuve et signe l'engagement C en utilisant la technique du protocole de Join de [BBS04] et sa clé secrète γ. Pour ce faire, il choisit aléatoirement s'', sa participation au secret partagé, et une valeur x. Il calcule aussi un double chiffrement ElGamal Δ_1 de la valeur $h_1^{s'+s''}$ à l'aide du chiffrement Δ_1' envoyé par $\mathcal{U}$. $\mathcal{S}$ enregistre la valeur Δ_1 dans la liste des identifiants construite pendant la phase d'enregistrement, en l'associant à l'utilisateur $\mathcal{U}$. $\mathcal{S}$ ne peut pas construire de preuve de consistance des chiffrés dans Δ_1 (il ne connaît par la valeur $h_1^{s'+s''}$) mais le chiffrement Δ_1 ainsi que la preuve donnée par $\mathcal{U}$ permettront néanmoins au juge de vérifier l'authenticité de ce chiffrement.
5. Finalement, $\mathcal{S}$ envoie à $\mathcal{U}$ la signature de C, ainsi que s'' et x.

Cette première étape est décrite de manière plus précise dans la figure 9.1.

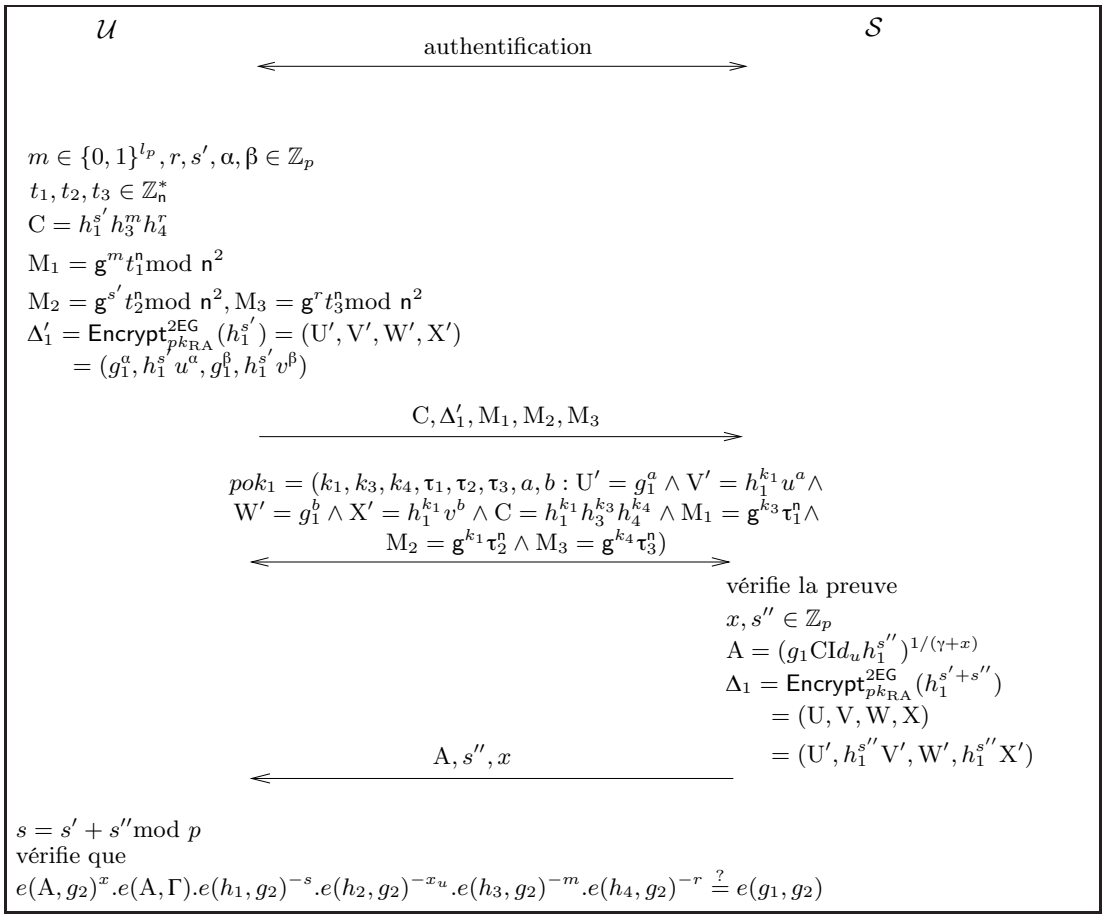

FIG. 9.1 – Première étape du schéma FBS

Dans la preuve de connaissance notée pok_1, $\mathcal{U}$ prouve :
- qu'il connaît les valeurs r, s' et m,
- que m, r et s' sont les mêmes dans l'engagement C et les chiffrements de Paillier M_1, M_2 et M_3,

- que la valeur s' chiffrée dans Δ_1' est la même que dans C.

Deuxième étape du protocole Après avoir reçu les valeurs A, s'' et x, l'utilisateur est en possession de toutes les données dont il a besoin pour construire sa signature. Cependant, le signataire connaît les valeurs A, s'' et x. C'est pourquoi, l'utilisateur ne peut pas les révéler dans sa signature mais peut seulement prouver qu'il les connaît. De plus, il doit prouver aussi qu'il connaît les valeurs r, x_u et m qu'il a lui-même choisies.

Afin de pouvoir être traçable, l'utilisateur ajoute deux valeurs à sa signature :
- h_1^s pour le traçage d'identité et
- $\Delta_2 = \mathsf{Encrypt}^{\mathtt{2EG}}_{pk_{\mathrm{RA}}}(\mathrm{Id}_{\mathcal{U}}) = (g_1^a, h_2^{x_u}u^a, g_1^b, h_2^{x_u}v^b)$, avec a et $b \in_{\mathrm{R}} [0, p-1]$ pour le traçage de signature.

Il doit également prouver que la valeur s de h_1^s est la même que dans la signature et que Δ_2 est bien un chiffrement de son identité.

Une signature valide d'un message m consiste donc en :
- la valeur h_1^s,
- un double chiffrement ElGamal de l'identité,
- une signature de connaissance pok_2 qui prouve que $\mathcal{U}$ connaît des valeurs (A, x, $\mathsf{x_u}$, s, r) telles que :

(P_1) : $e(\mathsf{A}, g_2)^{\mathsf{x}} \cdot e(\mathsf{A}, \Gamma) \cdot e(h_1, g_2)^{-\mathsf{s}} \cdot e(h_2, g_2)^{-\mathsf{x_u}} \cdot e(h_3, g_2)^{-m} \cdot e(h_4, g_2)^{-\mathsf{r}} = e(g_1, g_2)$

(P_2) : $\mathsf{x_u}$ vaut $\log_{h_2} \mathrm{Id}_{\mathcal{U}}$, avec Δ_2 le chiffrement de $\mathrm{Id}_{\mathcal{U}}$.

(P_3) : s est bien égal à $\log_{h_1} h_1^s$

Afin de détailler la signature de connaissance P, nous montrons d'abord comment l'utilisateur peut prouver qu'il connaît les valeurs $(\mathrm{A}, x, x_u, s, r)$ et que Δ_2 est bien construit à l'aide d'une preuve de connaissance à divulgation nulle de connaissance. Puis, à l'aide de l'heuristique de Fiat-Shamir, nous transformons cette preuve en signature.

1. Tout d'abord, $\mathcal{U}$ chiffre $\mathrm{Id}_{\mathcal{U}}$ de la manière suivante

$$\Delta_2 = \mathtt{Encrypt}^{\mathtt{2EG}}_{pk_{\mathcal{RA}}}(\mathrm{Id}_{\mathcal{U}}) = (\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4) = (g_1^a, h_2^{x_u}u^a, g_2^b, h_2^{x_u}v^b).$$

   On pose, pour la suite de la preuve, $h_1^s = \mathrm{T}_5$.

2. Il choisit ensuite $\alpha \in_{\mathrm{R}} [0, p-1]$ et calcule un chiffrement ElGamal de A

$$\mathrm{T} = \mathrm{A}h_2^{\alpha}, \qquad \tilde{\mathrm{T}} = g_1^{\alpha}$$

3. Il doit ensuite prouver qu'il existe α, a et b tels que

$$\begin{aligned} \mathrm{T} &= \mathrm{A}h_2^{\alpha}, \\ \tilde{\mathrm{T}} &= g_1^{\alpha}, \\ \mathrm{T}_1 &= g_1^a, \\ \mathrm{T}_2 &= h_2^{x_u}u^a, \\ \mathrm{T}_3 &= g_1^b, \\ \mathrm{T}_4 &= h_2^{x_u}v^b. \end{aligned}$$

T masque correctement les valeurs (A, x, x_u, r) si et seulement s'il existe un α tel que

$$e(\mathrm{T}, g_2)^x \cdot e(h_2, \Gamma)^{-\alpha} \cdot e(h_1, g_2)^{-s} \cdot e(h_2, g_2)^{-x_u - z} \cdot e(h_4, g_2)^{-r} = \frac{e(g_1, g_2) \cdot e(h_3, g_2)^m}{e(\mathrm{T}, \Gamma)}$$

et $\tilde{\mathrm{T}}^x g_1^{-z} = 1$, avec $z := x\alpha$.

4. $\mathcal{U}$, qui est ici le prouveur dans notre preuve interactive, choisit r_α, r_s, r_{x_u}, r_x, r_z, r_r, r_a, r_b et calcule

$$\begin{aligned}
\mathrm{T}'_1 &= g_1^{r_a}, \\
\mathrm{T}'_2 &= h_2^{r_{x_u}} u^{r_a}, \\
\mathrm{T}'_3 &= g_1^{r_b}, \\
\mathrm{T}'_4 &= h_2^{r_{x_u}} v^{r_b}, \\
\mathrm{T}'_5 &= h_1^{r_s}, \\
\tilde{\mathrm{T}}' &= g^{r_a}, \\
\bar{\mathrm{T}} &= \tilde{\mathrm{T}}^{r_x} g_1^{-r_z},
\end{aligned}$$

$$\mathrm{T}' = e(\mathrm{T}, g_2)^{r_x} \cdot e(h_2, \Gamma)^{-r_\alpha} \cdot e(h_1, g_2)^{-r_s} e(h_2, g_2)^{-r_{x_u} - r_z} \cdot e(h_4, g_2)^{-r_r}.$$

5. Le prouveur envoie ensuite les valeurs (T_1, T_2, T_3, T_4, T_5, T, $\tilde{\mathrm{T}}$, T'_1, T'_2, T'_3, T'_4, T'_5, T′, $\tilde{\mathrm{T}}'$, $\bar{\mathrm{T}}$) au vérificateur (qui est ici le destinataire de la signature).
6. Le vérificateur choisit un challenge $c \in_{\mathrm{R}} \mathbb{Z}_p$ et le renvoie au prouveur.
7. Celui-ci calcule et renvoie les valeurs

$$\begin{aligned}
s_\alpha &= r_\alpha + c\alpha, \quad & s_s &= r_s + cs, \quad & s_x &= r_x + cx, \quad & s_{x_u} &= r_{x_u} + cx_u, \\
s_r &= r_r + cr, \quad & s_z &= r_z + cz, \quad & s_a &= r_a + ca, \quad & s_b &= r_b + cb.
\end{aligned}$$

8. Afin de vérifier la preuve P, le vérifieur vérifie les équations suivantes :

$$\begin{aligned}
g_1^{s_a} &= \mathrm{T}'_1 \mathrm{T}_1^c, \\
h_2^{s_{x_u}} u^{s_a} &= \mathrm{T}'_2 \mathrm{T}_2^c, \\
g_1^{s_b} &= \mathrm{T}'_3 \mathrm{T}_3^c, \\
h_2^{s_{x_u}} v^{s_b} &= \mathrm{T}'_4 \mathrm{T}_4^c, \\
h_1^{s_s} &= \mathrm{T}'_5 . \mathrm{T}_5^c, \\
g_1^{s_\alpha} &= \tilde{\mathrm{T}}' \tilde{\mathrm{T}}^c, \\
\tilde{\mathrm{T}}^{s_x} g_1^{-s_z} &= \bar{\mathrm{T}},
\end{aligned}$$

$$\begin{aligned}
&e(\mathrm{T}, g_2)^{s_x} \cdot e(h_2, \Gamma)^{-s_\alpha} \cdot e(h_1, g_2)^{-s_s} \cdot e(h_2, g_2)^{-s_{x_u} - s_z} \cdot e(h_4, g_2)^{-s_r} \\
&\qquad = \mathrm{T}' \cdot \Big(\frac{e(g_1, g_2) \cdot e(h_3, g_2)^m}{e(\mathrm{T}, \Gamma)} \Big)^c.
\end{aligned}$$

La transformation de la preuve de connaissance à divulgation nulle de connaissance en signature de connaissance s'effectue de la façon suivante :

1. $\mathcal{U}$ choisit les valeurs $r_a, r_s, r_{x_u}, r_x, r_z, r_r, r_a, r_b$ et calcule

$$\begin{aligned} \mathrm{T}'_1 &= g_1^{r_a}, \\ \mathrm{T}'_2 &= h_2^{r_{x_u}} u^{r_a}, \\ \mathrm{T}'_3 &= g_1^{r_a}, \\ \mathrm{T}'_4 &= h_2^{r_{x_u}} v^{r_b}, \\ \mathrm{T}'_5 &= h_1^{r_s}, \end{aligned}$$

$$\mathrm{T}' = e(\mathrm{T}, g_2)^{r_x} \cdot e(h_2, \Gamma)^{-r_a} \cdot e(h_1, g_2)^{-r_s} e(h_2, g_2)^{-r_{x_u} - r_z} \cdot e(h_4, g_2)^{-r_r}.$$

2. Il calcule lui même le challenge c à l'aide de la fonction de hachage H :

$$c := \mathrm{H}(m, \mathrm{T}, \tilde{\mathrm{T}}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, \mathrm{T}_5, \mathrm{T}'_1, \mathrm{T}'_2, \mathrm{T}'_3, \mathrm{T}'_4, \mathrm{T}'_5, \mathrm{T}', \tilde{\mathrm{T}}', \bar{\mathrm{T}}).$$

3. Sa signature est alors

$$\sigma = (m, h_1^s, \Delta_2, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, \mathrm{T}, \tilde{\mathrm{T}}, s_a, s_x, s_{x_u}, s_s, s_r, s_z, s_a, s_b).$$

Notre signature étant une signature de connaissance, nous considérerons que deux signatures sont différentes si leurs premières composantes (à savoir les valeurs h_1^s) sont différentes. En effet, l'utilisateur a la possibilité de créer autant de signatures de connaissance qu'il veut des valeurs $(\mathrm{A}, x, x_u, s, r)$ en n'ayant interagi qu'une seule fois avec le signataire. Nous allégeons cette contrainte dans nos preuves en considérant que deux signatures sont différentes si l'un des secrets sous-jacents (à savoir A, x, x_u, s, r ou le message m) diffère entre les deux signatures émises.

9.1.3 Vérification de la signature : Verify

Le destinataire reçoit un couple (m, σ) où σ est une signature de m de la forme $(h_1, \Delta_2, \mathrm{P})$. Il vérifie la validité de la signature de connaissance P qui prouve que m a été correctement signé par le signataire. Il est aussi en possession de la valeur de traçage d'identité Δ_2 et est convaincu, par la signature de connaissance, que cette valeur correspond à un chiffré de l'identité de l'utilisateur.

9.1.4 Révocations : Revoc

Selon que l'autorité cherche à tracer une signature ou une identité, la révocation se fait de manière différente.

Traçage d'identité : $\mathsf{R_{id}}, \mathsf{Match_{id}}$ Étant donné une signature $(h_1^s, \Delta_2, \mathrm{P})$, l'autorité de révocation déchiffre la valeur Δ_2 et retrouve $\mathrm{Id}_{\mathcal{U}}$.

$$\Delta_2 = (\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, pok) = (g_1^a, h_2^{x_u} u^a, g_1^b, h_2^{x_u} v^b, pok(\log_{h_2} \mathrm{T}_2 = \log_{h_2} \mathrm{T}_4)).$$

L'autorité calcule $\mathrm{R}_1 = \mathrm{T}_1^{\xi_1} = g_1^{a\xi_1}$ puis $\mathrm{Id}_{\mathcal{U}} = \mathrm{T}_2/\mathrm{R}_1$ et $\mathrm{R}_2 = \mathrm{T}_3^{\xi_2} = g_1^{b\xi_2}$ puis $\mathrm{Id}'_{\mathcal{U}} = \mathrm{T}_4/\mathrm{R}_2$ et vérifie que $\mathrm{Id}_{\mathcal{U}} = \mathrm{Id}'_{\mathcal{U}}$ et que la preuve est exacte.

Pour prouver au juge que ce déchiffrement a été fait honnêtement, l'autorité de révocation prouve que $\log_{\mathrm{T}_1}(\mathrm{R}_1) = \log_{g_1} u$ ou que $\log_{\mathrm{T}_3}(\mathrm{R}_2) = \log_{g_1} v$.

Le juge reçoit la valeur $\mathrm{Id}_{\mathcal{U}}$ et la preuve associée. Il n'a plus qu'à retrouver l'utilisateur à qui correspond cette identité dans la liste produite par le signataire au moment de l'enregistrement.

Traçage de signature : $\mathsf{R_{sig}}, \mathsf{Match_{sig}}$ Étant donné le transcript d'une session, l'autorité de révocation est capable de tracer une signature en déchiffrant la valeur Δ_1 calculée dans la première phase du protocole BSign :

$$h_1^s = \mathsf{Decrypt}_{rsk}^{2\mathtt{EG}}(g_1^\alpha, h_1^s u^a, g_1\beta, h_1^s v^\beta).$$

La valeur h_1^s obtenue dans ce déchiffrement correspond à la valeur h_1^s que l'utilisateur doit révéler lorsqu'il rend publique la signature qu'il a obtenue (*cf.* étape 2 du protocole BSign). L'autorité prouve que son déchiffrement est honnête de la même manière que précédemment.

Grâce à h_1^s, le juge est en mesure de tracer la signature obtenue par l'utilisateur $\mathcal{U}$ (la signature recherchée sera celle dont la première composante est h_1^s).

9.2 Preuves de sécurité

9.2.1 Consistance

Afin de vérifier la consistance de notre schéma, nous devons vérifier que l'équation de vérification de notre schéma est valide.

$$\begin{aligned}
e(\mathrm{A}, g_2)^x &= e((g_1 \mathrm{CI} d_u h_1^{s''})^{1/(\gamma+x)}, g_2)^x \\
&= e((g_1 h_1^s h_2^{x_u} h_3^m h_4^r)^{1/\gamma+x}, g_2)^x \\
&= e(g_1, g_2)^{x/\gamma+x} \cdot e(h_1, g_2)^{sx/\gamma+x} \cdot e(h_2, g_2)^{x_u x/\gamma+x} \cdot \\
&\quad e(h_3, g_2)^{mx/\gamma+x} \cdot e(h_4, g_2)^{rx/\gamma+x} \\
e(\mathrm{A}, \Gamma) &= e((g_1 \mathrm{CI} d_u h_1^{s''})^{1/(\gamma+x)}, g_2^\gamma) \\
&= e((g_1 h_1^s h_2^{x_u} h_3^m h_4^r)^{1/\gamma+x}, g_2^\gamma) \\
&= e(g_1, g_2)^{\gamma/\gamma+x} \cdot e(h_1, g_2)^{s\gamma/\gamma+x} \cdot e(h_2, g_2)^{x_u\gamma/\gamma+x} \cdot \\
&\quad e(h_3, g_2)^{m\gamma/\gamma+x} \cdot e(h_4, g_2)^{r\gamma/\gamma+x} \\
e(\mathrm{A}, g_2)^x \cdot e(\mathrm{A}, \Gamma) &= e(g_1, g_2)^{(\gamma+x)/\gamma+x} \cdot e(h_1, g_2)^{s(\gamma+x)/\gamma+x} \cdot e(h_2, g_2)^{x_u(\gamma+x)/\gamma+x} \cdot \\
&\quad e(h_3, g_2)^{m(\gamma+x)/\gamma+x} \cdot e(h_4, g_2)^{r(\gamma+x)/\gamma+x} \\
&= e(g_1, g_2) \cdot e(h_1, g_2)^s \cdot e(h_2, g_2)^{x_u} \cdot e(h_3, g_2)^m \cdot e(h_4, g_2)^r
\end{aligned}$$

En reprenant l'équation dans sa totalité on obtient alors :

$$e(\mathrm{A}, g_2)^x \cdot e(\mathrm{A}, \Gamma) \cdot e(h_1, g_2)^{-s} \cdot e(h_2, g_2)^{-x_u} \cdot e(h_3, g_2)^{-m} \cdot e(h_3, g_2)^{-r} = e(g_1, g_2) \cdot e(h_1, g_2)^{s} \cdot e(h_2, g_2)^{x_u} \cdot e(h_3, g_2)^{m} \cdot e(h_4, g_2)^{r} \cdot e(h_1, g_2)^{-s} \cdot e(h_2, g_2)^{-x_u} \cdot e(h_3, g_2)^{-m} \cdot e(h_3, g_2)^{-r} = e(g_1, g_2)$$

9.2.2 Sécurité du schéma BBS étendu

Nous montrons ici que le schéma de signature BBS étendu présenté à la section 3.4.2 est inforgeable contre des attaques à messages choisis sous l'hypothèse que le schéma de signature BBS l'est aussi.

La sécurité du schéma de signature BBS étendu se montre en réduisant la sécurité de celui-ci au schéma de signature BBS prouvé sûr sous l'hypothèse Diffie-Hellman fort.

Inforgeabilité Soit $\mathcal{A}$ un adversaire contre l'inforgeabilité du schéma BBS étendu qui réussit avec une probabilité de succès non négligeable. A l'aide de l'adversaire $\mathcal{A}$, nous construisons un algorithme $\mathcal{B}$ qui s'attaque à l'inforgeabilité du schéma de signature BBS. Il a donc accès, s'il le souhaite, à un oracle de signature BBS $\mathsf{Sign}_{\mathrm{BBS}_l^{ext}}$.

$\mathcal{B}$ construit les paramètres du système et donne à $\mathcal{A}$ les paramètres publics, c'est-à-dire $(\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e, h_1, \ldots h_l, p, pk)$. $\mathcal{A}$ interagit ensuite avec $\mathcal{B}$ pour obtenir des signatures sur des messages de son choix $\mathrm{M}_1 = (m_1^1, \ldots, m_l^1)$, $\mathrm{M}_2 = (m_1^2, \ldots m_l^2)$, ..., $\mathrm{M}_{q_s} = (m_1^{q_s}, \ldots m_l^{q_s})$. Il reçoit en réponse les signatures associées (A_1, x_1), ..., $(\mathrm{A}_{q_s}, x_{q_s})$.

Au final, $\mathcal{A}$ renvoie sa forge $(\mathrm{A}, x, \mathrm{M} = (m'_1, \ldots, m'_l))$. On distingue alors deux types de forgeurs :

- un forgeur qui renvoie une signature telle que $(\mathrm{A}, x) \neq (\mathrm{A}_i, x_i)$ pour $i \in \{1, \ldots, q_s\}$ et qu'on appelle forgeur de Type 1,
- un forgeur qui renvoie une signature telle que $(\mathrm{A}, x) = (\mathrm{A}_i, x_i)$ pour un $i \in \{1, \ldots, \mathsf{q}_s\}$ et $(m'_1, \ldots, m'_l) \neq (m_1^i, \ldots m_l^i)$, où q_S est le nombre de signatures demandées par $\mathcal{A}$ et qu'on appelle forgeur de Type 2.

Chacun des deux forgeurs peut être utilisé pour construire une forge d'une signature BBS. Cependant, la réduction se fait différemment selon le type de forgeur. C'est pourquoi, au début de l'attaque, $\mathcal{B}$ tire un bit aléatoire $c_{mode} \in \{0, 1\}$ qui représente son choix pour le type de forge que l'adversaire $\mathcal{A}$ va renvoyer.

Cas $c_{mode} = 0$:

- $\mathcal{B}$ reçoit de son challenger les paramètres publics d'une signature BBS, à savoir $(\mathbb{G}_1, \mathbb{G}_2, e, g_1, g_2, pk = g_2^\gamma, h, p)$. À partir de ces valeurs, il est capable de construire des paramètres publics pour l'adversaire $\mathcal{A}$. Il pose $h_1 = h$ et pour $i \in [2, l]$, il pose $h_i = h^{r_i}$ où $r_i \in_\mathrm{R} \mathbb{Z}_p^*$. Il envoie alors à $\mathcal{A}$ les paramètres publics suivants : $(\mathbb{G}_1, \mathbb{G}_2, e, g_1, g_2, pk = g_2^\gamma, p, h_1, \ldots, h_l)$.
- Quand $\mathcal{A}$ envoie une requête de signature à $\mathcal{B}$ sur un message $(m_1, \ldots, m_l)$, $\mathcal{B}$ calcule d'abord un message $\mathrm{M} = m_1 + m_2 r_2 + \cdots + m_l r_l$ et fait une requête à son oracle de signature en donnant M en entrée. Il reçoit en réponse une signature (A, x) qu'il renvoie à $\mathcal{A}$. (A, x) est une signature BBS étendue valide pour le message $m = (m_1, \ldots, m_l)$.

- Finalement, $\mathcal{A}$ renvoie sa forge (A', x') sur un message $m' = (m'_1, \ldots, m'_l)$.
- À l'aide de ces valeurs, $\mathcal{B}$ est capable de produire directement une forge sur le message $\mathrm{M}' = m'_1 + m' - 2r_2 \cdots + m'_l r_l$.
- $\mathcal{B}$ a donc réussi à casser l'inforgeabilité du schéma BBS.

Cas $c_{mode} = 1$: $\mathcal{B}$ s'attaque directement au problème du logarithme discret. Il reçoit donc de son challengeur une valeur $h \in \mathbb{G}_1$ et son but est de trouver $\log_{g_1} h$.

- B choisit une valeur $\gamma \in_{\mathrm{R}} \mathbb{Z}_p^*$ et calcule la clé publique $pk = g_2^{\gamma}$. Il choisit un $\mathrm{I} \in [1, l]$ et pose $h_{\mathrm{I}} = h$ et pour $i \neq \mathrm{I}$, $h_i = g^{r_i}$ où $r_i \in_{\mathrm{R}} [0, p-1]$. Il envoie alors à $\mathcal{A}$ les paramètres publics d'un schéma de signature BBS étendu $(\mathbb{G}_1, \mathbb{G}_2, e, g_1, g_2, pk = g_2^{\gamma}, p, h_1, \ldots, h_l)$.
- Lorsque $\mathcal{A}$ envoie une requête à $\mathcal{B}$ sur un message $m = (m_1, \ldots, m_l)$ de son choix, $\mathcal{B}$ est capable de jouer directement le rôle de signataire grâce à sa connaissance de la clé privée γ.
- Finalement, $\mathcal{A}$ renvoie sa forge (A, x) sur un message $m' = (m'_1, \ldots, m'_l)$. Par hypothèse, le couple (A, x) est égal à une des réponses faites par $\mathcal{B}$ aux requêtes de $\mathcal{A}$, disons à la i^{eme} requête, mais pour un message différent, c'est-à-dire que $m' = (m'_1, \ldots, m'_l) \neq m_i = (m_1^i, \ldots, m_l^i)$. Les messages m et m' comportent au moins deux blocs différents. Si m et m' différaient d'un seul bloc, par exemple $m_i \neq m'_i$, cela impliquerait, étant donné que $(\mathrm{A}, x) = (\mathrm{A}_i, x_i)$, que $m_i = m'_i$.
- Avec probabilité $2/l$, $m_{\mathrm{I}}^i \neq m'_{\mathrm{I}}$. En utilisant cette inégalité et l'équation de vérification, $\mathcal{B}$ est capable de calculer le logarithme discret de $h_{\mathrm{I}} = h$ en base g_1 et obtient que
$$\log_{g_1} h = g_1^{(\Sigma_{j \neq \mathrm{I}}^{\mathrm{L}} r_j (m'_j - m_j^i))/(m_{\mathrm{I}}^i - m'_{\mathrm{I}})}$$
avec probabilité ε/l si ε est la probabilité pour $\mathcal{A}$ de casser l'inforgeabilité du schéma de signature BBS étendu.
- Si $\mathcal{B}$ est capable de résoudre le problème du logarithme discret, alors il est capable de casser l'inforgeabilité du schéma de signature BBS. Par exemple, il peut retrouver la clé secrète de signature en retrouvant le logarithme discret de la clé publique.

La probabilité de trouver le bon scénario, c'est-à-dire le type de forge que $\mathcal{A}$ va produire est de 1/2, donc en se plaçant dans le cas le plus pessimiste (c'est-à-dire le cas $c_{mode} = 1$), $\mathcal{B}$ peut alors casser l'inforgeabilité du schéma de signature BBS avec probabilité $\varepsilon/2l$.

9.2.3 Preuves de connaissance

Lemme 6 (Preuves de connaissance à divulgation nulle de connaissance)**.** *Les preuves de connaissance interactives pok_1 et pok_2 du schéma de signature aveugle à anonymat révocable sont des preuves de connaissance à divulgation nulle de connaissance face à un vérificateur honnête.*

La première preuve pok_1 est une « composition » de preuves de connaissance de représentation dans des groupes d'ordre connu ou inconnu. Leurs sécurités ont été détaillées et montrées dans [Sch91],[Poi96], [PS98]. Nous nous référons à ces démonstrations pour

affirmer que la preuve pok_1 est à divulgation nulle de connaissance face à un vérificateur honnête.

Nous détaillons la preuve pour la preuve pok_2.

Démonstration. **Significativité** : afin de prouver la significativité de la preuve, nous montrons l'existence d'un extracteur pour le protocole. Soit $\mathcal{A}$ une PPTM dont la probabilité de succès est non négligeable, c'est-à-dire qu'après s'être engagée sur $(\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, \mathrm{T}_5, \mathrm{T}, \tilde{\mathrm{T}}, \mathrm{T}'_1, \mathrm{T}'_2, \mathrm{T}'_3, \mathrm{T}'_4, \mathrm{T}'_5, \mathrm{T}', \tilde{\mathrm{T}}')$, elle répond correctement et avec probabilité non négligeable aux deux challenges c et c' par $(s_\alpha, s_s, s_x, s_{x_u}, s_r, s_z, s_a, s_b)$ et $(s'_\alpha, s'_s, s'_x, s'_{x_u}, s'_r, s'_z, s'_a, s'_b)$ respectivement.
On note $c'' = c - c'$. c'' est non nul modulo p étant donné que $c \neq c'$ et que $c, c' \in_{\mathrm{R}} \mathbb{Z}_p$ par définition. Pour $i \in \{\alpha, s, x, x_u, r, z, a, b\}$, on note $s''_i = s_i - s'_i$.
On obtient alors les égalités suivantes :

$$\begin{aligned} g_1^{s''_a} &= \mathrm{T}_1^{c''}, \\ g_1^{s''_{x_u}} u^{s''_a} &= \mathrm{T}_2^{c''}, \\ g_1^{s''_b} &= \mathrm{T}_3^{c''}, \\ g_1^{s''_{x_u}} v^{s''_b} &= \mathrm{T}_4^{c''}, \\ h_1^{s''_s} &= \mathrm{T}_5^{c''}, \\ \tilde{\mathrm{T}}^{c''} &= g_1^{s''_\alpha}, \end{aligned}$$

et

$$e(\mathrm{T}, g_2)^{s''_x} \cdot e(h_2, \Gamma)^{-s''_\alpha} \cdot e(h_1, g_2)^{-s''_s} \cdot e(h_2, g_2)^{-s''_{x_u} - s''_z} \cdot e(h_4, g_2)^{-s''_r} = \Big(\frac{e(g_1, g_2) \cdot e(h_3, g_2)^m}{e(\mathrm{T}, \Gamma)} \Big)^{c''}.$$

L'égalité $g_1^{s_\alpha} = \tilde{\mathrm{T}}' \tilde{\mathrm{T}}^c$ calculée dans la vérification de la preuve de connaissance nous permet de dire que $g_1^{s''_\alpha} = \tilde{\mathrm{T}}^{c''}$. Les exposants sont dans un groupe d'ordre premier connu, on peut alors poser $\tilde{\alpha} = s''_\alpha / c''$. On a alors $g_1^{\tilde{\alpha}} = \tilde{\mathrm{T}}$. En reprenant l'équation $\tilde{\mathrm{T}}^{s_x} g_1^{-s_z} = \bar{\mathrm{T}}$ on obtient, en divisant les deux instances : $\tilde{\mathrm{T}}^{s''_x} = g_1^{s''_z}$. Comme $\tilde{\mathrm{T}} = g_1^{\tilde{\alpha}}$, on peut écrire que $g_1^{\tilde{\alpha} s''_x} = g_1^{s''_z}$, autrement dit $\tilde{\alpha} s''_x = s''_z \mod p$.
On définit maintenant

$$\tilde{s} = \frac{s''_s}{c''}, \qquad \tilde{x} = \frac{s''_x}{c''}, \qquad \tilde{x}_u = \frac{s''_{x_u}}{c''}, \qquad \tilde{r} = \frac{s''_r}{c''}.$$

On obtient

$$e(\mathrm{T}, g_2)^{\tilde{x}} \cdot e(h_2, \Gamma)^{-\tilde{\alpha}} \cdot e(h_1, g_2)^{-\tilde{s}} \cdot e(h_2, g_2)^{-(\tilde{x}_u + \tilde{\alpha}\tilde{x})} \cdot e(h_4, g_2)^{-\tilde{r}} = \frac{e(g_1, g_2) \cdot e(h_3, g_2)^m}{e(\mathrm{T}, \Gamma)}.$$

Par conséquent, en posant $\tilde{\mathrm{A}} = \mathrm{T} h_2^{-\tilde{\alpha}}$ on obtient l'égalité suivante :

$$e(\tilde{\mathrm{A}}, g_2)^{\tilde{x}} \cdot e(\tilde{\mathrm{A}}, \Gamma) \cdot e(h_1, g_2)^{-\tilde{s}} \cdot e(h_2, g_2)^{-\tilde{x}_u} \cdot e(h_3, g_2)^{-m} \cdot e(h_4, g_2)^{-\tilde{r}} = e(g_1, g_2),$$

ce qui signifie que $(\tilde{\mathrm{A}}, \tilde{s}, \tilde{x}, \tilde{x}_u, \tilde{r})$ sont valides.

□

Divulgation nulle de connaissance : il suffit de montrer que les transcripts du protocole peuvent être simulés de manière indistinguable, sous l'hypothèse DDH, sans la connaissance des secrets. Le simulateur choisit $\mathrm{A} \in_{\mathrm{R}} \mathbb{G}_1$ et $\alpha \in_{\mathrm{R}} \mathbb{Z}_p$ et pose $\mathrm{T} = \mathrm{A}h_2^{\alpha}$ et $\tilde{\mathrm{T}} = g_1^{\alpha}$. Sous l'hypothèse DDH, le couple $(\mathrm{T}, \tilde{\mathrm{T}})$ généré par le simulateur est indistinguable de la sortie de n'importe quel prouveur honnête. Le simulateur choisit ensuite des valeurs x, x_u, s, r aléatoirement et calcule les valeurs $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, \mathrm{T}_5$ comme décrit dans la preuve. Cet ensemble de valeurs est indistinguable de la sortie de n'importe quel prouveur honnête. La simulation se fait ensuite en utilisant les mêmes techniques que pour une preuve de Schnorr : le simulateur choisit la valeur c aléatoirement dans $[0, p-1]$ et les $s_i, i \in \{\alpha, s, x, x_u, r, z, a, b\}$ aléatoirement dans $\mathbb{Z}_p$. Il calcule ensuite les valeurs $\mathrm{T}', \tilde{\mathrm{T}}', \bar{\mathrm{T}}, \mathrm{T}'_1, \mathrm{T}'_2, \mathrm{T}'_3, \mathrm{T}'_4, \mathrm{T}'_5$ en utilisant les équations de vérification.

□

9.2.4 Indistinguabilité

Nous montrons l'indistinguabilité de notre schéma à l'aide d'une preuve par jeux (*cf.* section 2.2.3). Les interactions entre l'adversaire $\mathcal{A}$ et les joueurs sont simulées par un distingueur $\mathcal{D}$ qui cherche à casser l'indistinguabilité du chiffrement double ElGamal ou du chiffrement de Paillier.

Nous nous plaçons dans le modèle de l'oracle aléatoire. $\mathcal{A}$ et $\mathcal{D}$ ont donc accès à un même oracle de hachage Hash qui répond de manière classique aux requêtes de hachage.

Nous commençons par décrire le Jeu 0 qui correspond au jeu joué par l'adversaire dans son attaque normale.

Jeu 0 : L'adversaire peut tout d'abord faire appel aux oracles AddU et CrptU afin d'ajouter de nouveaux utilisateurs au système.

Dans cette attaque nous nous intéressons plus particulièrement à la requête Choose_b faite par $\mathcal{A}$. Comme nous l'avons vu dans la description de la propriété d'indistinguabilité (*cf.* section 8.4), il est possible de se restreindre au cas où $\mathcal{A}$ fait une seule requête à l'oracle Choose_b. $\mathcal{A}$ spécifie deux utilisateurs $\mathcal{U}_1$ et $\mathcal{U}_2$ et deux messages m_0 et m_1. Il reçoit en réponse les deux signatures produites dans un ordre aléatoire et doit déterminer lors de quelles interactions elles ont été produites.

Les vues produites lors des deux interactions sont de la forme suivante :

$$\begin{aligned}
\mathcal{V}_{\mathcal{A}}^{b} &= (\mathrm{C}_b = h_1^{s'_b} h_3^{m_b} h_4^{r_b}, \mathrm{Id}_{\mathcal{U}_0}, \mathrm{M}_1^b = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(m_b), \mathrm{M}_2^b = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(s'_b), \\
&\quad \mathrm{M}_3^b = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(r_b), \Delta_1^b = \mathsf{Encrypt}_{sk_{\mathcal{RA}}}^{\mathsf{2EG}}(h_1^{s'_b}), pok_1^b), \\
\mathcal{V}_{\mathcal{A}}^{\bar{b}} &= (\mathrm{C}_{\bar{b}} = h_{\bar{b}}^{s'_{\bar{b}}} h_3^{m_{\bar{b}}} h_4^{r_{\bar{b}}}, \mathrm{Id}_{\mathcal{U}_1}, \mathrm{M}_1^{\bar{b}} = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(m_{\bar{b}}), \mathrm{M}_2^{\bar{b}} = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(s'_{\bar{b}}), \\
&\quad \mathrm{M}_3^{\bar{b}} = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(r_{\bar{b}}), \Delta_1^{\bar{b}} = \mathsf{Encrypt}_{sk_{\mathcal{RA}}}^{\mathsf{2EG}}(h_1^{s'_{\bar{b}}}), pok_1^{\bar{b}}).
\end{aligned}$$

Les deux signatures obtenues sont, quant à elles, de la forme :

$$\begin{aligned}\sigma_b &= m_b, h_1^{s_b}, \Delta_1^b = \mathsf{Encrypt}^{\mathsf{2EG}}_{sk_{\mathcal{RA}}}(\mathrm{Id}_{\mathcal{U}_0}), pok_2^b,\\ \sigma_{\bar{b}} &= m_{\bar{b}}, h_1^{s_{\bar{b}}}, \Delta_1^{\bar{b}} = \mathsf{Encrypt}^{\mathsf{2EG}}_{sk_{\mathcal{RA}}}(\mathrm{Id}_{\mathcal{U}_1}), pok_2^{\bar{b}}.\end{aligned}$$

$\mathcal{A}$ reçoit les vues et les signatures ordonnées selon le bit b de l'oracle Choose. À la fin de son attaque, $\mathcal{A}$ renvoie un bit $\hat{b}$. On note alors S_0 l'événement $\hat{b} = b$. On a alors

$$\mathbf{Adv}^{\mathrm{blind}}_{\mathrm{FBS},\mathcal{A}}(k) = 2|\Pr[\mathrm{S}_0] - 1/2|.$$

Jeu 1 : $\mathcal{D}$ construit les paramètres du système en choisissant toutes les clés, à savoir celle pour le signataire $sk_{\mathcal{S}}$ et celle pour l'autorité de révocation $sk_{\mathcal{RA}}$. Il envoie ensuite les paramètres du système construit selon la description du protocole et la clé $sk_{\mathcal{S}}$ à $\mathcal{A}$. Le Jeu 1 se déroule comme le Jeu 0 : $\mathcal{D}$ répond aux requêtes aux oracles et garde tous les secrets associés aux utilisateurs. En particulier, il choisit la valeur du bit b de la requête Choose_b.

AddU : lorsque $\mathcal{A}$ demande à ajouter un utilisateur au système, $\mathcal{D}$ calcule les clés pour cet utilisateur et renvoie la clé publique à $\mathcal{A}$. $\mathcal{D}$ enregistre alors la clé secrète de l'utilisateur dans la liste HU.

CrptU : lorsque $\mathcal{A}$ souhaite ajouter un utilisateur corrompu au système, $\mathcal{D}$ inscrit cet utilisateur dans la liste CU.

USK : lorsque $\mathcal{A}$ souhaite connaître la clé secrète d'un utilisateur honnête, $\mathcal{D}$ est capable de la lui renvoyer grâce à la liste HU.

User : lorsque $\mathcal{A}$ souhaite faire signer un utilisateur (honnête) $\mathcal{D}$ est capable de répondre à la requête en utilisant la clé secrète de cet utilisateur.

Tsig : lorsque $\mathcal{A}$ demande à tracer une signature, $\mathcal{D}$ est capable de répondre à la requête en utilisant la clé secrète de l'autorité de révocation.

Tid : lorsque $\mathcal{A}$ demande à tracer une identité, $\mathcal{D}$ est capable de répondre à la requête en utilisant la clé secrète de l'autorité de révocation.

La seule différence est que cette fois $\mathcal{D}$ simule la preuve de connaissance pok_2^b, notée $\tilde{pok}_2^b$, dans la première signature :
- il choisit $\alpha \in_{\mathrm{R}} \mathbb{Z}_p$,
- il pose $\mathrm{T} \leftarrow \mathrm{A}_b h_2^{\alpha}$ et $\tilde{\mathrm{T}} = g_1^{\alpha}$,
- il calcule $(\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4) \leftarrow \mathsf{Encrypt}^{\mathsf{2EG}}_{sk_{\mathcal{RA}}}(\mathrm{Id}_{\mathcal{U}_0})$,
- il choisit $s_a, s_b, s_{x_b}, s_{x_{u_b}}, s_r, s_z, s_\alpha, s_b \in_{\mathrm{R}} \mathbb{Z}_p^8$,
- il choisit $c \in_{\mathrm{R}} \{0,1\}^k$,
- il calcule alors les valeurs $\mathrm{T}', \tilde{\mathrm{T}}', \bar{\mathrm{T}}, \mathrm{T}'_1, \mathrm{T}'_2, \mathrm{T}'_4, \mathrm{T}'_5$ à l'aide des équations de véri-

fication de la preuve :

$$\begin{aligned}
\mathrm{T}'_1 &= g_1^{s_a}\mathrm{T}_1^{-c}, \quad \mathrm{T}'_2 = g_1^{s_{x_{u_b}}} u^{s_a}\mathrm{T}_2^{-c}, \quad \mathrm{T}'_3 = g_1^{s_b}\mathrm{T}_1^{-c},\\
\mathrm{T}'_4 &= g_1^{s_{x_{u_b}}} v^{s_b}\mathrm{T}_4^{-c}, \quad \mathrm{T}'_5 = h_1^{s_s}(h_1^s)^c,\\
\mathrm{T}' &= e(\mathrm{T}, g_2)^{s_x} \cdot e(h_2, \Gamma)^{-s_\alpha} \cdot e(h_1, g_2)^{-s_s} \cdot e(h_2, g_2)^{-s_{x_u}-s_z} \cdot e(g_1, g_2)^{-s_r},\\
\tilde{\mathrm{T}}' &= g_1^{s_a}\tilde{\mathrm{T}}^{-c}, \quad \bar{\mathrm{T}} = \tilde{\mathrm{T}}^{s_x} g_1^{-s_z},\\
&\quad \Big(\frac{e(g_1, g_2) \cdot e(h_3, g_2)^m}{e(\mathrm{T}, \Gamma)}\Big)^{-c} c.
\end{aligned}$$

- Il pose alors

$$\mathsf{Hash}(m, \mathrm{T}, \tilde{\mathrm{T}}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, \mathrm{T}_5, \mathrm{T}', \tilde{\mathrm{T}}', \bar{\mathrm{T}}, \mathrm{T}'_1, \mathrm{T}'_2, \mathrm{T}'_3, \mathrm{T}'_4, \mathrm{T}'_5) \leftarrow .$$

La simulation produite par $\mathcal{D}$ est calculatoirement indistinguable pour l'adversaire s'il ne constate aucune collision aux requêtes de hachage Si on note F l'événement « une collision dans les requêtes de hachage est arrivée », on a alors, par le lemme 2 (*cf.* section 2.2.3)

$$|\Pr[\mathrm{S}_0] - \Pr[\mathrm{S}_1]| \leq \Pr[\mathrm{F}].$$

Si on note q_{H} le nombre de requêtes faites par $\mathcal{A}$ à l'oracle de hachage et q_{S} le nombre de requêtes à l'oracle User on obtient alors

$$\Pr[\mathrm{F}] \leq \frac{q_{\mathsf{H}} + q_{\mathsf{S}}}{p^{14}}.$$

D'où

$$|\Pr[\mathrm{S}_1] - \Pr[\mathrm{S}_0]| \leq \frac{q_{\mathsf{H}} + q_{\mathsf{S}}}{p^{14}}.$$

Jeu 2 Le Jeu 2 est identique au Jeu 1, à la différence qu'ici $\mathcal{D}$ envoie un chiffrement ElGamal d'une valeur aléatoire $\tilde{\mathrm{A}}_b$. Il pose $\mathrm{T} \leftarrow \tilde{\mathrm{A}}_b h_2^\alpha$ et $\tilde{\mathrm{T}} = g_1^\alpha$ dans la seconde preuve de connaissance. La simulation produite par $\mathcal{D}$ est calculatoirement indistinguable pour l'adversaire s'il n'est pas capable de distinguer la simulation de la preuve d'une preuve réelle. $\mathcal{D}$ peut donc utiliser cet adversaire comme distingueur contre le chiffrement ElGamal. Cet avantage étant limité par l'avantage contre le problème DDH on obtient

$$|\Pr[\mathrm{S}_2] - \Pr[\mathrm{S}_1]| \leq \mathbf{Adv}_{\mathcal{D}}^{\mathrm{DDH}}(k).$$

Jeu 3 : Le Jeu 3 est identique au Jeu 2, à la différence qu'ici $\mathcal{D}$ simule aussi la preuve $pok_2^{\bar{b}}$ et renvoie un chiffré d'une valeur aléatoire $\tilde{\mathrm{A}}_{\bar{b}}$. On obtient de manière identique

$$|\Pr[\mathrm{S}_3] - \Pr[\mathrm{S}_2]| \leq \frac{q_{\mathsf{H}} + q_{\mathsf{S}}}{p^{14}} + \mathbf{Adv}_{\mathcal{D}}^{\mathrm{DDH}}(k).$$

Jeu 4 : Dans le Jeu 4, $\mathcal{D}$ va maintenant simuler la preuve de connaissance de la première vue.

En reprenant les mêmes arguments qu'au Jeu 1, on peut borner la différence statistique[1] entre les Jeux 3 et 4 par la probabilité de collisions aux requêtes de hachage. Comme précédemment cette probabilité est négligeable, d'où

$$|\Pr[\mathrm{S}_4] - \Pr[\mathrm{S}_3]| \leq \nu(k),$$

où $\nu(k)$ est une fonction négligeable.

Jeu 5 : Le Jeu 5 est identique au Jeu 4, à la différence que cette fois, $\mathcal{D}$ simule aussi la preuve $pok_1^{\bar{b}}$. On obtient de manière identique

$$|\Pr[\mathrm{S}_5] - \Pr[\mathrm{S}_4]| \leq \nu(k).$$

Jeu 6 : $\mathcal{D}$ va maintenant commencer à modifier les éléments composant la signature σ_b et utiliser $\mathcal{A}$ afin de casser l'indistinguabilité du double chiffrement ElGamal. Au lieu de choisir lui-même les clés pour l'autorité de révocation, il demande à son challengeur $\mathcal{C}$ (pour l'indistinguabilité du double ElGamal) de lui fournir une clé publique $pk_{\mathcal{RA}}$. Il peut toujours répondre aux requêtes User de $\mathcal{A}$ et lorsque $\mathcal{A}$ fait une requête $\mathsf{T_{sig}}$ ou $\mathsf{T_{id}}$, il interroge son propre oracle de déchiffrement $\mathsf{Decrypt}^{\mathsf{2EG}}_{sk_{\mathcal{RA}}}$ pour répondre.
Lorsque $\mathcal{A}$ fait sa requête Choose_b, au lieu de chiffrer l'identité $\mathrm{Id}_{\mathcal{U}_0}$, il choisit une valeur aléatoire $h_2^{\tilde{x}_b} \in_{\mathrm{R}} \mathbb{G}_1$.
Nous décrivons ici le jeu que joue $\mathcal{D}$ vis-à-vis de son challengeur et expliquons ensuite comment il utilise $\mathcal{A}$ pour réussir son attaque.

Jeu d **:** $\mathcal{D}$ fournit à son challengeur $m_0 = \mathrm{Id}_{\mathcal{U}_0}$ et $m_1 = h_2^{\tilde{x}_b}$. $\mathcal{C}$ renvoie à $\mathcal{D}$ le chiffrement $\tilde{\Delta}_2^d = \mathsf{Encrypt}^{\mathsf{2EG}}_{sk_{\mathcal{RA}}}(m_d)$ où $d \in \{0,1\}$ a été choisi par $\mathcal{C}$ et est tenu secret de $\mathcal{D}$. $\mathcal{D}$ doit donc deviner d.

Pour ce faire, au lieu de renvoyer le chiffré de l'identité $\mathrm{Id}_{\mathcal{U}_b}$, $\mathcal{D}$ renvoie $\tilde{\Delta}_2^d = \mathsf{Encrypt}^{\mathsf{2EG}}_{sk_{\mathcal{RA}}}(m_d)$ et une preuve simulée $\tilde{pok}_2^b$ de sa connaissance du chiffré dans $\tilde{\Delta}_2^d$. Celle-ci peut être faite de manière classique dans le modèle de l'oracle aléatoire. À la fin de son attaque, $\mathcal{A}$ retourne un bit $\hat{b}$. Si $\hat{b} = b$, alors $\mathcal{D}$ retourne 1 à $\mathcal{C}$ sinon, il retourne 0. On obtient alors

$$\Pr[\hat{d} = 1 | d = 0] = \Pr[\mathrm{S}_5] \quad \text{et} \quad \Pr[\hat{d} = 1 | d = 1] = \Pr[\mathrm{S}_6],$$

où $\hat{d}$ est le bit renvoyé par $\mathcal{D}$ à $\mathcal{C}$. Par conséquent

$$|\Pr[\mathrm{S}_6] - \Pr[\mathrm{S}_5]| = \mathbf{Adv}^{\mathrm{ind-cca2}}_{\mathrm{2EG},\mathcal{D}}(k).$$

Jeu 7 Le Jeu 7 est identique au Jeu 6 à la différence que cette fois $\mathcal{D}$ modifie également le double chiffrement ElGamal de la signature $\sigma_{\bar{b}}$. On obtient comme précédemment

$$|\Pr[\mathrm{S}_7] - \Pr[\mathrm{S}_6]| = \mathbf{Adv}^{\mathrm{ind-cca2}}_{\mathrm{2EG},\mathcal{D}}(k).$$

[1] La simulation nécessitant des simulations de preuves de connaissance de chiffrement de Paillier, elle est statistiquement indistinguable.

Jeu 8 : $\mathcal{D}$ change les vues qu'il renvoie à l'adversaire lors de sa requête Choose_b. Au lieu de calculer correctement la valeur M_1^b, il choisit une valeur $\tilde{m}_b$ et renvoie $\tilde{\mathrm{M}}_1^b = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Pai}}(\tilde{m}_b)$. Il construit ensuite une preuve simulée $\tilde{pok}_1^b$ de la même manière qu'il l'a simulée dans les jeux précédents. On obtient alors

$$|\Pr[\mathrm{S}_8] - \Pr[\mathrm{S}_7]| = \mathbf{Adv}^{\mathrm{ind}}_{\mathsf{Pai},\mathcal{D}}(k).$$

Jeu 9 : $\mathcal{D}$ change la deuxième vue de manière similaire au Jeu 8 en chiffrant une valeur $\tilde{m}_{\bar{b}}$. On obtient comme précédemment

$$|\Pr[\mathrm{S}_9] - \Pr[\mathrm{S}_8]| = \mathbf{Adv}^{\mathrm{ind}}_{\mathsf{Pai},\mathcal{D}}(k).$$

Jeu 10-13 : Par des jeux similaires, $\mathcal{D}$ remplace les valeurs s'_b, $s'_{\bar{b}}$, r_b et $r_{\bar{b}}$ par des valeurs aléatoires de $\mathbb{Z}_p$ dans les chiffrements de Paillier des deux vues. On obtient

$$|\Pr[\mathrm{S}_{13}] - \Pr[\mathrm{S}_9]| = 4\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{Pai},\mathcal{D}}(k).$$

Jeu 14 : Comme il l'a fait au Jeu 6, $\mathcal{D}$ remplace le double chiffrement Δ_1^b de la valeur $h_1^{s'_b}$ par le double chiffrement $\tilde{\Delta}_1^b$ de $h_1^{\tau_b}$ où τ_b est choisi aléatoirement dans $\mathbb{Z}_p$. Comme au Jeu 6 la différence de probabilité est majorée par l'avantage de $\mathcal{D}$ contre l'indistinguabilité du double chiffrement ElGamal. On obtient

$$|\Pr[\mathrm{S}_{14}] - \Pr[\mathrm{S}_{13}]| = \mathbf{Adv}^{\mathrm{ind-cca2}}_{\mathrm{2EG},\mathcal{D}}(k).$$

Jeu 15 Le Jeu 15 est identique au Jeu 14 à la différence que cette fois $\mathcal{D}$ modifie également le double chiffrement ElGamal de la deuxième vue. Les vues sont alors de la forme suivante

$$\begin{aligned}
\mathcal{V}_{\mathcal{A}}^b &= (\mathrm{C}_b = h_1^{s'_b} h_3^{m_b} h_4^{r_b}, \mathrm{Id}_{\mathcal{U}_b}, \tilde{\mathrm{M}}_1^b = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(\tilde{m}_b), \tilde{\mathrm{M}}_2^b = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Pai}}(\tilde{s}'_b),\\
&\quad \tilde{\mathrm{M}}_3^b = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(\tilde{r}_b), \tilde{\Delta}_1^b = \mathsf{Encrypt}_{sk_{\mathcal{RA}}}^{\mathsf{2EG}}(h_1^{\tau_b}), \tilde{pok}_1^b),\\
\mathcal{V}_{\mathcal{A}}^{\bar{b}} &= (\mathrm{C}_{\bar{b}} = h_1^{s'_{\bar{b}}} h_3^{m_{\bar{b}}} h_4^{r_{\bar{b}}}, \mathrm{Id}_{\mathcal{U}_{\bar{b}}}, \tilde{\mathrm{M}}_1^{\bar{b}} = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(\tilde{m}_{\bar{b}}), \tilde{\mathrm{M}}_2^{\bar{b}} = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(\tilde{s'}_{\bar{b}}),\\
&\quad \tilde{\mathrm{M}}_3^{\bar{b}} = \mathsf{Encrypt}_{\mathsf{n,g}}^{\mathsf{Paillier}}(\tilde{r}_{\bar{b}}), \tilde{\Delta}_1^{\bar{b}} = \mathsf{Encrypt}_{sk_{\mathcal{RA}}}^{\mathsf{2EG}}(h_1^{\tau_{\bar{b}}}), \tilde{pok}_1^{\bar{b}}).
\end{aligned}$$

Les deux signatures obtenues sont, quant à elles, de la forme

$$\begin{aligned}
\sigma_b &= m_b, h_1^{s_b}, \Delta_2^b = \mathsf{Encrypt}_{sk_{\mathcal{RA}}}^{\mathsf{2EG}}(h_2^{\tilde{x}_b}), \tilde{pok}_2^b,\\
\sigma_{\bar{b}} &= m_{\bar{b}}, h_1^{s_{\bar{b}}}, \Delta_2^{\bar{b}} = \mathsf{Encrypt}_{sk_{\mathcal{RA}}}^{\mathsf{2EG}}(h_2^{\tilde{x}_{\bar{b}}}), \tilde{pok}_2^{\bar{b}}.
\end{aligned}$$

Comme précédemment on obtient

$$|\Pr[\mathrm{S}_{15}] - \Pr[\mathrm{S}_{14}]| = \mathbf{Adv}^{\mathrm{ind-cca2}}_{\mathrm{2EG},\mathcal{D}}(k).$$

Dans ce dernier jeu, aucune information sur le bit b ne transparaît. On a alors

$$Pr[\mathrm{S}_{15}] = 1/2.$$

Il est donc possible maintenant d'évaluer l'avantage de l'adversaire $\mathcal{A}$ contre l'indistinguabilité du schéma FBS. Cet avantage est donné par

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{blind}}_{\mathrm{FBS}}(\mathcal{A}) &= 2|\Pr[\mathrm{S}_0] - 1/2| \\
&= 2|\Pr[\mathrm{S}_0] - \Pr[\mathrm{S}_{15}]| \\
&\leq 2|\Pr[\mathrm{S}_0] - \Pr[\mathrm{S}_1]| + \cdots + |\Pr[\mathrm{S}_{15}] - \Pr[\mathrm{S}_{14}]| \\
&\leq 8 \times \mathsf{Adv}^{\mathrm{ind-cca2}}_{\mathrm{2EG},\mathcal{D}}(k) + 12 \times \mathsf{Adv}^{\mathrm{ind}}_{\mathrm{Pai},\mathcal{D}}(k) + 4 \times \frac{q_{\mathsf{H}} + q_{\mathsf{S}}}{p^8} + \nu(k).
\end{aligned}$$

9.2.5 Traçabilité

Nous montrons dans cette section que notre schéma est traçable pour l'identité ou la signature.

Théorème 21 (Traçabilité)**.** *Le schéma de signature aveugle à anonymat révocable proposé est traçable pour la signature et pour l'identité, dans le modèle de l'oracle aléatoire, sous l'hypothèse q-SDH.*

Les deux preuves utilisent le lemme de bifurcation. Nous ramenons la sécurité de notre schéma à l'inforgeabilité du schéma BBS^{ext}_{l} que nous avons montrée dans la section 9.2.2. L'inforgeabilité de ce schéma reposant sur l'hypothèse q-SDH, il en est de même de notre schéma.

9.2.5.1 Traçage d'identité

Afin de casser le traçage d'identité de notre schéma, l'adversaire $\mathcal{A}$ doit être capable de produire une signature

$$\sigma = (h_1^s, \Delta_2, \mathrm{T}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, c, s_\alpha, s_x, s_s, s_{x_u}, s_r, s_z, s_a, s_b)$$

valide, sur un message m de son choix telle que $\mathsf{R}_{\mathsf{id}}(m, \sigma, sk_{\mathcal{RA}}) = (\mathrm{Id}_{\mathcal{U}}, \pi)$ et que
- cas 1 : $\mathrm{Id}_{\mathcal{U}} = \perp$,
- cas 2 : $\mathsf{Match}_{\mathsf{id}}(\mathrm{Id}_{\mathcal{U}}, m, \sigma, \pi) = 0$.

Nous détaillons pourquoi aucun de ces deux cas ne peut arriver.

Cas 1 Dans ce cas, l'adversaire a réussi à produire une signature valide telle que l'autorité de révocation renvoie une identité non conforme. Rappelons que l'identité est chiffrée à l'aide d'un double chiffrement ElGamal $\Delta_2 = (\mathrm{M}_1 = g_1^a, \mathrm{M}_2 = g_1^{x_u} u^a, \mathrm{M}_3 = g_1^b, \mathrm{M}_4 = g_1^{\tilde{x_u}} v^b, pok(\alpha : \mathrm{M}_2 = g_1^\alpha u^a \wedge \mathrm{M}_4 = g_1^\alpha v^b))$. La signature produite par $\mathcal{A}$ étant valide, cela signifie que $\mathcal{A}$ a réussi à produire un chiffrement Δ_2 tel que $\mathsf{Decrypt}^{\mathrm{2EG}}_{pk_{\mathcal{RA}}}(\Delta_2)$ renvoie deux valeurs $h_2^{x_u}$ et $h_2^{\tilde{x_u}}$ et une fausse preuve d'égalité des textes clairs correspondants. Fouque et Pointcheval ont montré dans [FP01] que la preuve de connaissance du double chiffrement ElGamal était significative. Par conséquent, la probabilité que $\mathcal{A}$ réussisse à produire un tel chiffrement possédant une fausse preuve d'égalité est négligeable.

Cas 2 Dans ce cas, le déchiffrement de Δ_2 est correct mais le juge n'est pas capable de retrouver l'identité produite dans la liste $\mathsf{Tab_{id}}$ construite par le signataire lors de l'enregistrement des utilisateurs. Nous montrons comment il est possible alors d'utiliser cet adversaire $\mathcal{A}$ pour construire un adversaire $\mathcal{B}$ utilisant $\mathcal{A}$ comme sous-programme contre l'inforgeabilité du schéma BBS_l^{ext} (dans le cas particulier où $l = 5$).

$\mathcal{B}$ a accès à un oracle de signature pour la signature $\mathsf{Sign}_{\mathrm{BBS}_{\mathrm{L}}^{ext}}$ qui prend en entrée un bloc de message et renvoie une signature BBS_l^{ext} valide. $\mathcal{B}$ va simuler toutes les réponses aux requêtes de $\mathcal{A}$ aux différents oracles.

Tout d'abord, $\mathcal{B}$ reçoit de son challenger les paramètres publics pour un schéma de signature BBS_l^{ext}, à savoir $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, e, \Gamma = g_2^{\gamma}, h_1, \ldots, h_4)$. Il choisit ensuite les clés pour le chiffrement de Paillier telles que nous les avons décrites et envoie à $\mathcal{A}$ les paramètres publics $params = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2, , h_1, h_2, h_3, h_4, \Gamma = g_2^{\gamma}, \mathsf{g}, \mathsf{n})$. Il répond aux requêtes de $\mathcal{A}$ de la manière suivante :

$\mathsf{AddU}(\mathcal{U})$: lorsque $\mathcal{A}$ souhaite inscrire un nouvel utilisateur honnête, $\mathcal{B}$ vérifie que cet utilisateur n'a pas déjà été corrompu, sinon il choisit $x_u \in_{\mathrm{R}} \mathbb{Z}_p^*$ et calcule $pk_{\mathcal{U}} = h_2^{x_u}$. Il renvoie $pk_{\mathcal{U}}$ à $\mathcal{A}$ et ajoute $pk_{\mathcal{U}}, sk_{\mathcal{U}}$ à la liste HU des utilisateurs honnêtes.

$\mathsf{CrptU}(\mathrm{Id}_{\mathcal{U}}, pk_{\mathcal{U}})$: lorsque $\mathcal{A}$ souhaite ajouter un utilisateur corrompu il envoie à son oracle l'identité et la clé publique de cet utilisateur. $\mathcal{B}$ ajoute $pk_{\mathcal{U}}$ à la liste CU des utilisateurs corrompus.

$\mathsf{USK}(\mathrm{Id}_{\mathcal{U}}, pk_{\mathcal{U}})$: lorsque $\mathcal{A}$ souhaite recevoir la clé privée d'un utilisateur, $\mathcal{B}$ vérifie que cet utilisateur appartient bien à la liste HU. Dans ce cas, il renvoie la clé $sk_{\mathcal{U}}$ correspondant à $pk_{\mathcal{U}}$ et ajoute $\mathrm{Id}_{\mathcal{U}}$ à CU.

$\mathsf{Sign}(\mathrm{Id}_{\mathcal{U}}, m)$: lorsque l'utilisateur souhaite obtenir une signature pour un message m, $\mathcal{B}$ interagit avec $\mathcal{A}$ comme suit :

1. Tout d'abord, $\mathcal{A}$ envoie le premier échange du protocole de signature : $(\mathrm{C} = h_1^{s'} h_3^m, h_4^r, \mathrm{M}_1 = \mathsf{g}^m t_1^{\mathsf{n}} \mod \mathsf{n}^2, \mathrm{M}_2 = \mathsf{g}^{s'} t_2^{\mathsf{n}} \mod \mathsf{n}^2, \mathrm{M}_3 = \mathsf{g}^r t_3^{\mathsf{n}} \mod \mathsf{n}^2, \Delta_1', pok_1)$.
2. $\mathcal{B}$ vérifie la preuve envoyée par $\mathcal{A}$ — rappelons que $\mathcal{B}$ joue ici le rôle d'un signataire honnête — et connaît les clés de chiffrement adéquates pour extraire les valeurs r, s', m des chiffrements de Paillier $\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3$. Si la preuve est valide, $\mathcal{B}$ choisit une valeur $s'' \in_{\mathrm{R}} \mathbb{Z}_p^*$ et calcule $s = s' + s'' \mod p$. Il extrait de même, la clé secrète associée à l'identité $\mathrm{Id}_{\mathcal{U}}$, grâce à la phase d'authentification.
3. Il envoie alors à son oracle de signature les valeurs m, s, x_u, r. Il reçoit en réponse un couple (A, x) qui est une signature BBS_l^{ext} valide sur le bloc de message (m, s, x_u, r).
4. Il renvoie (A, x) à $\mathcal{A}$. La simulation de $\mathcal{B}$ pour la signature est alors parfaite pour l'adversaire $\mathcal{A}$.

À la fin de son attaque, $\mathcal{A}$ renvoie un couple message/signature (m, σ) tel que

$$\sigma = (h_1^s, \Delta_2, \mathrm{T}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, c, s_\alpha, s_x, s_z, s_s, s_{x_u}, s_r, s_a, s_b)$$

et $\mathsf{R_{id}}(m, \sigma, sk_{\mathcal{RA}}) = (\mathrm{Id}_{\mathcal{U}}, \pi)$ et $\mathsf{Match_{id}}(\mathrm{Id}_{\mathcal{U}}, m, \sigma, \pi) = 0$ avec une probabilité non négligeable ε.

La signature σ est de la forme

$$(m, h_1^s, \Delta_2, \mathrm{T}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, c, s_\alpha, s_x, s_z, s_s, s_{x_u}, s_r, s_a, s_b)$$

et peut être transformée sans ajouter de nouvelles requêtes à l'oracle aléatoire sous la forme plus classique

$$(m, h_1^s, \Delta_2, \mathrm{T}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, \mathrm{T}', \mathrm{T}'_1, \mathrm{T}'_2, \mathrm{T}'_3, \mathrm{T}'_4, \mathrm{T}'_5, c, s_\alpha, s_x, s_z, s_s, s_{x_u}, s_r, s_a, s_b)$$

avec $c = \mathrm{H}(m, h_1^s, \mathrm{T}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, \mathrm{T}', \mathrm{T}'_1, \mathrm{T}'_2, \mathrm{T}'_3, \mathrm{T}'_4, \mathrm{T}'_5)$.

Il nous est alors possible d'utiliser le lemme de bifurcation (*cf.* lemme 3, section 2.2.3) avec notre adversaire $\mathcal{A}$. On obtient alors deux signatures valides $(m, \sigma_1, c, \sigma_2)$ et $(m, \sigma'_1, c', \sigma'_2)$ où $c \neq c'$. En utilisant la même méthode que pour montrer la preuve de significativité de la preuve pok_2, $\mathcal{B}$ est capable d'extraire les secrets $(\tilde{\mathrm{A}}, \tilde{x}, \tilde{s}, \tilde{x_u}, \tilde{r})$ contenus dans la signature avec une probabilité non négligeable. Le couple $(\tilde{\mathrm{A}}, \tilde{x})$ qu'il a ainsi obtenu est une signature BBS_l^{ext} valide sur le bloc de message $(m, \tilde{s}, \tilde{x_u}, \tilde{r})$. Par définition de l'attaque de $\mathcal{A}$, $\tilde{x_u}$ n'est la clé secrète d'aucun utilisateur. Par conséquent, $\mathcal{B}$ n'a jamais fait de requête sur le bloc de message $(m, \tilde{s}, \tilde{x_u}, \tilde{r})$ à son oracle de signature. $\mathcal{B}$ a donc réussi à casser l'inforgeabilité du schéma BBS_l^{ext} avec une probabilité non négligeable.

9.2.5.2 Traçage de signature

Afin de casser le traçage de signature de notre schéma, l'adversaire $\mathcal{A}$ a deux possibilités :

- cas 1 : il renvoie un couple message/signature (m, σ) valide tel que pour toutes les valeurs $(\mathrm{I}_{sig_i}, \pi_{1,i})$ renvoyées par l'autorité RA et calculées à partir des transcriptions $trans_i \in \mathrm{Trans}$, l'algorithme $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig_i}, m, \sigma, \pi_{1,i})$ renvoie toujours 0 et donc le juge n'accepte jamais la signature,
- cas 2 : il renvoie deux couples message/signature (m_1, σ_1) et (m_2, σ_2) et on dit que $\mathcal{A}$ réussit son attaque si l'autorité RA trouve un transcript $trans \in \mathrm{Trans}$ tel que $\mathsf{R}_{\mathsf{sig}}(trans, sk_{\mathrm{RA}}) = (\mathrm{I}_{sig}, \pi_1)$ et l'algorithme $\mathsf{Match}_{\mathsf{sig}}$ renvoie 1 pour les deux signatures, c'est-à-dire $\mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, m_1, \sigma_1, \pi_1) = \mathsf{Match}_{\mathsf{sig}}(\mathrm{I}_{sig}, m_2, \sigma_2, \pi_1) = 1$. Autrement dit, le juge accepte deux signatures pour le même transcript.

Comme précédemment, nous utilisons un adversaire contre la traçabilité du schéma afin de construire un algorithme $\mathcal{B}$ contre l'inforgeabilité du schéma BBS_l^{ext}.

Avant de commencer son attaque, $\mathcal{B}$ fait une hypothèse sur la sortie de $\mathcal{A}$. Deux cas sont alors possibles.

Cas 1 $\mathcal{B}$ suppose que $\mathcal{A}$ renvoie un couple message/signature qui lui permet de casser le traçage de signature avec une probabilité non négligeable. $\mathcal{B}$ simule alors toutes les requêtes que $\mathcal{A}$ fait. Il reçoit de son challenger les paramètres pour une signature BBS_l^{ext} et construit les paramètres publics pour $\mathcal{A}$ comme décrit dans le protocole. Il répond aux requêtes de $\mathcal{A}$ comme nous l'avons décrit précédemment.

À la fin de son attaque, $\mathcal{A}$ renvoie un couple message/signature (m, σ) tel que $\sigma = (h_1^s, \Delta_2, \mathrm{T}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4, c, s_\alpha, s_x, s_s, s_{x_u}, s_r, s_z, s_a, s_b)$ qui casse la traçabilité de signature avec probabilité non négligeable. Par définition du succès de $\mathcal{A}$, la valeur h_1^s

contenue dans la signature est différente de toutes les valeurs $h_1^{s_i}$ issues des signatures obtenues par $\mathcal{A}$ après ses requêtes à son oracle de signature Sign, avec $i \in \{1, \ldots, q_\mathsf{S}\}$ où q_S est le nombre de requêtes de signature faites par $\mathcal{A}$. En utilisant les mêmes techniques que précédemment, $\mathcal{B}$ est capable d'extraire les secrets A, x, x_u, s, r contenus dans la signature σ. Il obtient ainsi une signature (A, x) sur le bloc de message (m, s, x_u, r) qui est une forge pour le schéma de signature BBS_l^{ext}.

Cas 2 $\mathcal{B}$ suppose que $\mathcal{A}$ va renvoyer deux couples message/signature (m_1, σ_1) et (m_2, σ_2) qui lui permettent de casser la traçabilité du schéma avec probabilité non négligeable. Par définition, ces deux signatures contiennent la même valeur h_1^s et au moins un des secrets de la signature de connaissance diffère entre les deux signatures (*cf.* preuve d'inforgeabilité de la signature BBS_l^{ext}). Comme précédemment, $\mathcal{B}$ envoie à $\mathcal{A}$ les paramètres du système et simule les requêtes aux oracles. Nous ne détaillons ici que la réponse à la requête Sign, les autres réponses aux requêtes aux oracles étant identiques à celles du traçage d'identité.

$\mathsf{Sign}(\mathrm{Id}_\mathcal{U}, m)$: lorsque $\mathcal{A}$ envoie une requête à son oracle de signature, $\mathcal{B}$ simule les réponses de la manière suivante :

1. $\mathcal{B}$ extrait les valeurs m, r, s' des chiffrements de Paillier et retrouve la clé secrète $sk_\mathcal{U} = x_u$ associée à l'identité $\mathrm{Id}_\mathcal{U}$.
2. Il choisit ensuite une valeur $s'' \in_\mathrm{R} \mathbb{Z}_p$ et calcule la valeur $h_1^{s'+s''}$. Si cette valeur correspond à une valeur calculée au cours d'une requête précédente, alors $\mathcal{B}$ arrête la procédure. Sinon, $\mathcal{B}$ fait appel à son oracle de signature $\mathsf{Sign}_{\mathrm{BBS}_\mathrm{L}^{ext}}$ sur le bloc de message $(m, s = s' + s'', r, x_u)$ et reçoit en réponse une signature (A, x) qu'il renvoie à $\mathcal{A}$.

À la fin de son attaque, $\mathcal{A}$ renvoie deux couples message/signature si $\mathcal{B}$ n'a pas arrêté la procédure. Si $\mathcal{B}$ n'a pas arrêté la procédure Sign de $\mathcal{A}$, il sait qu'au moins une de ces signatures n'est pas une réponse à l'une de ses requêtes (dans le cas contraire, cela signifierait que $\mathcal{B}$ a produit deux signatures avec la même valeur h_1^s ce qui, de par le fonctionnement de $\mathcal{B}$, n'est pas possible). Il choisit alors l'une des signatures comme étant sa forge potentielle et le lemme de bifurcation et la significativité de la preuve de connaissance pok_2 nous permettent de dire que $\mathcal{B}$ est capable d'extraire les valeurs $(\mathrm{A}, x, x_u, s, r)$. $\mathcal{B}$ obtient ainsi sa forge pour une signature BBS_l^{ext}.

La probabilité que $\mathcal{B}$ s'arrête dans cette attaque est inférieure à q_S^2/p qui est négligeable.

Dans les deux cas, $\mathcal{B}$ réussit à casser l'inforgeabilité du schéma BBS_l^{ext} en utilisant un adversaire $\mathcal{A}$ contre le traçage de signature de notre schéma avec un avantage non négligeable.

9.2.6 Non-diffamation

Nous démontrons dans cette section la non-diffamation de notre schéma.

Théorème 22 (Non-diffamation)**.** *Le schéma de signature aveugle à anonymat révocable proposé est non-diffamable pour la signature et pour l'identité, dans le modèle de l'oracle aléatoire, sous l'hypothèse du logarithme discret.*

Nous n'utilisons plus dans cette preuve le schéma de signature BBS_l^{ext}. Nous montrons comment nous ramener directement au problème du logarithme discret pour prouver la sécurité de notre schéma. Nous utilisons cependant la même technique de réduction.

9.2.6.1 Non-diffamation d'identité

Afin de simplifier la description de notre preuve, nous faisons reposer ici la sécurité de notre schéma sur le problème du logarithme discret de plus. Toutefois, il est aisé de modifier cette preuve afin de faire reposer la sécurité sur le problème plus faible du logarithme discret.

Soit $\mathcal{B}$ un algorithme contre le problème du logarithme discret de plus qui utilise un attaquant $\mathcal{A}$ contre le propriété de non-diffamation pour mener à bien son attaque. $\mathcal{B}$ a accès à un oracle Dlog et reçoit en entrée son instance, à savoir l'ensemble $(q, h_2, \mathbb{G}_1, h_2^{x_1}, h_2^{x_2}, \ldots, h_2^{x_l})$. Il construit pour $\mathcal{A}$ les paramètres publics sauf les clés du signataire. $\mathcal{A}$ choisit $\gamma \in_{\mathrm{R}} \mathbb{Z}_p$ et pose $\Gamma = g_2^{\gamma}$. Il calcule aussi les clés $(sk_{\mathcal{RA}}, pk_{\mathcal{RA}})$ pour l'autorité de révocation.

$\mathcal{B}$ simule les réponses aux requêtes de $\mathcal{A}$ de la manière suivante :

$\mathsf{AddU}(\mathcal{U})$: lorsque $\mathcal{A}$ demande à son oracle d'ajouter un nouvel utilisateur, $\mathcal{B}$ lui renvoie une valeur $h_2^{x_i}$ de son entrée contre le logarithme discret de plus et ajoute cet utilisateur à la liste HU.

$\mathsf{CrptU}(\mathrm{Id}_{\mathcal{U}}, pk_{\mathcal{U}})$: $\mathcal{B}$ enregistre les valeurs que $\mathcal{A}$ lui envoie et ajoute l'utilisateur à la liste CU.

$\mathsf{USK}(\mathrm{Id}_{\mathcal{U}}, pk_{\mathcal{U}})$: $\mathcal{B}$ vérifie que $pk_{\mathcal{U}}$ correspond à l'identité d'un utilisateur honnête, sinon il arrête la procédure. Il interroge son oracle Dlog en lui donnant en entrée $pk_{\mathcal{U}}$ et renvoie la réponse à $\mathcal{A}$.

$\mathsf{User}(\mathrm{Id}_{\mathcal{U}}, pk_{\mathcal{U}}, m)$: dans le modèle de l'oracle aléatoire, $\mathcal{B}$ est capable de simuler parfaitement les réponses. La requête se faisant avec un utilisateur honnête, $\mathcal{B}$ ne connaît pas la clé secrète de cet utilisateur. Il doit donc simuler les preuves de connaissance en faisant appel à son oracle de hachage et en utilisant les méthodes de simulation que nous avons décrites dans la preuve d'indistinguabilité.

À la fin de son attaque, $\mathcal{A}$ renvoie une signature valide

$$h_1^s, \mathsf{Encrypt}_{sk_{\mathcal{RA}}}^{2\mathrm{EG}}(h_2^{x'_u}), pok_2(m, r, s, x, x'_u, \mathrm{A})$$

qui casse la non-diffamation d'identité de notre schéma après avoir fait l requêtes à l'oracle USK. Par définition, cette signature ne provient pas d'une requête User et l'identité associée n'a pas été demandée à l'oracle USK. En revanche l'ouverture d'identité renvoie sur une identité contenue dans la liste HU et donc dans l'instance du logarithme discret de plus donnée à $\mathcal{B}$. Si $\mathcal{A}$ a réussi à produire une signature valide avec probabilité

non négligeable, alors, d'après le lemme de séparation (*cf.* lemme 3, section 2.2.3), il existe un algorithme $\mathcal{A}'$ qui peut produire deux signatures

$$(h_1^s, \mathsf{Encrypt}^{\mathrm{2EG}}_{sk_{\mathcal{RA}}}(h_2^{x'_u}), pok_2(m, r, s, x, x'_u, \mathrm{A}))$$

et

$$(h_1^s, \mathsf{Encrypt}^{\mathrm{2EG}}_{sk_{\mathcal{RA}}}(h_2^{x'_u}), \tilde{pok}_2(m, r, s, x, x'_u, \mathrm{A}))$$

avec probabilité non négligeable. En utilisant ces valeurs et la preuve de significativité de pok_2, B est capable d'extraire les valeurs $(\mathrm{A}', s', r,', x'_u)$. $\mathcal{B}$ renvoie alors les k valeurs x_i qui proviennent de ses requêtes à son oracle Dlog ainsi que la valeur x'_u qui correspond au logarithme discret d'une des valeurs qu'il a reçu en instance. $\mathcal{B}$ a ainsi extrait $l+1$ logarithmes discrets de son instance après seulement l requêtes à l'oracle Dlog. Il a donc cassé le problème du logarithme discret de plus.

9.2.6.2 Non-diffamation de signature

De même, on construit un algorithme $\mathcal{B}$ contre le problème du logarithme discret utilisant un adversaire $\mathcal{A}$ qui réussit à casser la non-diffamation de signature avec probabilité non négligeable. $\mathcal{B}$ reçoit en entrée les paramètres $(p, h_1, \mathbb{G}_1)$ et son challenge (I, h_1). Il va alors utiliser $\mathcal{A}$ pour trouver $\log_{h_1} \mathrm{I}$. $\mathcal{B}$ choisit les clés pour le chiffrement de Paillier et fournit à $\mathcal{A}$ les paramètres suivants : (q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_\mathrm{T}$, g_1, g_2, h_1, h_2, h_3, h_4, n, g). Il génère également les clés pour l'autorité de révocation et les envoie à $\mathcal{A}$. Celui-ci choisit $\gamma \in_\mathrm{R} \mathbb{Z}_p$ et pose $pk_\mathcal{S} = \Gamma = g_2^\gamma$.

$\mathcal{B}$ simule les réponses aux requêtes de $\mathcal{A}$ de la manière suivante :

$\mathsf{AddU}(\mathcal{U}_i)$: $\mathcal{B}$ choisit $sk_{\mathcal{U}_i} = x_{u_i}$ et pose $pk_{\mathcal{U}_i} = h_2^{x_{u_i}}$. Il ajoute $sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}$ à la liste HU et renvoie $pk_{\mathcal{U}_i}$ à $\mathcal{A}$.

$\mathsf{CrptU}(\mathrm{I}d_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, sk_{\mathcal{U}_i})$ $\mathcal{B}$ enregistre $pk_{\mathcal{U}_i}, sk_{\mathcal{U}_i}$ dans la liste CU.

$\mathsf{USK}(\mathrm{I}d_\mathcal{U}, pk_{\mathcal{U}_i})$ $\mathcal{B}$ vérifie que $pk_{\mathcal{U}_i}$ correspond à une entrée de HU et renvoie, le cas échéant, la valeur x_{u_i} associée.

$\mathsf{User}(\mathrm{I}d_\mathcal{U}, pk_\mathcal{U}, m)$ $\mathcal{B}$ joue ici le rôle d'un utilisateur honnête face à $\mathcal{A}$. Il connaît donc la clé secrète x_u de cet utilisateur. Il simule la réponse de l'oracle comme suit :

1. $\mathcal{B}$ choisit $\alpha, \beta, r \in_\mathrm{R} \mathbb{Z}_p$ et calcule $\mathrm{C} = \mathrm{I}^\alpha h_1^\beta h_3^m h_4^r$.
2. Il calcule comme prévu par le protocole les valeurs M_1, M_2, M_3 et $\Delta'_1 = \mathsf{Encrypt}^{\mathsf{2EG}}_{pk_{\mathcal{RA}}}(\mathrm{I}^\alpha h_1^\beta)$ et les envoie à $\mathcal{A}$. Il simule la preuve de connaissance pok_1 (ce qu'il peut facilement faire dans le modèle de l'oracle aléatoire bien que ne connaissant pas la valeur $\log_{h_1} \mathrm{I}$).
3. Il reçoit de $\mathcal{A}$ un triplet (A, s'', x). Il vérifie que la signature (A, x) est une signature valide. Il est ensuite capable de parfaitement simuler la preuve pok_2 afin de produire une signature valide. Il ajoute dans une I-liste les valeurs $(\mathrm{I}_{sig} = h_1^{s''+\beta} \mathrm{I}^\alpha, \alpha, \beta, r, s")$.

À la fin de son attaque, $\mathcal{A}$ renvoie avec probabilité non négligeable, un couple message/signature (m, σ) qui casse la non-diffamation de signature du schéma. Par définition, cette signature n'a pas été produite lors d'une requête de $\mathcal{A}$ à l'oracle User. Il est

possible, comme précédemment, d'extraire de cette signature les valeurs $(\mathrm{A}, x, s, x_u, r)$. $\mathcal{B}$ sait que la valeur h_1^s correspond à l'un des identifiants I_{sig} qu'il a enregistrés dans sa I-liste. Par conséquent, $h_1^s = \mathrm{I}^{\alpha} h_1^{s"+\beta}$ pour des valeurs α et β qu'il connaît. Il obtient alors $\log_{h_1} \mathrm{I} = \frac{s-s''-\beta}{\alpha}$ sauf si $\alpha = 0$ ce qui ne se produit qu'avec probabilité $1/p$.

$\mathcal{B}$ réussit donc à casser le problème du logarithme discret avec probabilité non négligeable.

Quatrième partie

Applications à la monnaie électronique et aux coupons

Chapitre 10

Définitions et état de l'art

La monnaie et les coupons électroniques sont deux moyens de paiement très semblables dont le but est d'offrir aux consommateurs la possibilité de régler ou d'acheter en ligne des biens et des services, tout en préservant leur anonymat. La monnaie a pour vocation d'émuler parfaitement la monnaie classique, tandis que les coupons sont plutôt comparables à des droits d'accès (bons d'achats, cartes de fidélité, ...). La cryptographie permet de répondre à ces contraintes et offre des solutions pratiques pour la mise en place de tels systèmes.

Ce chapitre présente de manière générale ces deux systèmes ainsi que les différentes solutions proposées dans la littérature. Nous nous intéressons plus spécifiquement au schéma de monnaie électronique *Compact e-cash* de Camenisch, Hohenberger et Lysyanskaya [CHL05] présenté à Eurocrypt'05. En effet, les techniques utilisées dans cet article nous serviront pour construire les schémas présentés dans les chapitres suivants.

Sommaire

10.1 Monnaie électronique

Comme nous l'avons dit, la monnaie électronique cherche à reproduire le fonctionnement de la monnaie fiduciaire. Un système de monnaie se compose donc des même participants que ceux que l'on retrouve dans le circuit classique de la monnaie, à savoir une banque, des consommateurs et des marchands.

De même, le fonctionnement général repose sur des procédures identiques à celles de la monnaie classique. L'utilisateur retire de l'argent auprès de sa banque, il la dépense ensuite auprès de marchands et ceux-ci re-déposent les pièces dépensées auprès de leur banque ou les transfèrent à d'autres utilisateurs (en rendant la monnaie par exemple).

La monnaie électronique est, par exemple, très intéressante pour les transactions de montant peu élevé. Dans ce cas là, elle est plus avantageuse que l'utilisation d'une carte de paiement. En particulier, le coût de traitement des transactions doit être faible afin que les petites dépenses soient possibles sans pénaliser le marchand.

10.1.1 Sécurité

Le souci d'anonymat est la motivation principale de la monnaie électronique. Tout comme une pièce de monnaie ne comporte aucune information sur son porteur, une pièce électronique ne doit rien divulguer sur son détenteur au moment où elle est dépensée. Lors d'un règlement par carte bancaire, un certains nombre d'informations sur le consommateur et le marchand sont envoyées à la banque. Dans un système de monnaie électronique, seules des informations publiques peuvent apparaître de manière claire au cours de la transaction.

C'est pourquoi la première propriété que doit respecter un schéma de monnaie électronique est *l'anonymat*. Cet anonymat doit être respecté quand bien même une banque malhonnête s'associerait avec des utilisateurs ou des marchands malhonnêtes. De surcroît, personne ne doit être capable de déterminer si deux transactions ont été effectuées par le même utilisateur ou non.

De même, tout comme dans la monnaie fiduciaire, il doit être impossible de créer de la fausse monnaie. Une schéma de monnaie électronique doit donc respecter la propriété de *non-falsification* (ou inforgeabilité).

Cependant, ce moyen de paiement fait apparaître un nouveau problème. Les pièces de monnaie classiques sont des données physiques qu'il n'est pas possible de dépenser deux fois. En revanche, une pièce électronique est une valeur numérique qui elle peut être dupliquée à l'infini et représente toujours la même pièce. C'est pourquoi, dans le cadre de la monnaie électronique, il est important de se protéger contre ce qu'on appelle les *double dépenses*. La banque doit être capable, lorsqu'elle reçoit une pièce d'un marchand, de déterminer si cette pièce a déjà été déposée ou non. La double dépense étant une fraude, il est aussi nécessaire de retrouver la personne qui en est à l'origine, que ce soit un marchand ou un consommateur. Ceci est fait, selon les schémas, par la banque elle même, ou par un tiers de confiance qui n'est pas impliqué dans les procédures de dépenses et de retraits.

En outre, afin d'éviter tout abus, il doit être impossible pour quiconque, de faire accuser à tort une personne d'avoir commis une double dépense. Cette propriété, appelée *non-diffamation*, inclut aussi le fait que la banque ne doit pas être en mesure d'imputer des retraits à un utilisateur alors que ce dernier y est totalement étranger.

10.1.2 Formalisation

Comme nous l'avons vu, un système de monnaie (électronique) se compose de trois entités :

- la banque $\mathcal{B}$ qui a pour rôle d'émettre la monnaie électronique à qui de droit et de collecter les pièces déposées par les marchands,
- un consommateur, ou utilisateur, $\mathcal{U}$ qui retire de la monnaie auprès de la banque (si son compte est suffisamment approvisionné) et le dépense ensuite auprès de marchands,
- un marchand $\mathcal{M}$ qui reçoit des pièces électroniques de la part des utilisateurs en échange de biens ou services et les dépose auprès de la banque, pour compensation de leurs valeurs.

Un système de monnaie électronique est composé des trois étapes principales suivantes : le retrait, la dépense et le dépôt. Le formalisme que nous décrivons ici repose sur la conception actuelle de la monnaie électronique, telle qu'introduite dans [CHL05]. Il ne s'applique donc pas directement aux constructions proposées auparavant.

- Lors de l'opération de retrait, l'utilisateur obtient de la banque une pièce validée ou un porte-monnaie validé qui lui permet de dépenser plusieurs pièces. Cette validation est réalisée à l'aide d'une signature électronique. Une pièce est alors représentée par un numéro de série et une signature de la banque. Cette signature porte, selon les cas, sur le numéro de série de la pièce elle-même ou sur le porte-monnaie. Afin d'améliorer l'utilisation pratique d'un tel système, nous donnons à l'utilisateur la possibilité de choisir la valeur et le nombre de pièces qu'il souhaite retirer (l'ensemble des valeurs possibles pour une pièce est fixé à l'avance dans les paramètres du système).
- Lors de l'opération de dépense, l'utilisateur doit prouver au marchand qu'il utilise bien une pièce validée par la banque, ou issue d'un porte-monnaie validé par la banque. Afin de préserver son anonymat, il ne peut pas révéler la signature de la banque, ni son identité. Il prouve donc au marchand sa connaissance d'une telle signature.
- Lors de l'opération de le marchand dépose sa pièce et la banque ne doit pas être capable de reconnaître la pièce qu'elle reçoit (notamment la signature associée à la pièce) ni le numéro de série afin d'éviter toute traçabilité des dépenses.

A ces procédures, il est nécessaire d'ajouter les méthodes permettant de détecter les doubles dépenses et de tracer les marchands ou utilisateurs frauduleux. Et afin d'éviter toute accusation non justifiée, un schéma de monnaie électronique doit aussi définir une procédure permettant de déterminer si une personne donnée est coupable ou non.

Nous considérons, dans notre formalisme, que l'utilisateur retire un porte-monnaie auprès de la banque.

Définition 93 (Système de monnaie électronique)**.** *Un système de monnaie électronique se construit à l'aide des procédures suivantes :*

- Setup*, phase de mise en place des paramètres du système, des paramètres publics et de construction des clés pour les différents participants. Les paramètres publics params contiennent au moins un paramètre de sécurité k, un entier n correspondant au nombre de valeurs monétaires possibles pour les pièces, l'ensemble*

$\{\mathrm{V}_i, i \in 1,n\}$,*des différentes valeurs possibles et une valeur* $\mathrm{M}ax$ *qui détermine le nombre maximal de pièces qu'un utilisateur peut retirer. Les clés sont générées à l'aide des algorithmes suivants :*

- $\mathsf{G}_{\mathcal{B}}$*, algorithme de génération de clés pour la banque. Il prend en entrée le paramètre de sécurité* k *et les paramètres publics et renvoie une clé privée* $sk_{\mathcal{B}}$ *et la clé publique associée* $pk_{\mathcal{B}}$ *pour la banque.*
  $(sk_{\mathcal{B}}, pk_{\mathcal{B}}) \leftarrow \mathcal{G}_{\mathcal{B}}(1^k, params)$.
- $\mathsf{G}_{\mathcal{M}}$*, algorithme de génération de clés pour le marchand. Il prend en entrée le paramètre de sécurité* k *et les paramètres publics et renvoie une clé privée* $sk_{\mathcal{M}}$ *et la clé publique associée* $pk_{\mathcal{M}}$ *pour le marchand.*
  $(sk_{\mathcal{M}}, pk_{\mathcal{M}}) \leftarrow \mathcal{G}_{\mathcal{M}}(1^k, params)$.
- $\mathsf{G}_{\mathcal{U}}$*, algorithme de génération de clés pour l'utilisateur. Il prend en entrée un paramètre de sécurité* k *et les paramètres publics et renvoie une clé privée* $sk_{\mathcal{U}}$ *et la clé publique associée* $pk_{\mathcal{U}}$ *pour l'utilisateur.*
  $(sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathcal{G}_{\mathcal{U}}(1^k, params)$.

- Withdraw*, protocole interactif de retrait d'un porte-monnaie entre la banque* $\mathcal{B}$ *et l'utilisateur* $\mathcal{U}$*. La banque donne en entrée sa clé privée* $sk_{\mathcal{B}}$ *et sa clé publique* $pk_{\mathcal{B}}$*. Pour chaque* $i \in [1,n]$*, l'utilisateur choisit le nombre* J_i *de pièces de valeur* V_i *qu'il veut retirer. Il donne en entrée ces valeurs* J_i *ainsi que ses clés* $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$*. Si les deux parties acceptent le protocole, la sortie de l'utilisateur est un porte-monnaie* W*, c'est-à-dire un identifiant* I *comprenant, entres autres, la signature de la banque, et un ensemble* $\mathcal{S} = \{(\mathrm{J}_i, \mathrm{V}_i), i \in [1,n]\}$*. La sortie de la banque est une vue* $\mathcal{V}_{\mathcal{B}}^{\mathsf{Withdraw}}$ *du protocole. Sinon, le protocole échoue.*
  $(\mathcal{U}(\mathrm{W} = (\mathrm{I}, \mathcal{S})), \mathcal{B}(\mathcal{V}_{\mathcal{B}}^{\mathsf{Withdraw}})) \leftarrow \mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{U}}, sk_{\mathcal{U}}, \mathrm{J}_1, \ldots, \mathrm{J}_n), \mathcal{B}(pk_{\mathcal{B}}, sk_{\mathcal{B}}))$
- Spend*, protocole de dépense interactif entre l'utilisateur* $\mathcal{U}$ *et le marchand* $\mathcal{M}$*. L'utilisateur fournit en entrée un porte-monnaie* W *et ses clés* $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$*. Il choisit aussi la valeur* V_j *de la pièce qu'il souhaite dépenser. Le marchand donne en entrée ses clés* $(sk_{\mathcal{M}}, pk_{\mathcal{M}})$*. Si la pièce est acceptée, l'utilisateur reçoit un porte-monnaie mis à jour* W′*, c'est-à-dire le même identifiant* I *et l'ensemble* $\mathcal{S}' = \{(\mathrm{J}'_i, \mathrm{V}_i), i \in [1,n]\}$ *où* $\mathrm{J}'_j = \mathrm{J}_j - 1$ *et* $\mathrm{J}'_i = \mathrm{J}_i$*,* $i \in [1,n]$ *et* $i \neq j$*. Sinon, l'algorithme renvoie un message d'erreur. Le marchand obtient un numéro de série* S *d'une pièce de valeur* V_j *et une preuve de validité* π*. Sinon, l'algorithme renvoie un message d'erreur.*
  $(\mathcal{U}(\mathrm{W}' = (\mathrm{I}, \mathcal{S}')), \mathcal{M}(\mathrm{S}, \mathrm{V}_j, \pi)) \leftarrow \mathsf{Spend}(\mathcal{U}(\mathrm{W}, pk_{\mathcal{U}}, sk_{\mathcal{U}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{M}}))$
- Deposit*, protocole de dépôt d'une pièce. Le marchand* $\mathcal{M}$ *donne en entrée un numéro de série* S *et une preuve* π *correspondant à la pièce qu'il souhaite déposer sur le compte qu'il possède auprès de la banque* $\mathcal{B}$ *et ses clés* $(sk_{\mathcal{M}}, pk_{\mathcal{M}})$*. La banque* $\mathcal{B}$ *donne en entrée ses clés* $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$*. Elle vérifie la validité de la pièce et la dépose sur le compte de* $\mathcal{M}$*. Sinon* $\mathcal{M}$ *reçoit un message d'erreur de la banque.*
  $(\mathcal{M}(update\ count), \mathcal{B}(\mathrm{S}, \pi)) \leftarrow \mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{M}}, \mathrm{S}, \pi), \mathcal{B}(sk_{\mathcal{B}}, pk_{\mathcal{B}}))$
- Identify*, algorithme permettant à la banque d'identifier les fraudeurs cherchant à double-dépenser une pièce de numéro de série* S *associée aux deux preuves de validité* π_1 *et* π_2*. Il prend en entrée les paramètres publics* $params$*, l'identifiant* S *et les deux preuves* π_1 *et* π_2*. Il renvoie une clé publique* $pk_{\mathcal{U}}$ *et une preuve de culpabilité* Π_{G}*. Si les deux preuves* π_1 *et* π_2 *ont été fournies par le même marchand et que celui-ci n'est pas le fraudeur, alors* Π_{G} *est une preuve que* $pk_{\mathcal{U}}$ *est la clé*

publique d'un utilisateur et que cet utilisateur est à l'origine de la tentative de double-dépense.
$(pk_{\mathcal{U}}, \Pi_{\mathrm{G}}) \leftarrow \mathsf{Identify}(params, \mathrm{S}, \pi_1, \pi_2)$

- $\mathsf{VerifyGuilt}$*, algorithme permettant de vérifier publiquement la preuve* Π_{G} *que l'utilisateur dont la clé publique est* $pk_{\mathcal{U}}$ *est coupable d'avoir double-dépensé la pièce* S.

On spécifie alors un schéma de monnaie électronique comme étant l'ensemble :

$$\mathrm{ME} = (\mathsf{Setup}, \mathsf{Withdraw}, \mathsf{Spend}, \mathsf{Deposit}, \mathsf{Identify}, \mathsf{VerifyGuilt}).$$

10.1.3 État de l'art

En 1982, en proposant le premier schéma de signature aveugle, Chaum s'intéressait à construire un système de monnaie électronique ayant les propriétés suivantes :

- la banque ne peut pas déterminer le destinataire, l'heure ou le montant des paiements faits par une personne,
- les utilisateurs peuvent prouver qu'ils sont à l'origine de leurs paiements en cas de litige,
- le moyen de paiement peut être mis en opposition s'il est volé.

La question de double-dépense n'était pas soulevée dans cet article. Par la suite, Chaum a proposé deux nouveaux schémas, afin d'améliorer l'efficacité du premier ([Cha83], [Cha85]). Cependant, ces schémas sont tous les trois, ce qu'on appelle des schémas *en ligne*, c'est-à-dire nécessitant la présence d'un tiers de confiance pour la dépense d'une pièce, ce qui alourdit la procédure en conséquence. Le premier schéma *hors ligne* (c'est-à-dire sans tiers de confiance) est dû à Chaum, Fiat et Naor [CFN88]. Ce schéma est le premier aussi à définir des procédures de détection de doubles dépenses ainsi que de traçabilité des fraudeurs. En revanche, de par sa construction, ce schéma n'est pas efficace.

Par la suite, afin de proposer des schémas efficaces et implémentables, de nombreux auteurs ont basé leurs constructions sur des schémas de signature aveugle, comme l'avait fait Chaum dans [Cha82]. C'est le cas par exemple de Frankel et Yung qui, dans [FY93] ont aussi proposé le premier modèle formel pour la monnaie électronique.

La même année Chaum et Pedersen ont introduit le concept de *porte-monnaie avec observateur* dans [CP92]. Les schémas proposés jusque là permettaient de détecter les double dépenses a posteriori, ce qui n'a qu'un effet dissuasif. Il est, en effet, plus intéressant de pouvoir prévenir les comportements frauduleux. Dans cette solution, l'*observateur* prend part à toutes les procédures de dépense et refuse de coopérer lorsqu'un utilisateur essaye de dépenser deux fois la même pièce. Brands a proposé dans [Bra93] une solution plus efficace d'un tel système.

Dans toutes ces solutions, il est impossible à quiconque, que ce soit la banque, les marchands, les utilisateurs ou une personne extérieure au système, de lever l'anonymat sur une transaction donnée. Cependant, comme nous l'avons vu pour les signatures aveugles (*cf* chapitre 7), cet anonymat fort peut être utilisé à des fins malhonnêtes [vSN92] et servir notamment au blanchiment d'argent. Afin de contrer ce problème nous avons présenté dans la partie III une variante des signatures aveugles, les signatures aveugles à anonymat révocable, qui permettent de tracer les utilisateurs indélicats.

Ces signatures introduites par Stadler, Piveteau et Camenisch [SPC95] ont trouvé une application directe dans ce que Brickell, Gemmel et Kravitz ont appelé *la monnaie équitable* [BGK95]. Ces systèmes comprennent une autorité de révocation habilitée à lever l'anonymat des pièces retirées à la banque et/ou utilisées lors d'un paiement. Cette notion de monnaie équitable à l'aide de signatures aveugles a été reprise dans de nombreuses constructions ([CMS96], [FTY96], [FTY98], [GT03]).

Parallèlement à ces constructions basées sur les signatures aveugles, Traoré a proposé en 1995, la première construction basée sur les signatures de groupe [Tra95]. Cette construction permet elle aussi de donner des schémas équitables, non falsifiables et anonymes. Cette technique a été exploitée aussi dans [Tra99], [MB01], [CL01] et [CT03b]. Ces solutions n'utilisent pas toutes les signatures de groupe de la même manière et certaines ([Tra99], [MB01]) nécessitent en plus, l'utilisation de signatures aveugles.

Une approche complémentaire, proposée par Lysyankaya et Ramzan [LR98] repose sur des signatures de groupe aveugles, c'est-à-dire une composition des signatures aveugles et des signatures de groupe. Ce système autorise la co-existence de plusieurs banques pour les retraits et les dépenses. Traoré dans [Tra99] a montré que ce schéma comportait des faiblesses dans sa construction.

Cependant, toutes ces constructions quelles qu'elles soient, permettent uniquement de retirer les pièces une par une. Par conséquent, la taille du porte-monnaie est linéairement proportionnelle au nombre de pièces retirés. En 2005, Camenisch, Hoenberger et Lysyanskaya [CHL05] ont produit une avancée importante en proposant un schéma permettant de retirer 2^l pièces en une seule fois, pour un porte-monnaie de taille $\mathcal{O}(l.k)$, où l est une valeur fixée par le système et k est un paramètre de sécurité. Nous détaillons plus précisément l'une des deux solutions de l'article *Compact e-cash* dans le suite de ce chapitre. Cette construction et les techniques utilisées ont ensuite été reprises afin de proposer de nouvelles propriétés pour la monnaie et les coupons. C'est le cas de Au *et al.* , qui dans [ASM07] présentent une solution au problème de la dépense simultanée de plusieurs pièces.

C'est aussi notre cas et nous montrerons, dans les chapitres suivants, comment ce schéma nous a permis de proposer le premier protocole de monnaie et de multi-coupons tel que les pièces et les coupons puissent avoir des valeurs différentes sans que la taille du porte-monnaie ne soit fixée par le système.

Tous les schémas que nous avons présentés jusque là reposent sur le principe selon lequel l'utilisateur retire plusieurs pièces pour ensuite les dépenser une à une. En 1989, Okamoto a introduit une nouvelle manière de concevoir la monnaie électronique [OO89a], appelée *monnaie divisible* ou *divisible e-cash*. Dans cette construction, l'utilisateur retire une seule pièce qui représente un certain montant. Il peut ensuite la dépenser en plusieurs fois en divisant la valeur de la pièce initiale, jusqu'à ce que la somme totale de ses dépenses soit égale au montant de la pièce retirée. Okamoto a proposé par la suite une nouvelle construction dans [Oka95]. Chan *et al.* dans [CFT98] ont proposé une amélioration de l'efficacité de [Oka95]. Cependant ce schéma est traçable, dans le sens où les dépenses peuvent être reliées entre elles. Le premier schéma de monnaie divisible non traçable est dû à Nakanishi et Sugiyama [NS00]. En revanche, il ne garantit pas un anonymat total des utilisateurs. Canard et Gouget ont montré récemment dans [CG07] que les schémas de monnaie divisible pouvaient être totalement

anonymes.

10.1.4 Compact E-cash

L'article *Compact E-cash* de Camenisch, Hoenberger et Lysyanskaya est le premier à avoir présenté un moyen efficace de rendre le coût des retraits non linéairement dépendant du nombre de pièces retirées. En outre, la taille du porte-monnaie n'est, elle aussi, pas linéairement liée au nombre de pièces retirées. Camenisch *et al.* proposent deux constructions différentes. Dans la première, la banque peut détecter efficacement les doubles dépenses et identifier rapidement les utilisateurs indélicats. La deuxième permet en outre de retrouver toutes les pièces dépensées par ce fraudeur. Nous ne décrivons ici que le premier schéma.

Leur construction repose sur l'utilisation de signature Camenish-Lysyanskaya permettant à la banque de signer des valeurs engagées par l'utilisateur (*cf.* section 3.4). L'identifiant de chaque pièce est calculé au moment de la dépense à l'aide de la fonction pseudo-aléatoire de Dodis-Yampolskiy (*cf.* section 3.1.2). Ainsi, le porte-monnaie ne contient que la graine permettant d'évaluer l'identifiant de la pièce, ce qui permet, d'une part, de simplifier le protocole de retrait, et, d'autre part, de rendre la taille du porte-monnaie indépendante du nombre de pièces retirées. C'est cette graine qui est signée par la banque de manière aveugle. L'utilisateur doit ensuite convaincre le marchand qu'il connaît une telle valeur signée par la banque sans révéler la signature, ni la graine.

10.1.4.1 Protocole

Mise en place des paramètres : Setup

Dans ce système, la taille du porte-monnaie est un paramètre du système et toutes les pièces de monnaie ont la même valeur. Il n'est donc pas nécessaire de spécifier les valeurs n et V_i données dans le formalisme.

Les paramètres du système sont donnés par les valeurs suivantes :

- k, le paramètre de sécurité et l une valeur en $\mathrm{O}(\log k)$. Max est alors fixé à 2^l.
- G, un groupe d'ordre n, où n est un module RSA de taille $2k$, g un résidu quadratique modulo n, générateur de G et $\mathsf{h} \in \mathsf{G}$.
- $\mathbb{G}$, un groupe de générateur g d'ordre premier q en $0(2^k)$ et h, un élément de $\mathbb{G}$ tel que le problème décisionnel Diffie-Hellman soit difficile dans $\mathbb{G}$.
- $\mathbb{G}_1$ et $\mathbb{G}_2$, deux groupes de même ordre premier q et de générateurs g_1 et g_2 respectivement. On suppose de plus l'existence d'une application bilinéaire admissible[1] e telle que : $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}$.
- h, h_1, h_2, h_3, des éléments de $\mathbb{G}_1$.

$\mathsf{G}_{\mathcal{B}}$ **:** la banque utilise l'algorithme $\mathsf{G}_{\mathcal{B}}$ pour obtenir ses clés de signature : $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$. Celles-ci correspondent à des clés pour une signature Camenisch-Lysyanskaya.

[1] Camenisch *et al.* privilégient l'utilisation des signatures BBS [BBS04] ou CL [CL04] dans leur construction, c'est pourquoi il est nécessaire de vérifier l'existence d'une application bilinéaire admissible. Il est cependant possible de s'en passer en utilisant une signature ACJT [ACJT00] pour la construction de la signature Camenisch-Lysyanskaya.

$\mathsf{G}_{\mathcal{U}}$ **:** chaque utilisateur utilise l'algorithme $\mathsf{G}_{\mathcal{U}}$ pour obtenir ses clés :
- $sk_{\mathcal{U}} = u$ avec $u \in \mathbb{Z}_q$,
- $pk_{\mathcal{U}} = g^u$ dans $\mathbb{G}$.

$\mathsf{G}_{\mathcal{M}}$ **:** chaque marchand utilise l'algorithme $\mathsf{G}_{\mathcal{M}}$ pour obtenir ses clés :
- $sk_{\mathcal{M}} = m$ avec $m \in \mathbb{Z}_q$,
- $pk_{\mathcal{M}} = g^m$ dans $\mathbb{G}$.

Retrait d'un porte-monnaie : Withdraw

L'utilisateur $\mathcal{U}$ et la banque $\mathcal{B}$ interagissent de manière à ce que l'utilisateur se retrouve, en fin de procédure, en possession d'un porte-monnaie de taille 2^l. Le protocole se déroule comme présenté à la figure 10.1.

1. Tout d'abord, $\mathcal{U}$ s'authentifie auprès de la banque $\mathcal{B}$ en prouvant sa connaissance de sa clé secrète $sk_{\mathcal{U}}$ associée à sa clé publique $pk_{\mathcal{U}}$. La banque peut ensuite vérifier que le compte de $\mathcal{U}$ est suffisamment approvisionné avant de décider si elle poursuit la procédure ou non.
2. $\mathcal{U}$ choisit ensuite sa participation $s' \in [0, q-1]$ à un secret partagé. Il choisit aussi un secret $t \in [0, q-1]$, associé au porte-monnaie qu'il est en train de retirer. Il envoie à $\mathcal{B}$ un engagement $\mathrm{C}' = \mathsf{PedCom}(sk_{\mathcal{U}}, s', t; r)$ sur sa clé secrète $sk_{\mathcal{U}}$ et sur les deux valeurs secrètes s' et t, en prenant comme aléa $r \in [0, q-1]$.
3. $\mathcal{B}$ choisit sa participation $r' \in [0, q-1]$ au secret partagé et calcule ensuite l'engagement $\mathrm{C} = \mathrm{C}' g^r$, c'est-à-dire un engagement sur $sk_{\mathcal{U}}, s'+r', t, s$. Il calcule ensuite une signature Camenisch-Lysyanskaya $\sigma_{\mathcal{B}}$ de cet engagement et envoie à $\mathcal{U}$ la valeur r' et la signature $\sigma_{\mathcal{B}}$.
4. $\mathcal{U}$ calcule le secret final $s = s' + r'$ et est en possession d'une signature $\sigma_{\mathcal{B}}$ de la banque, sur les valeurs $(sk_{\mathcal{U}}, s, t, r)$.
5. $\mathcal{U}$ sauvegarde son porte-monnaie comme étant $\mathrm{W} = (sk_{\mathcal{U}}, s, t, \sigma_{\mathcal{B}}, \mathrm{J})$[2], où s et t sont les secrets du porte-monnaie, $\sigma_{\mathcal{B}}$ est une signature de la banque et J est un compteur de taille l, initialisé à 0.
6. La banque enregistre un débit de 2^l sur le compte associé à la clé publique $pk_{\mathcal{U}}$.

Dépense d'une pièce : Spend

$\mathcal{U}$ dépense une pièce de monnaie auprès d'un marchand $\mathcal{M}$ de la façon suivante :

1. $\mathcal{U}$ calcule $\mathrm{R} = \mathrm{H}(pk_{\mathcal{M}} || info)$ où $info \in \{0,1\}^*$ représente des informations fournies par le marchand.
2. $\mathcal{U}$ calcule le numéro de série S de la pièce et un tag de sécurité T à l'aide de la fonction pseudo-aléatoire de Dodis-Yampolskiy F. Pour calculer S, $\mathcal{U}$ prend comme graine la valeur s et évalue la fonction en J, où J représente le rang de la pièce qu'il est en train de dépenser. Afin d'éviter toute reliabilité des dépenses, J ne doit pas être dévoilée. Par conséquent, $\mathcal{U}$ devra aussi prouver que la valeur J

[2]La taille et les valeurs des pièces étant identiques, le porte-monnaie est uniquement composé de l'identifiant I.

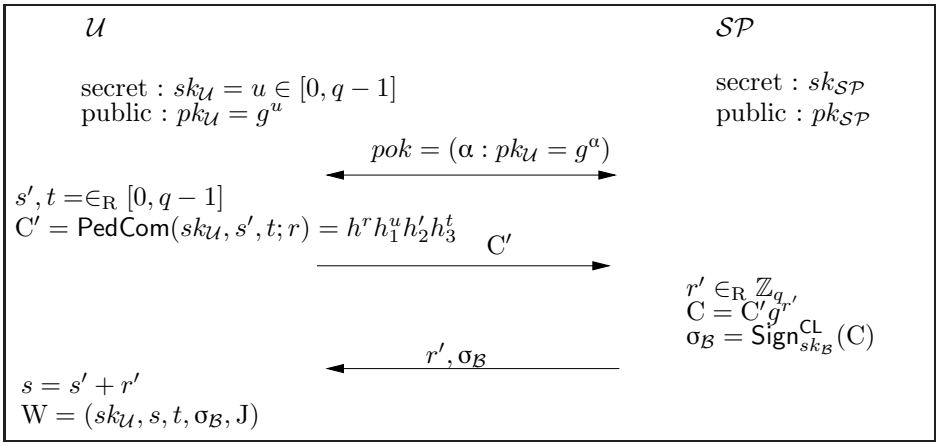

FIG. 10.1 – Protocole de Retrait de *Compact e-cash*

appartient bien à l'intervalle $[0, 2^l - 1]$, afin de montrer qu'il est bien en train de dépenser une pièce contenue dans son porte-monnaie.

T est calculé lui aussi à l'aide de la fonction de la F. La graine est alors t, et la fonction est évaluée comme précédemment en J. $\mathcal{U}$ ajoute aussi sa clé publique et l'aléa R à l'équation, afin d'engager sa responsabilité dans la dépense.

Les valeurs S et T sont donc :

$$\mathrm{S} = \mathrm{F}^{\mathrm{DY}}_{g,s}(\mathrm{J}) = g^{\frac{1}{\mathrm{J}+s+1}}$$

$$\mathrm{T} = pk_{\mathcal{U}} \mathrm{F}^{\mathrm{DY}}_{g,t}(\mathrm{J})^{\mathrm{R}} = pk_{\mathcal{U}} g^{\frac{\mathrm{R}}{\mathrm{J}+t+1}}$$

3. $\mathcal{U}$ doit ensuite prouver au marchand que son porte-monnaie est valide, c'est-à-dire sa connaissance d'une signature sur les secrets (s, t) et que J est bien dans l'intervalle $[0, 2^l - 1]$, tout en ne révélant pas la valeur J. Cette preuve de connaissance notée Φ est en faite une signature de connaissance.
4. Si $\mathcal{M}$ accepte la preuve Φ , alors il accepte la pièce $(\mathrm{S}, (\mathrm{R}, \mathrm{T}, \Phi))$.
5. $\mathcal{U}$ met à jour son compteur $\mathrm{J} = \mathrm{J} + 1$. Si $\mathrm{J} > 2^l - 1$, alors son porte-monnaie est vide.

La preuve de connaissance Φ décrite dans l'étape 3 se déroule de la manière suivante :

1. $\mathcal{U}$ calcule tout d'abord des engagements sur les valeurs secrètes (J, u, s, t) que l'on note T_{J}, T_u, T_s et T_t.
2. La preuve Φ comprend alors :
   (a) une preuve de connaissance sur les valeurs engagées T_{J}, T_u, T_s et T_t, c'est-à-dire la connaissance de $\alpha = \mathrm{J}$, $\beta = u$, $\gamma = s$, $\delta = t$ tels que T_{J}, T_u, T_s et T_t sont consistants ;
   (b) une preuve de connaissance de la signature de $\mathcal{B}$ sur les valeurs engagées (J, u, s, t). Cette preuve peut être faite comme décrite en section 3.4 ;

(c) une preuve que S et T sont bien construits à l'aide des valeurs engagées, c'est-à-dire la preuve de connaissance

$$pok\big(\alpha, \beta, \delta, \gamma_1, \gamma_2, \gamma_3 : \mathsf{g} = (\mathrm{T_J T}_s)^{\alpha}\mathsf{h}^{\gamma_1} \wedge \mathrm{S} = g^{\alpha} \\ \wedge\, \mathsf{g} = (\mathrm{T_J T}_t)^{\beta}\mathsf{h}^{\gamma_2} \wedge \mathrm{T}_u = \mathsf{g}^{\delta}\mathsf{h}^{\gamma_3} \wedge \mathrm{T} = g^{\delta}(g^{\mathrm{R}})^{\beta}\big);$$

(d) une preuve que J appartient au bon intervalle, en utilisant la preuve qu'une valeur engagée est plus petite qu'une autre valeur, décrite en section 3.3.2.1.

En utilisant l'heuristique de Fiat-Shamir, toutes les preuves ci-dessus sont converties en signature de connaissance des valeurs (S,T, $\mathrm{T_J}$, T_u, T_s, T_t, g, g, n, g, $pk_{\mathcal{M}}$, R, $info$). C'est cette signature finale qui constitue la preuve Φ.

Le marchand reçoit donc, en fin de procédure, une pièce $(\mathrm{S}, (\mathrm{R}, \mathrm{T}, \Phi))$ et $\mathcal{U}$ est en possession d'un porte-monnaie mis à jour, dans le sens où son compteur J est incrémenté. Dans la description de nos algorithmes, nous considérons que le compteur est décrémenté à chaque dépense. Afin de respecter la description de Camenisch *et al.* nous conservons, ici l'incrémentation de J.

Dépôt d'une pièce : Deposit

Afin de déposer une pièce auprès de la banque, le marchand agit de la façon suivante. Il envoie à la banque $\mathcal{B}$ la pièce $(\mathrm{S}, \pi = (\mathrm{R}, \mathrm{T}, \Phi))$. Si Φ est valide et R n'est pas une valeur déjà connue (autrement dit, la paire (S, R) n'est pas déjà enregistrée dans la liste L des pièces dépensées), alors, $\mathcal{B}$ accepte la pièce pour dépôt, ajoute (S, π) à la liste L et crédite le compte de $\mathcal{M}$. Dans le cas contraire, $\mathcal{B}$ renvoie un message d'erreur à $\mathcal{M}$.

Identification des fraudeurs : Identify

Supposons que la banque ait deux entrées $(\mathrm{R}_1, \mathrm{T}_1) \in \pi_1$ et $(\mathrm{R}_2, \mathrm{T}_2) \in \pi_2$ dans sa base L des pièces dépensées, correspondant toutes deux à un même numéro de série S. Alors, la banque renvoie la preuve de culpabilité $\Pi_{\mathrm{G}} = (\pi_1, \pi_2)$ et l'identité $pk = (\mathrm{T}_2^{\mathrm{R}_1}/\mathrm{T}_1^{\mathrm{R}_2})^{(\mathrm{R}_1 - \mathrm{R}_2)^{-1}}$.

Vérification de culpabilité : VerifyGuilt

Afin d'identifier le fraudeur, la banque, ou l'autorité mandatée, sépare la preuve Π_{G} en (π_1, π_2) et chaque preuve π_i en $(\mathrm{R}_i, \mathrm{T}_i, \Phi_i)$. Elle exécute ensuite le protocole Identify en donnant en entrée, les paramètres du système, le numéro de série de la pièce double-dépensée S et les deux preuves π_1 et π_2. Elle compare ensuite la sortie de l'algorithme à la clé publique de la personne soupçonnée. Si les deux correspondent, elle vérifie la validité des preuves Φ_i. Si elles sont toutes deux valides, alors le fraudeur est coupable. Autrement, la procédure est arrêtée.

10.1.4.2 Sécurité du schéma

La sécurité du schéma repose sur l'hypothèse RSA fort et l'hypothèse y-DDHI. Le schéma est consistant. Il permet de détecter les doubles dépenses et fournit un anonymat

fort aux utilisateurs. Il les protège aussi contre toute diffamation. La sécurité du schéma est montrée dans le modèle de l'oracle aléatoire en respectant les définitions de sécurité proposées par Camenisch *et al.* dans leur article [CHL05].

10.2 Coupons et multi-coupons

Après avoir décrit le fonctionnement d'un schéma de monnaie électronique, nous nous intéressons dans cette section aux systèmes de coupons et multi-coupons.

Avec le développement du commerce en ligne, il est de plus en plus important pour les commerçants de fidéliser leur clientèle. En effet, les consommateurs n'ayant plus besoin de se déplacer et n'ayant pas de contacts directs avec les commerçants, sont amenés à changer beaucoup plus aisément de fournisseurs. Cependant, les attentes des deux parties, à savoir les consommateurs et les marchands, ne sont pas les mêmes. L'utilisateur souhaite utiliser les services qu'on lui propose tout en préservant son anonymat, notamment afin d'éviter toute création de profils. Le marchand, quant à lui, désire établir une relation à long terme avec ses clients.

C'est dans ce contexte que Chen *et al.* ont introduit à, *Financial Crypto'05* [CES+05], le concept de coupons anonymes. Ils définissent un coupon comme étant le droit de réclamer un service ou un bien proposé par le fournisseur qui a délivré ce coupon, tout en préservant l'anonymat de l'utilisateur. Les concepts de coupons et de monnaie électronique sont donc très proches : la monnaie est émise par la banque tandis qu'un coupon est émis par un fournisseur de services. De ce fait, la construction d'un système de coupon est très semblable à celle d'un schéma de monnaie. Mais, contrairement à la monnaie, un système de coupons ne se compose que de deux types de participants : un marchand (ou fournisseur de services) et les utilisateurs. Ces derniers reçoivent ou retirent leurs coupons auprès du fournisseur et ne peuvent ensuite les dépenser qu'auprès de celui-ci. Lorsqu'un ensemble de coupons est retiré en une seule transaction, on appelle alors cet ensemble un *multi-coupons*.

L'application la plus directe des coupons est donc d'être un moyen de fidélisation d'une clientèle. Les coupons peuvent alors représenter des bons de réduction, des avoirs, ou des carnets de tickets de cinéma.

Mais ils peuvent aussi être utilisés dans un contexte tout à fait différent et servir à établir des ordonnances médicales. Dans ce cas, chaque coupon représente la prescription d'un médicament, le multi-coupons étant l'ordonnance totale. De ce fait, le patient peut acheter ses médicaments tout en restant anonyme vis à vis du pharmacien.

Nous verrons par la suite, comment ces différentes applications modifient la façon de concevoir un schéma de coupons.

10.2.1 Sécurité

Les coupons étant une monnaie d'échange, la première propriété qu'ils doivent respecter est d'être *non forgeables*.

Afin de préserver la vie privée des utilisateurs, les coupons se doivent d'être, tout comme la monnaie, *anonymes*, dans le sens où aucune information sur le possesseur ne

doit apparaître lors d'une dépense. De même, il ne doit pas être possible de relier entre eux deux dépenses ou un retrait et une dépense.

Cependant, contrairement à la monnaie, le problème de double dépense ne se pose pas ici. En effet, l'utilisateur dépensant toujours ses coupons auprès du même fournisseur de services, celui-ci peut vérifier directement si le coupon qui est en train d'être dépensé a déjà été utilisé ou non. De ce fait, il n'y a pas non plus de procédure de non-diffamation.

Il existe en revanche, une autre propriété de sécurité pour les coupons que nous n'avons pas considérée dans le contexte de la monnaie[3]. Il s'agit de la propriété de transférabilité de coupons (comparable au rendu de monnaie). Les coupons étant des systèmes plus simples que la monnaie, la question de transférer des coupons d'un carnet a un autre a été soulevée dès la première construction de Chen *et al.* . Deux visions différentes des coupons sont alors apparues, selon que les systèmes autorisent ou non le transfert de coupons. Chen *et al.* définissent une notion de *all or not sharing* afin d'empêcher l'utilisateur de céder une partie de son carnet, se plaçant du côté du marchand qui souhaite empêcher la revente de ses coupons, ou alors considérant l'utilisation des coupons pour les ordonnances médicales. Ils définissent ainsi une notion de *non séparabilité faible*, dans le sens où rien ne peut empêcher un utilisateur de céder son carnet en entier. Pour notre part, nous considérons dans ce mémoire qu'un multi-coupons peut être divisé, privilégiant ainsi l'intérêt des utilisateurs qui peuvent souhaiter faire profiter à d'autres personnes de leurs coupons. Ce transfert doit cependant se faire tout en respectant les propriétés de non falsification et d'anonymat.

10.2.2 Formalisation

Comme nous l'avons vu, un système de coupons (ou de multi-coupons) se compose de deux étapes principales.

- une phase de retrait, dans laquelle l'utilisateur $\mathcal{U}$ reçoit ses coupons d'un fournisseur de services $\mathcal{SP}$,
- une phase de dépense, dans laquelle l'utilisateur $\mathcal{U}$ dépense un ou des coupons auprès du même fournisseur de services $\mathcal{SP}$.

Nous définissons dans cette section, les algorithmes nécessaires à la construction d'un schéma de multi-coupons. Ces définitions prennent en compte le fait que l'utilisateur peut décider du nombre de coupons qu'il retire (dans une limite définie par le système), mais aussi de la valeur que ses coupons vont représenter.

Définition 94 (Système de multi-coupons)**.** *Un système de multi-coupons se construit à l'aide des procédures suivantes :*

- Setup*, phase de mise en place des paramètres du système, des paramètres publics et de construction des clés pour le fournisseur de services $\mathcal{SP}$. Les paramètres publics params comprennent entre autre un paramètre de sécurité k, l'ensemble $\mathcal{V} : \{\mathrm{V}_1, \ldots, \mathrm{V}_n\}$ des valeurs autorisées pour les coupons, ainsi qu'une valeur* $\mathrm{M}ax$*, représentant le nombre maximal de coupons que peut retirer un utilisateur. Les clés de $\mathcal{SP}$ sont données par l'algorithme suivant :*

[3]La raison étant qu'il n'existe pas à l'heure actuelle de solution efficace à ce problème.

- $\mathsf{G}_{\mathcal{SP}}$, *algorithme de génération de clés pour le fournisseur de services. Il prend en entrée le paramètre de sécurité* k *et les paramètres publics* $params$ *et renvoie une clé privée* $sk_{\mathcal{SP}}$ *et la clé publique associée* $pk_{\mathcal{SP}}$ *pour le signataire.* $(sk_{\mathcal{SP}}, pk_{\mathcal{SP}}) \leftarrow \mathcal{G}_{\mathcal{SP}}(1^k, params)$.

- Withdraw, *protocole interactif de retrait de coupons entre le fournisseur de services* $\mathcal{SP}$ *et l'utilisateur* $\mathcal{U}$. *Pour chaque* $i \in [1, n]$, *l'utilisateur choisit, le nombre* J_i *de coupons de valeurs* V_i *qu'il veut retirer et les donne en entrée. Le fournisseur de services donne en entrée ses clés* $(sk_{\mathcal{SP}}, pk_{\mathcal{SP}})$. *S'ils acceptent le protocole, l'utilisateur reçoit en sortie son multi-coupons* MC, *c'est-à-dire un identifiant* I *et un ensemble* $\mathcal{S} = \{(\mathrm{J}_i, \mathrm{V}_i), i \in [1, n]\}$ *et la banque reçoit une vue* $\mathcal{V}_{\mathcal{SP}}^{\mathsf{Withdraw}}$ *de l'exécution, sinon ils reçoivent un message d'erreur.*
  $(\mathcal{U}(\mathrm{I}, \mathrm{S}), \mathcal{B}(\mathcal{V}_{\mathcal{SP}}^{\mathsf{Withdraw}})) \leftarrow \mathsf{Withdraw}(\mathcal{U}(\mathrm{J}_1, \ldots, \mathrm{J}_n), \mathcal{SP}(pk_{\mathcal{SP}}, sk_{\mathcal{SP}})$

- Redeem, *protocole de dépense interactif entre l'utilisateur et le fournisseur de services. L'utilisateur fournit en entrée un multi-coupons. Il choisit aussi la valeur* V_j *du coupon qu'il souhaite dépenser. Le fournisseur de services donne en entrée ses clés* $sk_{\mathcal{SP}}, pk_{\mathcal{SP}}$. *Si le coupon est accepté, l'utilisateur reçoit un multi-coupons mis à jour, c'est-à-dire le même identifiant* I *et l'ensemble* $\mathcal{S}' = \{(\mathrm{J}'_i, \mathrm{V}_i), i \in [1, n]\}$ *où* $\mathrm{J}'_j = \mathrm{J}_j - 1$ *et* $\mathrm{J}'_i = \mathrm{J}_i$, $i \in [1, n]$ *et* $i \neq j$. *Le fournisseur de service reçoit un identifiant de coupon* S *et une preuve de validité* π. *Sinon, l'algorithme renvoie un message d'erreur.*
  $(\mathcal{U}(\mathrm{I}, \mathrm{S}'), \mathcal{SP}(\mathrm{S}, \pi)) \leftarrow \mathsf{Redeem}(\mathcal{U}(\mathrm{I}, \mathrm{S}, \mathrm{V}_j), \mathcal{SP}(sk_{\mathcal{SP}}, pk_{\mathcal{SP}}))$

- Transfer, *protocole interactif entre un utilisateur* $\mathcal{U}_1$ *qui donne en entrée un multi-coupons, c'est-à-dire un identifiant* I *et un ensemble* $\mathcal{S} = \{(\mathrm{J}_i, \mathrm{V}_i); i \in [1, n]\}$ *et un utilisateur* $\mathcal{U}_2$. *Pour chaque* $i \in [1, n]$, *l'utilisateur* $\mathcal{U}_1$ *choisit le nombre* J'_1 *avec* $\mathrm{J}'_1 \leq \mathrm{J}_i$ *de coupons de valeur* V_i *qu'il souhaite donner à l'utilisateur* $\mathcal{U}_2$. *A la fin du protocole l'utilisateur* $\mathcal{U}_2$ *est en possession d'un nouveau multi-coupons, c'est-à-dire un identifiant* I′ *et l'ensemble* $\{(\mathrm{J}'_i, \mathrm{V}_i); i \in [1, n]\}$ *ou reçoit un message d'erreur et l'utilisateur* $\mathcal{U}_1$ *a un multi-coupons mis à jour, c'est-à-dire l'identifiant* I *et l'ensemble* $\{(\mathrm{J}_i - \mathrm{J}'_i, \mathrm{V}_i); i \in [1, n]\}$ *ou un message d'erreur.*

On spécifie alors un schéma de multi-coupons comme l'ensemble :

$$\mathrm{MC} = (\mathsf{Setup}, \mathsf{Withdraw}, \mathsf{Redeem}, \mathsf{Transfer}).$$

10.2.3 État de l'art

Le premier schéma de multi-coupons a été proposé par Chen, Enzmann, Sadeghi, Schneider et Steiner en 2005 dans [CES+05]. Ce système permet à l'utilisateur de retirer un multi-coupons, c'est-à-dire un ensemble m de coupons, où la valeur m est fixée par les paramètres du système. En outre, tous les coupons ont la même valeur dans le carnet, ce qui implique que les utilisateurs ne peuvent retirer qu'un seul type de multi-coupons. En revanche, ce schéma ne nécessite pas l'implication d'un tiers de confiance et satisfait les propriétés de sécurité souhaitées. Malheureusement, la solution de dépense proposée est inefficace, notamment parce qu'elle est basée sur des preuves du « OU » et est proportionnelle au nombre de coupons retirés.

Plus récemment, Nguyen a relevé dans son article [Ngu06], certaines faiblesses de [CES+05] et a proposé un nouveau modèle de sécurité pour les multi-coupons. La solution présentée est efficace et basée sur l'utilisation de *fonctions pseudo-aléatoire vérifiables*. Le retrait et la dépense sont de complexité constante et le schéma vérifie les même propriétés de sécurité que [CES+05]. En revanche, cette fois encore, le nombre de coupons qu'il est possible de retirer est fixé par le système et les coupons ont tous la même valeur.

En 2007, Chen, Escalante, Löhr, Manulis et Sadeghi ont proposé une nouvelle construction de multi-coupons proposant une protection forte contre la séparabilité. Ils s'attachent aussi à définir de manière plus formelle les définitions de sécurité.

Chapitre 11

Nouvelles propriétés pour les coupons

Nous avons présenté au chapitre précédent, l'état de l'art dans le domaine des coupons. Cependant, dans toutes les solutions proposées, les schémas n'autorisent l'utilisateur qu'à retirer un nombre prédéterminé de coupons. De plus, ceux-ci ont tous la même valeur. Au final, les utilisateurs ne peuvent donc obtenir que le même type de multi-coupons. Afin de pallier ces défauts, nous décrivons dans ce chapitre un nouveau système de multi-coupons conforme à la définition 94 (*cf.* section 10.2). Par conséquent, nous donnons à l'utilisateur la possibilité de choisir le nombre et la valeur de ses coupons, lorsqu'il les retire. En nous inspirant des travaux de Camenisch *et al.* [CHL05], nous avons proposé à la conférence ACNS'06, dans un travail commun avec Sébastien Canard et Aline Gouget, un nouveau schéma de multi-coupons possédant ces avantages. En outre, nous avons redéfini les propriétés de sécurité de [CES+05], afin de renforcer leur modèle de sécurité.

Sommaire

11.1 Motivation

Le schéma que nous proposons possède deux nouvelles caractéristiques tout à fait intéressantes dans le contexte des coupons. Tout d'abord, le nombre de coupons contenu dans un multi-coupons n'est pas fixé par le système. L'utilisateur peut donc choisir le nombre de coupons qu'il souhaite retirer, en respectant une valeur limite donnée par le fournisseur de services.

Ensuite, la valeur des coupons n'est pas, elle non plus, déterminée par le système. L'utilisateur a à sa disposition un ensemble de valeurs possibles pour les coupons qu'il va retirer. Pour chacune de ces valeurs, il choisit le nombre de coupons qu'il souhaite réellement retirer.

Notre construction trouve une utilisation directe dans des applications comme les carnets de tickets ou les bons de réductions. Prenons, par exemple, le cas de chèques cadeaux. Un utilisateur qui souhaite en offrir peut alors décider du nombre de chèques qu'il achète, ainsi que des différents montants de ceux-ci. Afin de les distribuer aux bénéficiaires de son choix, il va devoir leur transférer ces chèques. C'est pourquoi nous nous plaçons ici dans un contexte où la transférabilité des coupons est autorisée et même souhaitée. Ceci nous démarque de [CBL+07] et nous oriente vers des définitions de sécurité différentes.

Pour réaliser notre construction, nous utilisons certaines des techniques proposées par Camenisch *et al.* pour la monnaie électronique dans [CHL05]. Comme nous l'avons présenté au chapitre 10, ce schéma combine les signatures de type Camenisch-Lysyanskaya [CL02], la fonction aléatoire vérifiable de Dodis-Yampolskiy [DY05] et un système de numéro de série. Cependant, dans [CHL05], les pièces de monnaie (que l'on peut considérer comme des coupons) ont toutes la même valeur et le nombre de pièces qu'un utilisateur peut retirer est fixé par le système. Le schéma de multi-coupons de Chen *et al.* présente le même défaut.

Notre travail a donc été de proposer un modèle de sécurité adapté aux multi-coupons, c'est-à-dire comprenant les propriétés d'inforgeabilité et d'anonymat mais permettant aussi aux utilisateurs de transférer leurs coupons, tout en préservant ces propriétés. Notre schéma est plus efficace que le schéma de Chen *et al.* tout en offrant les nouvelles caractéristiques énoncées ci-dessus. Le protocole de dépense est basé sur une preuve du OU dont la complexité est proportionnelle au logarithme du nombre maximum de coupons possible, ce qui est plus efficace que la solution de Chen *et al.* .

11.2 Nouveau modèle de sécurité

Le modèle définit par Chen *et al.* est un modèle assez complet. Cependant, certaines définitions sont décrites trop informellement pour caractériser exactement la sécurité des schémas. De plus, certaines de leurs propriétés peuvent être réunies en une seule, ce qui simplifie la mise en place des preuves de sécurité. Nous commençons tout d'abord par définir notre modèle et nous montrons dans la suite en quoi il est plus satisfaisant que celui de Chen *et al.*

Notre modèle de sécurité prouve la sécurité d'un schéma décrit par les algorithmes donnés à la définition 94.

11.2.1 Oracles

Comme nous l'avons fait précédemment pour définir le modèle de sécurité des signatures aveugles à anonymat révocable, nous commençons par décrire les différents oracles auxquels l'adversaire peut être amené à faire appel durant ses attaques.

Nous utilisons, dans la suite de ce chapitre, les notations suivantes :
- HU représente l'ensemble des utilisateurs honnêtes.
- CU représente l'ensemble des utilisateurs corrompus par l'adversaire.

Un protocole de multi-coupons est composé de deux ensembles d'entités, les utilisateurs et les fournisseurs de services. L'adversaire peut vouloir, au cours de ses attaques, utiliser l'un ou l'autre de ces acteurs.

Pour cela, il a tout d'abord, accès à deux oracles lui permettant de jouer avec les utilisateurs.

AddU(.) Cet oracle permet à l'adversaire d'inscrire de nouveaux utilisateurs. L'adversaire fournit à l'oracle le nom d'un utilisateur. Lorsque ce dernier demande un multi-coupons ou dépense un coupon, l'adversaire n'a accès qu'aux transcripts des protocoles effectués par cet utilisateur. En particulier, il n'a pas connaissance des multi-coupons retirés. L'utilisateur est ajouté à la liste HU.

CrptU(.) Grâce à cet oracle, l'adversaire est capable de corrompre des utilisateurs. L'adversaire fournit à l'oracle le nom de l'utilisateur visé. Il reçoit alors une liste de tous les multi-coupons retirés par cet utilisateur lors des sessions passées, en particulier, il a connaissance des coupons déjà dépensés et est capable de dépenser ceux restants.

L'adversaire peut aussi vouloir simuler des exécutions de protocoles de retrait ou de dépense, tout en se mettant dans le rôle d'un utilisateur ou d'un marchand. Pour cela, il a à sa disposition deux oracles.

WithdrawU(.,.) Cet oracle sert à simuler des protocoles de retrait entre un utilisateur honnête et un fournisseur de services corrompu (joué par l'adversaire). L'oracle joue ici un utilisateur honnête. L'adversaire fournit en entrée le nom de l'utilisateur choisi ainsi que les informations nécessaires à la transaction pour le fournisseur de services. L'oracle renvoie à l'adversaire une vue du protocole $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}}$.

WithdrawSP(.) Cet oracle sert à simuler des protocoles de retrait entre un fournisseur de services honnête et un utilisateur corrompu (joué par l'adversaire). L'oracle joue ici le rôle d'un fournisseur de services honnête. L'adversaire fournit en entrée les entrées du protocole de Withdraw pour un utilisateur. Si le protocole est accepté par les deux parties, il reçoit en retour la sortie normale du protocole de Withdraw.

RedeemU(.) Cet oracle sert à simuler une dépense entre un utilisateur honnête et un fournisseur de services corrompu (joué par l'adversaire). L'adversaire fournit en entrée

le nom de l'utilisateur choisi. Si l'utilisateur accepte le protocole, l'adversaire reçoit une vue $\mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}}$ de l'exécution du protocole et le coupon dépensé.

$\mathsf{RedeemSP}(.)$ Cet oracle sert à simuler une exécution du protocole Redeem entre un fournisseur de services honnête et un utilisateur (honnête ou corrompu et joué par l'adversaire). L'adversaire fournit en entrée le nom d'un utilisateur. Il exécute ensuite le protocole avec l'oracle au nom de cet utilisateur. Si le fournisseur de services et l'utilisateur acceptent tous deux le protocole, l'adversaire reçoit le multi-coupons mis à jour de l'utilisateur ainsi qu'une vue du protocole.

$\mathsf{Transfer}(.,.)$ Cet oracle prend, en première entrée, soit l'utilisateur $\mathcal{U}_1$, c'est-à-dire la personne qui donne certains de ses coupons, soit l'utilisateur $\mathcal{U}_2$, c'est-à-dire celle qui reçoit des coupons.

Selon la donnée mise en première entrée, l'oracle va simuler une exécution d'un protocole de transfert soit en jouant le rôle d'un utilisateur $\mathcal{U}_2$ honnête, soit en simulant un joueur $\mathcal{U}_1$ honnête. L'adversaire donne en deuxième entrée les informations nécessaires à l'exécution d'un protocole de transfert selon le participant qu'il fait jouer.

Le dernier oracle est un oracle de challenge, utilisé pour l'anonymat du schéma. Cet oracle permet à l'adversaire de recevoir un challenge auquel il doit répondre. Le bit b est choisi avant l'appel à l'oracle et est inconnu de l'adversaire.

$\mathsf{Choose}_b(.,.,.)$ L'adversaire donne à l'oracle deux utilisateurs $\mathcal{U}_0$ et $\mathcal{U}_1$. L'oracle vérifie que les utilisateurs appartiennent à HU et $\mathcal{A}$ exécute deux protocoles de retrait avec ces utilisateurs. Il conserve les vues $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_0}$ et $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_1}$. Le bit b est choisi aléatoirement sans le concours de l'adversaire. Ensuite, un protocole de dépense Redeem ou un protocole de transfert $\mathsf{Transfer}$ est joué par le propriétaire du multi-coupons issu de l'échange $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_b}$ ainsi que celui issu de l'échange $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_{\bar{b}}}$. $\mathcal{A}$ reçoit les vues $\mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_b}$ ou $\mathcal{V}_{\mathcal{A}}^{\mathsf{Transfer}_b}$ et $\mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_{\bar{b}}}$ ou $\mathcal{V}_{\mathcal{A}}^{\mathsf{Transfer}_{\bar{b}}}$.

11.2.2 Consistance

Cette propriété garantie que toute personne qui utilise correctement le protocole ne recevra pas de message d'erreur. Plus précisément, si un utilisateur honnête $\mathcal{U}$ exécute un protocole de retrait $\mathsf{Withdraw}$ avec un fournisseur de services $\mathcal{SP}$ honnête, alors le protocole ne retournera pas de message d'erreur. De même, si un utilisateur honnête $\mathcal{U}$ exécute un protocole de dépense Redeem avec un fournisseur de services $\mathcal{SP}$ honnête, alors $\mathcal{SP}$ acceptera toujours le coupon.

11.2.3 Anonymat

L'anonymat d'un schéma de multi-coupons est assez proche de l'indistinguabilité que nous avons définie pour les schémas de signature aveugle. Elle garantit à l'utilisateur que ses dépenses ne pourront ni être reliées entre elles, ni à une de ses sessions de retrait. De plus, comme nous autorisons le transfert, l'utilisateur doit aussi avoir la garantie que

l'adversaire ne sera pas capable de relier un transfert à un retrait, un transfert à une dépense ou deux transferts entre eux.

Le fournisseur de services ne doit donc apprendre aucune information compromettant l'identité de l'utilisateur lors des dépenses.

Pour le schéma de multi-coupons noté MC, tout adversaire $\mathcal{A}$, un bit $b \in \{0,1\}$ et tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{MC},\mathcal{A}}(k)$ (*cf.* figure 11.1). L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{blind}}_{\text{MC},\mathcal{A}}(k) = 2|\Pr[\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{MC},\mathcal{A}}(k) = b] - 1/2|.$$

On dit alors que le schéma est *anonyme* si la fonction $\mathbf{Adv}^{\text{blind}}_{\text{MC},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

Pour exécuter son attaque, l'adversaire reçoit la clé privée du fournisseur de services $sk_{\mathcal{SP}}$. Il peut ensuite faire autant d'appels qu'il le souhaite aux oracles $\mathsf{WithdrawU}$, $\mathsf{RedeemU}$ et $\mathsf{Transfer}$. Il peut aussi corrompre tous les utilisateurs de son choix et recevoir ainsi les données contenues dans leurs multi-coupons. Il peut aussi, à tout moment de son attaque, faire appel à son oracle Choose_b.

À la fin de son attaque, l'adversaire renvoie ensuite un bit b', qui est sa supposition sur le bit choisi par le challenger.

Il est requis à ce moment là que les multi-coupons obtenus lors de chaque appel à l'oracle Choose_b aient encore un coupon non dépensé. Dans le cas contraire, si le multi-coupons de $\mathcal{U}_0$ ou $\mathcal{U}_1$ est vide, alors $\mathcal{A}$ peut aisément deviner le bit b en remarquant qu'il ne peut faire dépenser ou transférer à $\mathcal{U}_b$ tous les coupons qu'il lui a fait retirer, étant donné que l'oracle Choose_b a effectué une dépense ou un transfert.

Expérience $\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{MC},\mathcal{A}}(k)$

$(pk_{\mathcal{S}}, sk_{\mathcal{S}}) \leftarrow \mathsf{G}_{\mathcal{S}}(1^k)$
$b' \leftarrow \mathcal{A}(pk_{\mathcal{S}}, sk_{\mathcal{S}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{WithdrawU}, \mathsf{RedeemU}, \mathsf{Transfer}, \mathsf{Choose}_b)$
retourne b'

FIG. 11.1 – Anonymat des multi-coupons

De manière semblable aux signatures aveugles à anonymat révocable, il est possible de restreindre l'expérience au cas où l'adversaire ne fait qu'une requête à l'oracle Choose_b.

11.2.4 Inforgeabilité

L'inforgeabilité d'un schéma de multi-coupons garantit qu'aucune coalition d'utilisateurs n'est capable de dépenser ou transférer plus de coupons qu'elle n'en a retirés.

Pour le schéma de multi-coupons noté MC, tout adversaire $\mathcal{A}$ et tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{Unforg}}_{\text{MC},\mathcal{A}}(k)$. L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{Unforg}}_{\text{MC},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{Unforg}}_{\text{MC},\mathcal{A}}(k) = 1].$$

On dit alors que le schéma est *inforgeable* si la fonction $\mathbf{Adv}^{\text{Unforg}}_{\text{MC},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

Pour mener à bien son attaque, l'adversaire reçoit la clé publie $pk_{\mathcal{SP}}$ du fournisseur de services. Il peut ensuite ajouter au système autant d'utilisateurs qu'il souhaite et les corrompre. Il peut aussi faire appel aux oracles WithdrawSP, RedeemSP et Transfer à sa guise. A chaque exécution d'un retrait et à chaque corruption d'utilisateur, un compteur de coupons $\mathrm{N_C}$ est incrémenté du nombre de coupons retirés lors du WithdrawSP ou obtenus lors du CrptU (le système connaît le nombre de coupons retirés, car, en faisant appel à l'oracle WithdrawSP, l'adversaire doit spécifier le nombre de coupons qu'il souhaite retirer). Le but de l'attaquant est alors de dépenser $\mathrm{N_C} + 1$ coupons valides, c'est-à-dire acceptés par le fournisseur de services.

L'adversaire renvoie, à la fin de son attaque, une liste

$$((\mathrm{I}_1, \mathrm{V}_{j_1}), \ldots (\mathrm{I}_{\mathrm{N_C}+1}, \mathrm{V}_{j_{\mathrm{N_C}+1}})), j \in \{1, n\}$$

où I_i est l'identifiant du coupon et V_{j_i} sa valeur.

Si tous les coupons sont acceptés par le fournisseur de services et sont tous distincts, alors $\mathcal{A}$ a réussi son attaque.

11.2.5 Comparaison avec le modèle de Chen *et al.*

Nous montrons ici que notre modèle comprend et améliore toutes les définitions essentielles données par Chen *et al.* dans [CES+05].

Inforgeabilité Chen *et al.* définissent l'inforgeabilité comme étant l'impossibilité de créer de nouveaux multi-coupons, d'augmenter le nombre de coupons non dépensés ou de remettre à zéro le nombre de coupons dépensés. De plus, Chen *et al.* définissent une propriété qu'ils appellent *limitation de dépense* qui consiste à limiter à m le nombre de fois qu'un fournisseur de services peut accepter un multi-coupons de taille m. Cette propriété signifie que l'utilisateur n'est pas capable d'augmenter le nombre de coupons contenus dans son multi-coupons, autrement dit, l'utilisateur n'est pas capable de créer de nouveau coupon dans son multi-coupons. Notre définition comprend cette notion de limitation de dépense en plus de celle d'inforgeabilité.

Détection des doubles dépenses La propriété de *détection des doubles dépenses* est définie dans le modèle de Chen *et al.* . Cependant, comme nous l'avons expliqué au chapitre 10, cette notion est inutile dans le contexte des coupons. En effet, avant d'accepter un coupon, le fournisseur de services peut directement vérifier si celui-ci a déjà été dépensé ou non. De ce fait, une double dépense est impossible. C'est pourquoi nous ne considérons pas cette propriété dans notre modèle.

Anonymat et divulgation minimale Notre définition de l'anonymat est semblable à celle de Chen *et al.* Elle considère le fait qu'il ne doit pas être possible de relier entre

eux un retrait et une dépense, deux retraits, une dépense et un transfert, un retrait et un transfert ou deux transferts.

La propriété de divulgation minimale définie par Chen *et al.* garantit que le nombre de coupons non dépensés ne peut pas être dévoilé pendant une dépense. Chen *et al.* séparent les propriétés d'anonymat et de divulgation minimale. La propriété de divulgation minimale est en fait, incluse dans notre définition de l'anonymat, c'est pourquoi nous ne séparons pas ces deux notions.

Transfert de coupons/protection contre la séparation La principale différence entre notre système et celui de Chen *et al.* réside dans la transférabilité.

Il est, bien évidemment, impossible d'empêcher quiconque de céder son multi-coupons à un autre utilisateur. Chen *et al.* définissent alors la propriété de *all or nothing sharing* qui consiste à se prémunir contre des utilisateurs cédant une partie de leur multi-coupons sans dévoiler le multi-coupons dans son intégralité. Cette propriété est définie de la manière suivante : aucune coalition d'utilisateurs $\mathcal{U}_i$ ne doit pouvoir diviser un multi-coupons M de taille m en multi-coupons M_i de taille s_i avec $\Sigma_i s_i \leq m$ de telle sorte que M_i ne puisse être utilisable que par l'utilisateur $\mathcal{U}_i$ et non par les autres utilisateurs $\mathcal{U}_j, j \neq i$, ou un sous-ensemble de ceux-ci.

Chen *et al.* définissent aussi la propriété de *non séparation faible*. L'utilisateur $\mathcal{U}_1$ cède une partie de son multi-coupons à l'utilisateur $\mathcal{U}_2$ et les deux utilisateurs se font confiance pour ne pas dépenser la partie du multi-coupons qu'ils n'ont pas.

Nous avons, pour notre part, adopté une approche différente pour notre construction. En effet, nous autorisons les utilisateurs à partager leurs multi-coupons en ajoutant un nouvel algorithme Transfer. Un utilisateur $\mathcal{U}_1$ possédant un multi-coupons $\mathrm{C} = \{\mathrm{C}_0, \ldots, \mathrm{C}_m\}$ peut transférer à l'utilisateur $\mathcal{U}_2$ une partie de C. A la fin du protocole, $\mathcal{U}_1$ obtient un multi-coupons C' et $\mathcal{U}_2$ le multi-coupons $\hat{\mathrm{C}}$ tels que $\mathrm{C}' \cup \hat{\mathrm{C}} = \mathrm{C}$ et $\mathrm{C}' \cap \hat{\mathrm{C}} = \emptyset$.

11.3 Description du schéma

11.3.1 Schéma

Nous décrivons dans cette section le schéma que nous avons présenté dans [CGH06]. Ce schéma est défini à l'aide des algorithmes donnés dans la définition 94 (*cf.* section 10.2).

Soit $\mathcal{U}$ un utilisateur souhaitant retirer un multi-coupons auprès d'un marchand ou un fournisseur de services $\mathcal{SP}$. Il décide tout d'abord du nombre m de coupons qu'il souhaite obtenir et choisit la valeur de ceux-ci dans la liste $\mathcal{V} = \{\mathrm{V}_1, \ldots, \mathrm{V}_n\}$ donnée par le fournisseur de services. Pour chaque valeur V_i, l'utilisateur choisit le nombre J_i de coupons qu'il veut retirer. En raison de la preuve de connaissance que nous utilisons dans notre construction, le nombre total de coupons doit être limité par une valeur 2^l fixée par le fournisseur de services. Les valeurs $\mathrm{J}_1, \ldots, \mathrm{J}_n$ sont choisies par l'utilisateur[1] et signées par le fournisseur de services pendant la phase de retrait. Elles ne sont, en

[1] Elles peuvent également être choisies par le fournisseur de services si l'application le requiert.

revanche, pas révélées pendant la phase de dépense, afin de conserver l'anonymat de l'utilisateur. Chaque valeur V_i est associée à une valeur aléatoire $\tilde{g}_i$ dans un groupe cyclique $\mathbb{G}$. Durant la phase de dépense d'un coupon de valeur V_i, l'utilisateur choisit un entier dans l'ensemble $\mathcal{J}_i = \{0, \ldots, \mathrm{J}_i - 1\}$. Pour chaque dépense d'un coupon de valeur V_i, l'utilisateur doit choisir un entier différent de ceux révélés durant les sessions de dépense qu'il a effectuées auparavant, pour des coupons de valeur V_i.

Nos procédures de retrait et de dépense utilisent les mêmes outils que celles de Camenisch *et al.* , c'est-à-dire des signatures Camenisch-Lysyanskaya sur des engagements et la fonction pseudo-aléatoire de Dodis-Yampolskyi. Afin de pouvoir détailler plus précisément le schéma, nous utilisons ici la signature Camenisch-Lysysankaya basée sur BBS que nous avons décrite en section 3.4.2.

Le multi-coupons est donc constitué d'une signature du fournisseur de service sur une graine qui servira à évaluer l'identifiant de chaque coupon au moment de la dépense. Cette signature fait partie de l'identifiant du multi-coupons. A cet identifiant, nous ajoutons l'ensemble $\mathcal{S} = \{(\mathrm{J}_i, \mathrm{V}_i); i \in [1, n]\}$ donnant le nombre J_i de coupons de valeur V_i que l'utilisateur a retirés.

De même, dans la procédure de dépense, l'utilisateur prouve la connaissance de ses secrets afin que ses dépenses ne puissent être reliées entre elles et ne révèlent rien sur son identité. En revanche, cette procédure est faite interactivement avec le fournisseur de services.

Notre protocole n'ayant pas de procédure de détection des double dépenses, nous n'aurons pas besoin du tag de sécurité T décrit dans [CHL05].

Mise en place du système : Setup

Les paramètres du système sont donnés par les valeurs suivantes :

- $k \in \mathbb{N}$ est le paramètre de sécurité.
- l est un entier servant à fixer la taille maximale d'un multi-coupons $\mathrm{M}ax = 2^l$.
- $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_\mathrm{T}$ sont trois groupes cycliques de même ordre p et de générateurs g_1, g_2 et g_T respectivement.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ est une application bilinéaire admissible.
- $\tilde{g}_1, \ldots, \tilde{g}_n$ sont des éléments choisis aléatoirement dans $\mathbb{G}_\mathrm{T}$,
- $g, h, h_1, \ldots h_{n+1}$ sont des éléments choisis aléatoirement dans $\mathbb{G}_1$ tels que $\log_{h_j} h_i$ pour $i \neq j$ soient inconnus de tous, et $h_0 = g_1^{\chi}$ pour une valeur (gardée secrète) $\chi \in_\mathrm{R} \mathbb{Z}_p$,
- $\{\mathrm{V}_1, \ldots, \mathrm{V}_n\}$ sont les différentes valeurs possibles pour un coupon.

Ces éléments composent les paramètres publics *params* du système.

$\mathsf{G}_\mathcal{S}$ **:** le fournisseur de services $\mathcal{SP}$ calcule sa paire de clés ($sk_{\mathcal{SP}}$, $pk_{\mathcal{SP}}$)pour une signature BBS :

- il choisit $\gamma \in \mathbb{Z}_p$ et pose $sk_{\mathcal{SP}} = \gamma$,
- il calcule ensuite $pk_{\mathcal{SP}} = g_2^{\gamma}$.

Retrait d'un multi-coupons : Withdraw

Durant un protocole de retrait, l'utilisateur $\mathcal{U}$ prend en entrée les paramètres publics

params et la clé publique $pk_{\mathcal{SP}}$ et interagit avec le fournisseur de services $\mathcal{SP}$ qui prend en entrée les paramètres *params* et sa clé privée $sk_{\mathcal{SP}}$.

1. Tout d'abord, $\mathcal{U}$ choisit sa participation à un secret partagé $s' \in \mathbb{Z}_p$, envoie à $\mathcal{SP}$ un engagement C sur la valeur s' et les valeurs $\mathrm{J}_1, \ldots, \mathrm{J}_n$ correspondant au nombre de coupons de valeurs $\mathrm{V}_1, \ldots, \mathrm{V}_n$ qu'il souhaite retirer.
2. $\mathcal{U}$ et $\mathcal{SP}$ exécutent un protocole de signature Camenisch-Lysyanskaya afin d'obtenir une signature sur les valeurs engagées s, $\mathrm{J}_1, \ldots, \mathrm{J}_n$. $\mathcal{U}$ obtient au final $\sigma = \mathsf{CLSign}^{\mathsf{BBS}}_{sk_{\mathcal{SP}}}(s, \mathrm{J}_1, \ldots, \mathrm{J}_n, r) = (\mathrm{A}, x)$ où $x \in \mathbb{Z}_p$ et $\mathrm{A} = (g h_0^s \Pi_{i=1}^n h_i^{\mathrm{J}_i})^{\frac{1}{x+\gamma}}$.
3. $\mathcal{U}$ sauvegarde son multi-coupons, c'est-à-dire l'identifiant $\mathrm{I} = (s, r, \sigma)$ et l'ensemble $\mathcal{S} = \{(\mathrm{J}_i, \mathrm{V}_i); i \in [1, n]\}$.

Le protocole de retrait est décrit plus précisément dans la figure 11.2.

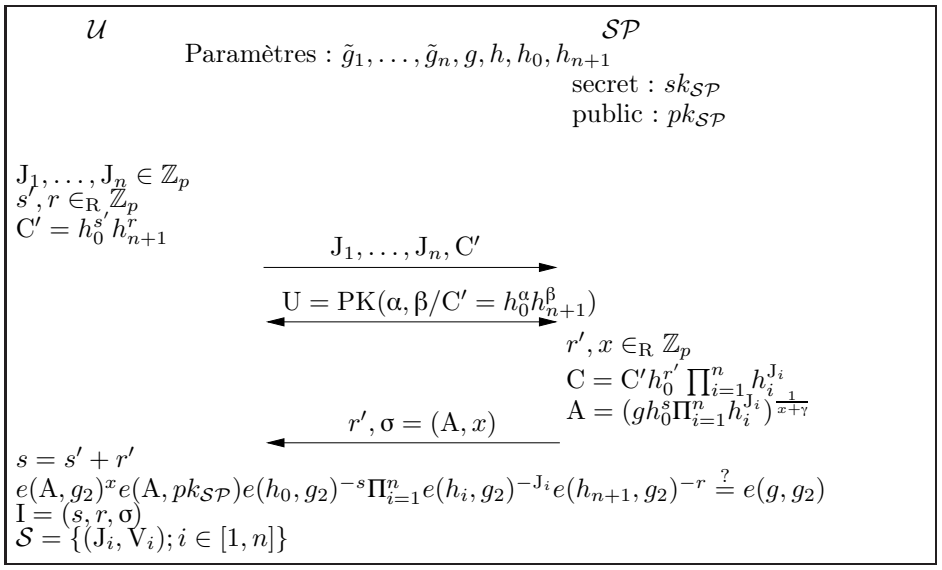

FIG. 11.2 – Protocole de Retrait

Dépense d'un coupon : Redeem

Lorsqu'un utilisateur souhaite dépenser un coupon de son multi-coupons $(\mathrm{I}, \mathcal{S})$, il choisit tout d'abord la valeur V_i du coupon qu'il souhaite dépenser. Ensuite, il choisit le rang j du coupon parmi l'ensemble de tous les coupons de valeur V_i, c'est-à-dire une valeur entre 0 et $\mathrm{J}_i - 1$.

Comme nous le détaillons dans la figure 11.3, un protocole de dépense se déroule de la manière suivante :

1. $\mathcal{U}$ calcule l'identifiant du coupon qu'il dépense à l'aide de la fonction pseudo-aléatoire de Dodis-Yampolskiy, en prenant comme graine s et comme générateur l'élément $\tilde{g}_i$ associé à la valeur monétaire V_i. L'identifiant S_i du coupon est alors : $\mathrm{S}_i = \tilde{g}_i^{\frac{1}{s+j+1}}$.

2. $\mathcal{U}$ interagit ensuite avec $\mathcal{SP}$, afin de lui prouver que son multi-coupons a bien été signé, c'est-à-dire sa connaissance d'une signature sur les secrets $(s, \mathrm{J}_1, \ldots, \mathrm{J}_n, r)$ et que le coupon qu'il dépense est bien compris dans l'intervalle $\mathcal{J}_i = \{0, \ldots, \mathrm{J}_i - 1\}$, tout en ne révélant pas la valeur J_i. Cette preuve de connaissance, notée Φ inclut un challenge c, envoyé par le fournisseur de services. De ce fait, la preuve est interactive.
3. Si $\mathcal{SP}$ accepte la preuve Φ, alors il accepte le coupon (S, Φ).
4. L'utilisateur est alors en possession d'un multi-coupons mis à jour, c'est-à-dire le même identifiant I et l'ensemble $\mathcal{S}' = \{(\mathrm{J}'_i, \mathrm{V}_i); i \in [1, n]\}$ où $\mathrm{J}'_j = \mathrm{J}_j - 1$ et $\mathrm{J}'_i = \mathrm{J}_i$ pour $i \neq j$.

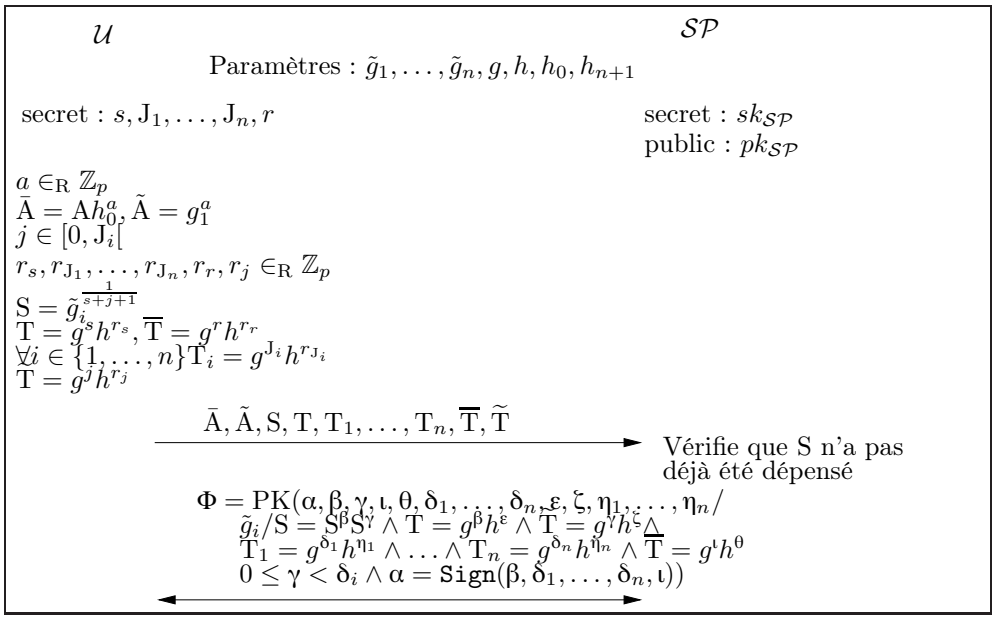

FIG. 11.3 – Protocole de Dépense

La preuve de connaissance Φ de l'étape 2 se déroule de la manière suivante :

1. $\mathcal{U}$ calcule des engagements sur ses secrets $(s, \mathrm{J}_1, \ldots, \mathrm{J}_n, r, j)$:

$$\mathrm{T} = g^s h^{r_s}, \quad \bar{\mathrm{T}} = g^r h^{r_r}, \quad \tilde{\mathrm{T}} = g^j h^{r_j} \tag{11.1}$$

$$\mathrm{T}_1 = g^{\mathrm{J}_1} h^{r_{\mathrm{J}_1}}, \quad \ldots, \quad \mathrm{T}_n = g^{\mathrm{J}_n} h^{r_{\mathrm{J}_n}}. \tag{11.2}$$

2. La preuve Φ comprend alors
   (a) une preuve de connaissance sur les valeurs engagées dans T, $\bar{\mathrm{T}}$, $\tilde{\mathrm{T}}$, T_1, ..., T_n, c'est à dire la connaissance de $\beta = s$, $\delta_1 = \mathrm{J}_1, \ldots, \mathrm{J}_n = \delta_n$ $\gamma = j$ et $\iota = r$, tels que $\mathrm{T}, \bar{\mathrm{T}}, \tilde{\mathrm{T}}, \mathrm{T}_1, \cdots, \mathrm{T}_n$ sont consistants ;
   (b) une preuve de connaissance de la signature de $\mathcal{B}$ sur les valeurs engagées $(s, \mathrm{J}_1, \ldots, \mathrm{J}_n, r)$. Cette preuve s'effectue comme nous l'avons décrite pour notre schéma de signature aveugle à anonymat révocable (*cf.* section 9.1.2) ;

(c) une preuve que la valeur j appartient bien à l'ensemble $\mathcal{J}_i = \{0, \ldots, \mathrm{J}_i - 1\}$ en utilisant la preuve qu'une valeur engagée est plus petite qu'une autre valeur engagée, décrite dans le chapitre 1 ;

(d) une preuve que S est bien construit en utilisant les valeurs s et j engagées.

Nous décrivons plus en détail dans la figure 11.4 les preuves de connaissance calculées dans Φ. Nous renvoyons le lecteur à la section 3.3.2 pour le détail de la preuve d'appartenance de j.

$\mathcal{U}$		$\mathcal{SP}$
$\lambda_s, \lambda_{r_s}, \lambda_r, \lambda_{r_r}, \lambda_j, \lambda_x, \lambda_\alpha, \lambda_z \in \mathbb{Z}_p$ $\lambda_{\mathrm{J}_i}, \lambda_{r_{\mathrm{J}_i}} \in \mathbb{Z}_p$ pour $i \in \{1, \ldots, n\}$		
$\tilde{\mathrm{A}}_\lambda = g_1^{\lambda_a}$ $\mathrm{T}_\lambda = g^{\lambda_s} h^{\lambda_{r_s}}$ $\bar{\mathrm{T}}_\lambda = g^{\lambda_r} h^{\lambda_{r_r}}$ $\tilde{\mathrm{T}}_\lambda = g^{\lambda_j} h^{\lambda_{r_j}}$ $\mathrm{S}_\lambda = \mathrm{S}^{\lambda_s} \mathrm{S}^{\lambda_j}$ $\mathrm{T}_{\lambda_i} = g^{\lambda_{\mathrm{J}_i}} h^{\lambda_{r_{\mathrm{J}_i}}}$ pour $i \in \{1, \ldots, n\}$ $\mathrm{A}_\lambda = e(\bar{\mathrm{A}}, g_2)^{\lambda_x} e(h_0, g_2)^{\lambda_s - \lambda_z} \Pi_{i=1}^n e(h_i, g_2)^{-\lambda_{\mathrm{J}_i}} e(h_{n+1}, g_2)^{-\lambda_r} e(h_0, \Gamma)^{-\lambda_a}$		
	$\xrightarrow{\mathrm{T}, \bar{\mathrm{T}}, \tilde{\mathrm{T}}, \mathrm{S}, \{\mathrm{T}_i\}, \bar{\mathrm{A}}, \tilde{\mathrm{A}}}$	
		$c \in [0, 2^k - 1]$
	$\xleftarrow{c}$	
$s_s = \lambda_s + cs \mod p$ $s_{r_s} = \lambda_{r_s} + cr_s \mod p$ $s_r = \lambda_r + cr \mod p$ $s_{r_r} = \lambda_{r_r} + c_{r_r} \mod p$ $s_j = \lambda_j + cj \mod p$ $s_{r_j} = \lambda_{r_j} + cr_j \mod p$ $s_x = \lambda_x + cx \mod p$ $s_a = \lambda_a + ca \mod p$ $s_z = \lambda_z + cz \mod p$ $s_{\mathrm{J}_i} = \lambda_{\mathrm{J}_i} + c\mathrm{J}_i \mod p$ pour $i \in \{1, \ldots, n\}$ $s_{r_{\mathrm{J}_i}} = \lambda_{r_{j_i}} + cr_{\mathrm{J}_i} \mod p$ pour $i \in \{1, \ldots, n\}$		
	$\xrightarrow[\mathrm{T}_\lambda, \bar{\mathrm{T}}_\lambda, \tilde{\mathrm{T}}_\lambda, \mathrm{S}_\lambda, \{\mathrm{T}_{\lambda_i}\}, \mathrm{A}_\lambda, \tilde{\mathrm{A}}_\lambda}^{s_s, s_{r_s}, s_r, s_{r_r}, s_j, s_{r_j}, s_x, s_a, s_z, s_{\mathrm{J}_i}, s_{r_{\mathrm{J}_i}}}$	vérifie que :
		$\tilde{\mathrm{A}}_\lambda = g_1^{s_a} \tilde{\mathrm{A}}^{-c}$ $\mathrm{T}_\lambda = g^{s_s} h^{s_{r_s}} \mathrm{T}^{-c}$ $\bar{\mathrm{T}}_\lambda = g^{s_r} h^{s_{r_r}} \bar{\mathrm{T}}^{-c}$ $\tilde{\mathrm{T}}_\lambda = g^{s_j} h^{s_{r_j}} \tilde{\mathrm{T}}^{-c}$ $\mathrm{S}_\lambda = \mathrm{S}^{s_s} \mathrm{S}^{s_j} (\tilde{g}_i / \mathrm{S})^{-c}$ $\mathrm{T}_{\lambda_i} = g^{s_{\mathrm{J}_i}} h^{s_{r_{\mathrm{J}_i}}} \mathrm{T}_i^{-c}$
$e(\bar{\mathrm{A}}, g_2)^{s_x} e(h_0, \Gamma)^{-s_a} e(h_0, g_2)^{s_s - s_z} \Pi_{i=1}^n e(h_i, g_2)^{-s_{\mathrm{J}_i}} e(h_{n+1}, g_2)^{-s_r} = \mathrm{A}_\lambda \left(\frac{e(h_0, g_2)}{e(\bar{\mathrm{A}}, \Gamma)}\right)^c$		

FIG. 11.4 – Preuves de connaissance de dépense d'un coupon

Transfert : Transfer

Nous décrivons ici, comment, de manière triviale, un utilisateur $\mathcal{U}_1$ transfère certains de ses coupons à un autre utilisateur $\mathcal{U}_2$.

Notre solution nécessite la participation du fournisseur de services $\mathcal{SP}$ qui est chargé de récupérer et de redistribuer les coupons. L'utilisateur $\mathcal{U}_1$ est en possession d'un multi-coupons (I, S). Afin de transférer certains de ses coupons, il procède de la façon suivante :

1. Il choisit tout d'abord, pour chaque valeur V_j, le nombre $\mathrm{J}'_i \leq \mathrm{J}_i$ de coupons qu'il souhaite transférer.
2. Pour chaque coupon, il s'engage dans une procédure de dépense auprès du fournisseur de services.
3. Une fois toutes les dépenses faites, son multi-coupons est de la forme $(\mathrm{I}, \mathcal{S}')$ où I est le même et $\mathcal{S}' = \{(\mathrm{J}''_i, \mathrm{V}_i); i \in [1, n]\}$ où $\mathrm{J}''_i = \mathrm{J}_i - \mathrm{J}'_i$.

L'utilisateur $\mathcal{U}_2$ va ensuite retirer tous les coupons dépensés auprès du fournisseur de services. Son multi-coupons est alors $(\tilde{\mathrm{I}}, \tilde{\mathcal{S}})$, où nécessairement $\tilde{\mathrm{I}} \neq \mathrm{I}$ et $\tilde{\mathcal{S}} = \{\mathrm{J}'_j, \mathrm{V}_i)\}$.

Il n'existe pas, à notre connaissance, de manière plus efficace de transférer des coupons.

11.3.2 Autres protocoles

Nous avons détaillé dans la section précédente, les algorithmes principaux nécessaires à la construction et à la sécurité d'un système de multi-coupons. Dans le but de faciliter l'utilisation des coupons, nous décrivons ici deux nouveaux algorithmes permettant d'ajouter de nouvelles propriétés au schéma.

MulitRedeem

La procédure de multi-dépense, consiste juste à rendre possible la dépense de plusieurs coupons en une seule interaction avec le fournisseur de services. La protocole global doit alors être plus efficace que l'exécution consécutive de plusieurs protocoles de dépense, tel que décrit à la figure 11.3. Il s'avère que dans notre cas, la preuve de connaissance de la signature σ délivrée par le fournisseur de services, n'a besoin d'être faite qu'une seule fois. En revanche, la preuve d'appartenance à un intervalle du rang de chaque coupon doit être faite pour chaque coupon. Le protocole final est décrit plus en détail à la figure 11.5

Revoc

La révocation d'un multi-coupons n'est pas une propriété considérée dans [CES+05]. Elle peut cependant être ajoutée à notre schéma. La révocation, dans notre contexte, signifie que le fournisseur de services a la possibilité de décider si un multi-coupons (et par extension un coupon) est toujours valide ou non. Pour révoquer un multi-coupons, le fournisseur de services doit calculer une nouvelle paire $(sk_{\mathcal{SP}}, pk_{\mathcal{SP}})$ de clés et les utilisateurs doivent mettre à jour la clé $pk_{\mathcal{SP}}$ dans leurs paramètres publics, ainsi que leur multi-coupons. Ensuite, le fournisseur de services révoque la signature qu'il a produite pendant la phase de retrait correspondant au multi-coupons révoqué. Cette révocation dépend donc du schéma de signature utilisé pour la construction de la signature Camenisch-Lysyanskaya.

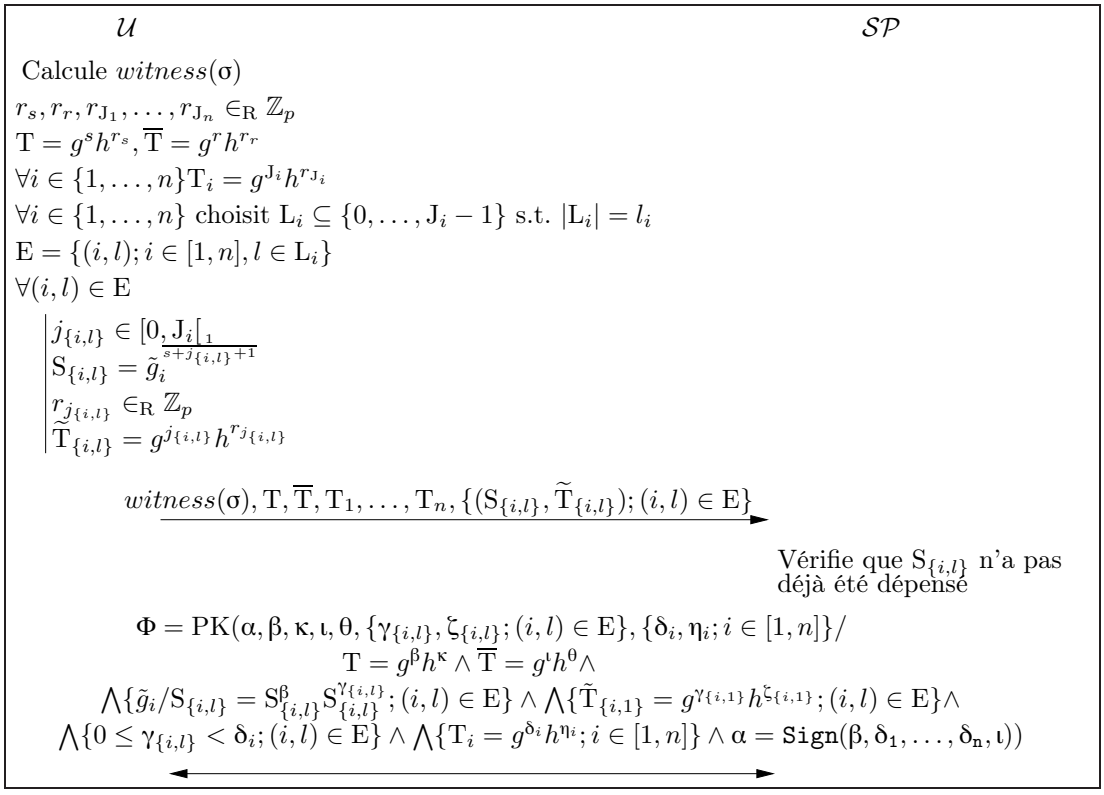

FIG. 11.5 – Protocole de Multi-dépense

Il est aussi possible, au fournisseur de services, d'ajouter une date de validité aux multi-coupons. Cette propriété est tout à fait intéressante pour un marchand qui souhaite organiser une opération commerciale délimitée dans le temps. Pour ce faire, le fournisseur de services ajoute une valeur d à sa signature lors de la phase de retrait de coupons. Ensuite, durant la phase de dépense, l'utilisateur prouve au fournisseur de services que la date contenue dans sa signature est plus grande que la date du jour. Nous décrivons plus en détail cette procédure dans le cadre de la monnaie électronique, au chapitre suivant.

11.4 Preuves de sécurité

Théorème 23. *Le système de multi-coupons est sûr, sous les hypothèses y-DDHI et q-SDH, dans le modèle de l'oracle aléatoire.*

11.4.1 Consistance

La consistance du schéma découle directement de l'équation de vérification $\sigma \stackrel{?}{=} \mathsf{Sign}_{sk_{\mathcal{SP}}}^{\mathrm{BBS}_l^{ext}}$ du protocole de retrait et de la vérification de la preuve de connaissance Φ du protocole de dépense.

11.4.2 Anonymat

Sans perte de généralité, nous supposons qu'il n'y a qu'une seule valeur monétaire possible V et qu'il n'y a pas de transfert de coupons[2]. De plus, comme nous l'avons dit, il est possible de réduire notre attaque au cas où l'adversaire ne fait qu'une requête à l'oracle Choose_b. La preuve d'anonymat de notre schéma est très semblable à celle que nous avons présentée en section 9.2.4 pour le schéma de signature aveugle à anonymat révocable. Les interactions entre l'adversaire $\mathcal{A}$ et les joueurs sont simulés par un distingueur $\mathcal{D}$ qui cherche à casser la *pseudo-randomness* de la fonction pseudo-aléatoire de Dodis-Yampolskiy. Nous nous plaçons, encore une fois, dans le modèle de l'oracle aléatoire.

Jeu 0 : Le Jeu 0 est une exécution normale de l'expérience menée par l'adversaire.

Avant de faire appel à son oracle Choose_b, $\mathcal{A}$ peut faire appel aux oracles AddU et CrptU afin d'ajouter de nouveaux utilisateurs au système et de récupérer certains des multi-coupons et en construire à l'aide de l'oracle $\mathsf{WithdrawU}$. À un moment de son attaque, $\mathcal{A}$ fait appel à son oracle Choose_b. $\mathcal{A}$ choisit donc deux utilisateurs $\mathcal{U}_0$ et $\mathcal{U}_1$ et exécute deux retraits. Il obtient les deux vues $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_0}$ et $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_1}$ correspondant aux deux retraits. Ces vues sont de la forme suivante :

$$\begin{aligned}
\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_0} &= \mathrm{J}_1^0, \ldots, \mathrm{J}_n^0, \mathrm{C}_0' = h_0^{s_0'} h_{n+1}^{r_0}, pok_1^0(\alpha, \beta : \mathrm{C}_0' = h_0^{\alpha} h_{n+1}^{\beta}), \\
\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_1} &= \mathrm{J}_1^1, \ldots, \mathrm{J}_n^1, \mathrm{C}_1' = h_1^{s_1'} h_{n+1}^{r_1}, pok_1^1(\alpha, \beta : \mathrm{C}_1' = h_1^{\alpha} h_{n+1}^{\beta}).
\end{aligned}$$

L'oracle vérifie que les deux utilisateurs ont encore un coupon non dépensé. Il renvoie à $\mathcal{A}$ les vues suivantes correspondant à une dépense d'un coupon de valeur V pour l'utilisateur $\mathcal{U}_b$ et $\mathcal{U}_{\bar{b}}$ suite au retrait effectué lors des vues $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_b}$ et $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_{\bar{b}}}$.

$$\begin{aligned}
\mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_b} &= (\mathrm{S}_b = \tilde{g}^{\frac{1}{s_b + j + 1}}, \Phi^b), \\
\mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_{\bar{b}}} &= (\mathrm{S}_{\bar{b}} = \tilde{g}^{\frac{1}{s_{\bar{b}} + j + 1}}, \Phi^{\bar{b}}).
\end{aligned}$$

À la fin de son attaque, $\mathcal{A}$ renvoie un bit $\hat{b}$. On note alors S_0 l'événement $\hat{b} = b$. On a alors

$$\mathbf{Adv}_{\mathrm{MC},\mathcal{A}}^{\mathrm{Indis}}(k) = |\Pr[\mathrm{S}_0] - 1/2|.$$

Jeu 1 : Dans ce jeu, lorsque $\mathcal{A}$ fait appel à l'oracle Choose_b sur les utilisateurs $\mathcal{U}_0$ et $\mathcal{U}_1$, $\mathcal{D}$ simule la première vue pour $\mathcal{A}$. Les vues des retraits sont de la forme :

$$\begin{aligned}
\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_0} &= \mathrm{J}_1^0, \ldots, \mathrm{J}_n^0, \mathrm{C}_0' = h_0^{s_0'} h_{n+1}^{r_0}, \tilde{pok}_1^0(\alpha, \beta : \mathrm{C}_0' = h_0^{\alpha} h_{n+1}^{\beta}), \\
\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_1} &= \mathrm{J}_1^1, \ldots, \mathrm{J}_n^1, \mathrm{C}_1' = h_1^{s_1'} h_{n+1}^{r_1}, pok_1^1(\alpha, \beta : \mathrm{C}_1' = h_1^{\alpha} h_{n+1}^{\beta}).
\end{aligned}$$

[2]Dans notre description, les étapes d'un transfert étant semblables à un retrait ou une dépense, nous pouvons faire cette hypothèse.

où $\tilde{pok}_1^b$ est une simulation de la preuve réelle. De même $\mathcal{D}$ simule la première vue des dépenses qu'il renvoie à $\mathcal{A}$ en réponse à la requête Choose_b :

$$\begin{aligned}\mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_b} &= (\mathrm{S}_b = \tilde{g}_i^{\frac{1}{s_b+j+1}}, \tilde{\Phi}^b),\\ \mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_{\bar{b}}} &= (\mathrm{S}_{\bar{b}} = \tilde{g}_i^{\frac{1}{s_{\bar{b}}+j+1}}, \Phi^{\bar{b}}).\end{aligned}$$

$\tilde{\Phi}^b$ est une preuve simulée de la connaissances des secrets de $\mathcal{U}_b$. $\mathcal{D}$ n'a pas besoin de simuler la preuve de connaissance de la signature σ_b, il simule juste la preuve de consistance de S, T et $\overline{\mathrm{T}}$. La simulation produite par $\mathcal{D}$ est calculatoirement indistinguable pour l'adversaire s'il ne constate aucune collision aux requêtes de hachage. Par les mêmes arguments que dans la preuve 9.2.4, on montre que cette probabilité est négligeable.

$\mathcal{A}$ renvoie un bit $\hat{b}$.

On note alors S_1 l'événement $\hat{b} = b$.

On obtient

$$|\Pr[\mathrm{S}_0] - \Pr[\mathrm{S}_1]| \leq \nu(k).$$

où $\nu(k)$ est une fonction négligeable.

Jeu 2 : Le Jeu 2 est identique au Jeu 1, à la différence que cette fois, $\mathcal{D}$ envoie un chiffrement ElGamal d'une valeur aléatoire $\tilde{\mathrm{A}}_b$ dans la preuve de connaissance $\tilde{pok}_2$ de la dépense de $\mathcal{U}_b$. La simulation produite par $\mathcal{D}$ est calculatoirement indistinguable pour l'adversaire s'il n'est pas capable de distinguer la simulation de la preuve d'une preuve réelle. $\mathcal{D}$ peut donc utiliser cet adversaire comme distingueur contre le chiffrement ElGamal. Cet avantage étant limité par l'avantage contre le problème DDH on obtient

$$|\Pr[\mathrm{S}_2] - \Pr[\mathrm{S}_1]| \leq \mathbf{Adv}_{\mathcal{D}}^{\mathrm{DDH}}(k).$$

Jeu 3 : Le Jeu 3 est identique au Jeu 2 à la différence que cette fois, $\mathcal{D}$ simule les preuves pour l'utilisateur $\mathcal{U}_{\bar{b}}$ de manière identique aux Jeux 1 et 2.

On obtient

$$|\Pr[\mathrm{S}_3] - \Pr[\mathrm{S}_2]| \leq \mathbf{Adv}_{\mathcal{D}}^{\mathrm{DDH}}(k) + \nu(k).$$

Jeu 4 : $\mathcal{D}$ reçoit de son challenger les paramètres pour la fonction DY : $(g, \mathbb{G})$. Il construit alors les paramètres pour $\mathcal{A}$ en posant $\tilde{g} = e(g, g)$ et $h_0 = g^r$ pour $r \in [0, q-1]$. Il choisit aussi les clés pour le fournisseur de services $sk_{\mathcal{SP}}$ et $pk_{\mathcal{SP}}$ et envoie toutes ces valeurs à $\mathcal{A}$.

Le Jeu 4 est semblable au Jeu 3 à la différence que $\mathcal{D}$ va cette fois donner à $\mathcal{A}$ une valeur aléatoire $\tilde{\mathrm{S}}^b$ pour la dépense de $\mathcal{U}_b$. Il simule alors la preuve $\tilde{\Phi}^b$ comme précédemment. Les vues des dépenses qu'il renvoie sont alors

$$\begin{aligned}\mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_b} &= (\tilde{\mathrm{S}}_b, \tilde{\Phi}),\\ \mathcal{V}_{\mathcal{A}}^{\mathsf{Redeem}_{\bar{b}}} &= (\mathrm{S}_{\bar{b}} = \tilde{g}_i^{\frac{1}{s_{\bar{b}}+j+1}}, \tilde{\Phi}^{\bar{b}}).\end{aligned}$$

$\mathcal{A}$ renvoie un bit $\hat{b}$. On note S_2 l'événement $\hat{b} = b$.

Jeu d **:** Le distingueur $\mathcal{D}$ s'attaque à la *pseudo randomness* de la PRF de Dodis-Yampolskiy. Il reçoit donc de son challengeur une valeur g^s, pour une valeur s secrète. Il peut ensuite demander à son challenger l'évaluation de $\mathrm{F}^{\mathrm{DY}}_{s,g}(x)$ pour des valeurs x de son choix. À un moment de son attaque, il envoie une valeur j qui n'a pas été préalablement requise. Son challenger choisit alors un bit d et répond soit $\mathrm{F}^{\mathrm{DY}}_{s,g}(j)$ si $d = 0$, soit une valeur aléatoire S si $d = 1$. $\mathcal{D}$ renvoie alors un bit $\hat{d}$ qui est sa supposition sur la valeur de b. Appelons $\tilde{\mathrm{S}}$ la valeur retournée par le challenger de $\mathcal{D}$

$\mathcal{D}$ va utiliser $\mathcal{A}$ comme distingueur pour son problème en utilisant la valeur $\tilde{\mathrm{S}}$ dans la vue $\mathcal{V}^{\mathsf{Redeem}_b}_{\mathcal{A}}$. Si $\hat{b} = b$ alors $\mathcal{D}$ renvoie $\hat{d} = 1$ à son challenger, sinon il renvoie 0. L'avantage de $\mathcal{D}$ est donné par :

$$\mathbf{Adv}^{\mathrm{random}}_{\mathrm{DY},\mathcal{D}}(k) = |\Pr[\hat{d} = 1|d = 1] - \Pr[\hat{d} = 1|d = 0]|.$$

Comme on a

$$\Pr[\mathrm{S}_3] = \Pr[\hat{d} = 1|d = 0]$$

et

$$\Pr[\mathrm{S}_4] = \Pr[\hat{d} = 1|d = 1]$$

on obtient finalement

$$\begin{aligned} |\Pr[\mathrm{S}_4] - \Pr[\mathrm{S}_3]| &= |\Pr[\hat{d} = 1|d = 1] - \Pr[\hat{d} = 1|d = 0]| \\ &= \mathbf{Adv}^{\mathrm{random}}_{\mathrm{DY},\mathcal{D}}(k). \end{aligned}$$

Jeu 5 : Ce jeu est identique au Jeu 4 à la différence que cette fois, $\mathcal{D}$ modifie la vue du deuxième retrait. Par les mêmes arguments, on obtient

$$|\Pr[\mathrm{S}_5] - \Pr[\mathrm{S}_4]| = \mathbf{Adv}^{\mathrm{random}}_{\mathrm{DY},\mathcal{D}}(k).$$

On remarque en outre, que dans la signature renvoyée par $\mathcal{D}$ aucune information sur le bit b n'apparaît. Par conséquent :

$$\Pr[\mathrm{S}_5] = 1/2.$$

Ainsi,

$$\begin{aligned} \mathbf{Adv}^{\mathrm{blind}}_{\mathrm{MC},\mathcal{A}}(k) &= 2|\Pr[\mathrm{S}_0] - \Pr[\mathrm{S}_5]| \\ &\leq 2(|\Pr[\mathrm{S}_1] - \Pr[\mathrm{S}_0]| + \ldots + |\Pr[\mathrm{S}_5] - \Pr[\mathrm{S}_4]) \\ &\leq 4\mathbf{Adv}^{\mathrm{random}}_{\mathrm{DY},\mathcal{D}}(k) + 4\mathbf{Adv}^{\mathrm{DDH}}_{\mathcal{A}}(k) + \nu(k). \end{aligned}$$

Ce qui conclut notre preuve.

11.4.3 Inforgeabilité

Nous ne traitons, dans cette preuve, que les attaques séquentielles. Afin de prouver l'inforgeabilité de notre schéma, nous montrons que s'il existe un adversaire $\mathcal{A}$ qui arrive à casser l'inforgeabilité du schéma avec une probabilité de succès non négligeable, alors il est possible de construire un algorithme $\mathcal{B}$ utilisant $\mathcal{A}$ qui casse l'inforgeabilité du schéma de signature BBS_l^{ext} utilisé dans le schéma. $\mathcal{B}$ reçoit de son challenger les paramètres $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e, p, g_1, g_2, g_\mathrm{T}, g, h, h_0, \ldots, h_{n+1})$ pour le schéma de signature et une clé publique de signature $pk_{\mathcal{SP}}$. Il a aussi accès à un oracle $\mathsf{Sign}_{sk_{\mathcal{SP}}}^{\mathrm{BBS}_l^{ext}}$ lui permettant d'obtenir une signature BBS_l^{ext} sur un bloc de message de son choix. Il choisit ensuite des éléments aléatoires $\tilde{g}_1, \ldots, \tilde{g}_n \in \mathbb{G}_\mathrm{T}$ et $h, h_0, \ldots, h_{n+1} \in \mathbb{G}_1$ et les envoie à $\mathcal{A}$ avec la clé $pk_{\mathcal{SP}}$.

$\mathcal{B}$ va simuler les requêtes aux oracles faites par $\mathcal{A}$ de la manière suivante :

$\mathsf{AddU}(\mathcal{U})$: lorsque $\mathcal{A}$ demande à ajouter un utilisateur $\mathcal{U}$, $\mathcal{B}$ enregistre son identité dans la liste HU.

$\mathsf{CrptU}(\mathcal{U})$: lorsque $\mathcal{A}$ demande à corrompre l'utilisateur $\mathcal{U}$, $\mathcal{B}$ lui renvoie la liste de tous les multi-coupons que cet utilisateur a retirés. $\mathcal{B}$ place alors cet utilisateur dans la liste CU.

$\mathsf{WithdrawSP}((\mathrm{J}_1, \ldots, \mathrm{J}_n, \mathrm{C}'))$: $\mathcal{A}$ demande à son oracle de retrait de créer un nouveau multi-coupons. Il donne en entrée le premier échange d'un protocole de retrait. $\mathcal{B}$ interagit ensuite avec $\mathcal{A}$ dans une preuve de connaissance et en rejouant $\mathcal{A}$ il est capable d'extraire les valeurs secrètes (s', r) contenue dans l'engagement C'. Il choisit ensuite la valeur r', il pose $s = s' + r'$ et calcule l'engagement C. Il fait ensuite une requête $\mathsf{Sign}_{sk_{\mathcal{SP}}}^{\mathrm{BBS}_l^{ext}}(s, \mathrm{J}_1, \ldots, \mathrm{J}_n, r)$ à son oracle. Il reçoit en réponse un couple (A, x) qu'il transmet à $\mathcal{A}$ avec la valeur r'. Il incrémente alors le compteur $\mathrm{N_C}$ de $\mathrm{J}_1 + \cdots + \mathrm{J}_n$ et calcule tous les identifiants de coupons que l'utilisateur vient de retirer et les enregistre dans une liste A_f.

$\mathsf{RedeemSP}(\mathcal{U})$: $\mathcal{A}$ veut faire dépenser un coupon à l'utilisateur $\mathcal{U}$. $\mathcal{B}$ répond à $\mathcal{A}$ comme décrit dans le protocole de retrait et n'accepte le coupon que si celui-ci est valide.

On suppose maintenant que $\mathcal{A}$ a réussi son attaque. Autrement dit, il produit une liste $((\mathrm{I}_1, \mathrm{V}_{j_1}), \ldots (\mathrm{I}_{\mathrm{N_C}+1}, \mathrm{V}_{j_{\mathrm{N_C}+1}})), j_i \in \{1, \ldots, n\}$ tels que tous les $(\mathrm{I}_i, \mathrm{V}_{j_i}), i \in \{1, \ldots, \mathrm{N}_c + 1\}, j_i \in \{1, \ldots, n\}$ soient acceptés lors d'un Redeem. Dans notre schéma $\mathrm{I}_i = \mathrm{S}_i = \tilde{g}_{j_i}^{\frac{1}{s+k+1}}$. $\mathcal{A}$ s'engage alors dans $\mathrm{N_C} + 1$ protocoles de dépense Redeem avec le fournisseur de services joué par $\mathcal{B}$. Ceux-ci, par définition, sont tous acceptés. $\mathcal{B}$ vérifie alors à l'aide de sa liste A_f quel identifiant proposé par $\mathcal{A}$, notons le $\tilde{\mathrm{S}}$, n'appartient pas à la liste A_f. À l'aide de la preuve de connaissance $\Phi_{\tilde{\mathrm{S}}}$ fournie par $\mathcal{A}$ lors de la dépense de ce coupon, $\mathcal{B}$ est capable de forger une signature BBS_l^{ext} en extrayant les valeurs $(s, \mathrm{J}_1, \ldots, \mathrm{J}_n, r, \mathrm{A}, x)$ de la preuve Φ. Le coupon étant valide, le couple (A, x) est une signature BBS_l^{ext} sur le message par blocs $(s, \mathrm{J}_1, \ldots, \mathrm{J}_n, r, \sigma)$. De plus, par définition du succès de $\mathcal{A}$, le bloc de messages est différent de toutes les requêtes que $\mathcal{B}$ a faites à son oracle de signature ou la signature est différente de toutes les réponses à ces requêtes.

Nous avons montré, au chapitre précédent, l'inforgeabilité du schéma BBS_l^{ext} sous l'hypothèse que le schéma BBS est inforgeable, c'est-à-dire sous l'hypothèse q-SDH. Il en découle que $\mathcal{A}$ n'a pu réussir son attaque qu'avec une probabilité négligeable.

La preuve que nous décrivons ici n'est valide que contre des attaques séquentielles en raison des rejeux que $\mathcal{B}$ doit faire pour extraire les valeurs s' et r. Il est possible cependant de rendre notre schéma sûr contre des attaques concurrentes en utilisant les mêmes techniques que celles utilisées pour le schéma de signature aveugle à anonymat révocable, notamment en utilisant des engagements extractables.

Chapitre 12

Extension à la monnaie électronique

Suite à la construction d'un nouveau schéma de multi-coupons, il nous est paru intéressant d'appliquer les nouvelles propriétés obtenues à la monnaie électronique.

Ce chapitre présente donc un schéma de monnaie électronique permettant à l'utilisateur de choisir le nombre de pièces qu'il veut retirer ainsi que leur valeur monétaire pour constituer son porte-monnaie. Ce schéma fonctionne en mode *hors ligne* et respecte les propriétés de sécurité définies au chapitre 10. Ce travail a été fait en collaboration avec Sébastien Canard et Aline Gouget [CGH07] et a été présenté à la conférence SAR-SSI 2007.

Sommaire

12.1 Motivation

Le protocole de monnaie que nous détaillons dans ce chapitre possède les mêmes propriétés que le protocole de multi-coupons présenté au chapitre précédent. Plus précisément :

- l'utilisateur choisit le nombre de pièces qu'il veut retirer, dans une limite fixée par le système,

- l'utilisateur choisit la valeur des pièces qu'il retire, dans un ensemble de valeurs fixées par le système.

L'ajout de ces nouvelles propriétés se fait tout en respectant les propriétés de sécurité que nous avons énoncées au chapitre 10.

A ces nouvelles propriétés, nous ajoutons aussi la possibilité d'imposer une durée de validité aux porte-monnaie. Ceci permet à la banque de gérer la taille de sa base de donnée. En effet, afin de détecter les doubles dépenses, la banque doit stocker toutes les pièces dépensées. Ajouter une date de validité aux porte-monnaie permet à la banque de rafraîchir sa base régulièrement.

12.2 Modèle de sécurité

Notre modèle de sécurité prouve la sécurité d'un schéma décrit par les algorithmes donnés à la définition 93. Il utilise le même formalisme que les modèles précédents.

Pour définir les oracles, nous utilisons les notations suivantes :
- HU est l'ensemble des utilisateurs honnêtes,
- CU est l'ensemble des utilisateurs corrompus,
- HM est l'ensemble des marchands honnêtes,
- CM est l'ensemble des marchands corrompus.

Définition des oracles

AddU(.) Cet oracle permet à l'adversaire d'ajouter un utilisateur honnête au système. Il spécifie en entrée l'identité $i \in \mathbb{N}$ de l'utilisateur. Si i n'appartient pas à HU$\cup$CU, alors l'oracle calcule les clés $(pk_{\mathcal{U}i}, sku_i)$ pour i, renvoie $pk_{\mathcal{U}i}$ à l'adversaire et ajoute $(i, pk_{\mathcal{U}i})$ à l'ensemble HU. Sinon il renvoie un message d'erreur.

USK(.) Cet oracle permet à l'adversaire de recevoir la clé privée d'un utilisateur honnête. Il spécifie en entrée l'identité i d'un utilisateur. Si $i \in$ HU, alors l'oracle renvoie la valeur $sk_{\mathcal{U}i}$ à l'adversaire, enlève $(i, pk_{\mathcal{U}i})$ de la liste HU et le met dans la liste CU. Sinon, il renvoie un message d'erreur. $\mathcal{A}$ reçoit aussi les différents porte-monnaie que i a pu retirer ainsi que la liste des pièces qu'il a dépensées.

AddM(.) Cet oracle fonctionne comme l'oracle AddU en prenant cette fois en entrée une identité i d'un marchand. Cette identité est alors ajoutée à la liste HM.

MSK(.) De même, cet oracle permet à l'adversaire de récupérer la clé privée d'un marchand honnête. Ce marchand est alors ajouté à la liste CM. $\mathcal{A}$ reçoit aussi la liste de pièces que le marchand a reçues au cours des dépenses précédentes.

Après avoir créé ou corrompu des utilisateurs et des marchands, l'adversaire peut accéder à différents oracles lui permettant de simuler les différents algorithmes d'un schéma de monnaie électronique.

$\mathsf{WithdrawU}(.)$ Cet oracle sert à simuler des protocoles de retrait entre un utilisateur honnête et une banque corrompue (jouée par l'adversaire) L'adversaire spécifie en entrée l'identité i de l'utilisateur ciblé ainsi que les paramètres du protocole $\mathsf{Withdraw}$. L'oracle vérifie que i appartient bien à HU et renvoie à l'adversaire, en fin de procédure, une vue $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}}$ de l'exécution.

$\mathsf{WithdrawB}(.)$ Cet oracle sert à simuler des protocoles de retrait entre une banque honnête et un utilisateur corrompu (joué par l'adversaire). L'adversaire donne en entrée les valeurs nécessaires à l'exécution d'un retrait. L'oracle renvoie à l'adversaire les réponses que la banque calcule lors d'une exécution de l'algorithme $\mathsf{Withdraw}$. Finalement, l'adversaire est capable de calculer un porte-monnaie W_i où i signifie qu'il a fait appel à cet oracle pour la i^{eme} fois.

$\mathsf{SpendU}(.,.)$ Cet oracle sert à simuler une dépense entre un utilisateur honnête et un marchand corrompu (joué par l'adversaire). L'adversaire fournit en entrée les informations d'un marchand nécessaires à une dépense, ainsi que l'identité i d'un utilisateur honnête que l'oracle va simuler. L'oracle vérifie tout d'abord que i appartient bien à HU et qu'il possède un porte-monnaie possédant au moins une pièce. Il lui renvoie ensuite les réponses faites par cet utilisateur. En fin de procédure, l'adversaire reçoit la pièce dépensée ainsi qu'une vue $\mathcal{V}_{\mathcal{A}}^{\mathsf{Spend}}$ de l'exécution.

$\mathsf{SpendM}(.,.)$ Cet oracle sert à simuler une exécution d'un protocole de dépense entre un marchand honnête et un utilisateur corrompu (joué par l'adversaire). L'adversaire fournit en entrée, l'identité du marchand ciblé ainsi que les entrées d'un protocole de dépense. L'oracle vérifie que le marchand appartient bien à HM et simule ses réponses pour l'adversaire. En fin de procédure, l'adversaire reçoit un porte-monnaie mis à jour.

Le dernier oracle est un oracle de challenge, utilisé pour l'anonymat du schéma. Cet oracle permet à l'adversaire de recevoir un challenge auquel il doit répondre. Le bit b est choisi avant l'appel à l'oracle et est inconnu de l'adversaire.

$\mathsf{Choose}_b(.,.,.)$ L'adversaire donne à l'oracle deux utilisateurs $\mathcal{U}_0$ et $\mathcal{U}_1$. L'oracle vérifie que les utilisateurs appartiennent à HU et $\mathcal{A}$ exécute deux protocoles de retrait avec ces utilisateurs. Il conserve les vues $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_0}$ et $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_1}$. Le bit b est choisi aléatoirement sans le concours de l'adversaire. Ensuite, deux protocoles de dépense Spend sont joués par les détenteurs des multi-coupons issus des échanges $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_b}$ et $\mathcal{V}_{\mathcal{A}}^{\mathsf{Withdraw}_{\bar{b}}}$. $\mathcal{A}$ reçoit les vues $\mathcal{V}_{\mathcal{A}}^{\mathsf{Spend}_b}$ et $\mathcal{V}_{\mathcal{A}}^{\mathsf{Spend}_{\bar{b}}}$.

Ces oracles nous permettent maintenant de décrire les différentes propriétés de sécurité de manière formelle, ainsi que les expériences menées par l'adversaire contre ses propriétés.

12.2.1 Consistance

La consistance du schéma garantit que si un honnête utilisateur exécute des protocoles de retrait Withdraw avec une banque honnête, alors le protocole ne renverra jamais de message d'erreur. Si un utilisateur exécute honnêtement un protocole de dépense Spend avec un marchand honnête, alors le marchand acceptera toujours la pièce.

12.2.2 Anonymat

L'anonymat d'un schéma de monnaie électronique garantit aux utilisateurs que la banque, même en coopérant avec des marchands ou des utilisateurs malhonnêtes n'apprendra rien sur les dépenses des utilisateurs mis à part les données publiques résultant des transactions.

Pour le schéma de monnaie électronique noté ME, tout adversaire $\mathcal{A}$, un bit $b \in \{0,1\}$ et tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{ME},\mathcal{A}}(k)$ (*cf.* figure 12.1). L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{blind}}_{\text{ME},\mathcal{A}}(k) = 2|\Pr[\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{ME},\mathcal{A}}(k) = b] - 1/2|.$$

On dit alors que le schéma est *indistinguable* si la fonction $\mathbf{Adv}^{\text{blind}(.)}_{\text{ME},\mathcal{A}}$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

Expérience $\mathbf{Exp}^{\text{blind}-\text{b}}_{\text{ME},\mathcal{A}}(k)$

$(pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{G}_{\mathcal{B}}(1^k)$
$b' \leftarrow \mathcal{A}(pk_{\mathcal{B}}, sk_{\mathcal{B}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{AddM}, \mathsf{MSK}, \mathsf{WithdrawU}, \mathsf{SpendU}, \mathsf{Choose}_b)$
retourne b'

FIG. 12.1 – Anonymat des multi-coupons

L'adversaire joue ici le rôle de la banque. Il reçoit donc la clé privée $sk_{\mathcal{B}}$ de la banque en plus des paramètres publics *params*. Il peut faire appel aux oracles AddU et AddM afin de créer de nouveaux utilisateurs et marchands et peut aussi les corrompre à l'aide des oracles USK et MSK. Il peut ensuite interagir avec ceux-ci afin de créer de nouveaux porte-monnaie et de nouvelles dépenses grâce aux oracles WithdrawU et SpendU. À tout moment, $\mathcal{A}$ peut faire appel à son oracle Choose_b en donnant en entrée deux utilisateurs de son choix. Il reçoit en réponse deux vues de protocoles de dépense d'une pièce.

Il continue ensuite ses requêtes de retrait et dépense mais n'est pas autorisé à corrompre les deux utilisateurs $\mathcal{U}_0$ et $\mathcal{U}_1$. De même, il ne peut faire dépenser entièrement les porte-monnaie issus des deux sessions $\mathcal{V}^{\mathsf{Withdraw}_0}_{\mathcal{A}}$ et $\mathcal{V}^{\mathsf{Withdraw}_1}_{\mathcal{A}}$ obtenues lors de l'appel à Choose_b.

En fin d'attaque, l'adversaire renvoie un bit b' correspondant à sa supposition sur le bit choisi par le challenger.

12.2.3 Inforgeabilité

Cette propriété permet à la banque d'être assurée qu'aucun adversaire ne pourra dépenser plus de pièces qu'il n'en a retirées, et ceci, même si les utilisateurs s'allient avec des marchands.

Pour le schéma de monnaie électronique noté ME, tout adversaire $\mathcal{A}$ et tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{Unforg}}_{\text{ME},\mathcal{A}}(k)$ (*cf.* figure 12.2). L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{Unforg}}_{\text{ME},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{Unforg}}_{\text{ME},\mathcal{A}}(k) = 1].$$

On dit alors que le schéma est *inforgeable* si la fonction $\mathbf{Adv}^{\text{Unforg}}_{\text{ME},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

Expérience $\mathbf{Exp}^{\text{Unforg}}_{\text{ME},\mathcal{A}}(k)$:

$(pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{G}_{\mathcal{B}}(1^k)$
$((\mathrm{I}_1, \mathrm{V}_{j_1}), \ldots (\mathrm{I}_{\mathrm{N_P}+1}, \mathrm{V}_{j_{\mathrm{N_P}+1}})) \leftarrow \mathcal{A}(pk_{\mathcal{B}} : \mathsf{AddU}, \mathsf{CrptU}, \mathsf{AddM}, \mathsf{MSK}, \mathsf{WithdrawB}, \mathsf{SpendM})$
`si` $\forall j \in [1, \mathrm{N}_p + 1]$,
$\mathsf{Spend}(\mathrm{I}_i, \mathrm{V}_{j_i}) = (\mathrm{S}_i, \pi_j)$ et
$\mathsf{Deposit}(pk_{\mathcal{M}}, sk_{\mathcal{M}}, pk_{\mathcal{B}}, sk_{\mathcal{B}}, \mathrm{S}_i, \pi_j) = 1$
et $(\mathrm{S}_i, \pi_i) \neq (\mathrm{S}_j, \pi_j)$ pour $1 \leq i < j < \mathrm{N_P} + 1$ et au plus N_p pièces ont été retirées
retourne 1 `sinon` retourne 0

FIG. 12.2 – Non-falsification supplémentaire

Tout d'abord, l'adversaire $\mathcal{A}$ peut créer autant d'utilisateurs qu'il le souhaite à l'aide de l'oracle AddU et recevoir les clés privées de ceux de son choix à l'aide de l'oracle USK. Il peut ensuite faire appel aux oracles WithdrawB et SpendM autant de fois qu'il le souhaite, en interagissant avec une banque honnête (dans cette attaque, il n'a pas connaissance de le clé privée de la banque ; les marchands cependant peuvent être corrompus à l'aide de l'oracle MSK). À chaque fois qu'il demande à un utilisateur de retirer un porte-monnaie, un compteur $\mathrm{N_P}$ est incrémenté du nombre de pièces retirées.

L'adversaire renvoie, à la fin de son attaque une liste $((\mathrm{I}_1, \mathrm{V}_{j_1}), \ldots, (\mathrm{I}_{\mathrm{N_P}+1}, \mathrm{V}_{j_{\mathrm{N_P}+1}}))$ pour $j \in \{1, \ldots, n\}$ de pièces de monnaie où I_i est l'identifiant de la pièce et V_{j_i} sa valeur. Si toutes les pièces sont acceptées par la banque lors d'une procédure Deposit et sont toutes distinctes, alors $\mathcal{A}$ a réussi son attaque.

12.2.4 Identification des fraudeurs

Cette propriété permet de prouver qu'il est impossible pour les utilisateurs ou les marchands de déposer deux fois une même pièce, sans être identifiés.

On suppose pour construire la définition de la propriété que nous sommes en présence de deux marchands honnêtes $\mathcal{M}_1$ et $\mathcal{M}_2$ qui reçoivent d'un adversaire $\mathcal{A}$ deux

pièces (S, π_1) et (S, π_2). La propriété d'*identification des fraudeurs* garantit alors que l'utilisateur ayant retiré cette pièce sera identifié.

Pour le schéma de monnaie électronique noté MC, tout adversaire $\mathcal{A}$ et tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{Ident}}_{\text{ME},\mathcal{A}}(k)$. L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{Ident}}_{\text{ME},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{Ident}}_{\text{ME},\mathcal{A}}(k) = 1].$$

On dit alors que le schéma est *identifiable pour les fraudeurs* si la fonction $\mathbf{Adv}^{\text{Ident}}_{\text{ME},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement à un paramètre de sécurité k).

L'adversaire reçoit les paramètres publics *params* et la clé publique de la banque $pk_{\mathcal{B}}$. Il peut ensuite ajouter de nouveaux utilisateurs et les corrompre à sa guise. Il peut retirer autant de pièces qu'il le souhaite en faisant appel à l'oracle WithdrawB. Il dépense celles-ci ensuite, auprès de marchands honnêtes à l'aide de l'oracle SpendM. On note A_i la liste de pièces (numéros de série) obtenus à partir de la clé publique $pk_{\mathcal{U}i}$. $\mathcal{A}$ gagne le jeu, si, pour un f et lors d'un protocole de dépense, la banque, simulant un marchand honnête, accepte deux fois une pièce dont le numéro de série appartient à A_f.

12.2.5 Non-diffamation

Cette propriété garantit que seuls les utilisateurs à l'origine de double dépenses peuvent en être accusés. La non-diffamation *faible* signifie que seuls les utilisateurs coupables d'avoir dépensé deux fois certaines pièces seront accusés, tandis que la non-diffamation *forte* signifie qu'un fraudeur sera accusé de double dépense uniquement pour les pièces qu'il aura effectivement dépensées deux fois.

Pour le schéma de monnaie électronique noté ME, tout adversaire $\mathcal{A}$ et tout $k \in \mathbb{N}$, on associe l'expérience $\mathbf{Exp}^{\text{Non}_\text{D}\text{iff}}_{\text{ME},\mathcal{A}}(k)$. L'avantage de l'adversaire est donné par

$$\mathbf{Adv}^{\text{Non}-\text{Diff}}_{\text{ME},\mathcal{A}}(k) = \Pr[\mathbf{Exp}^{\text{Non}-\text{Diff}}_{\text{ME},\mathcal{A}}(k) = 1].$$

On dit alors que le schéma est *non-diffamable* si la fonction $\mathbf{Adv}^{\text{Non}-\text{Diff}}_{\text{ME},\mathcal{A}}(.)$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial (relativement au paramètre de sécurité k).

Pour réaliser cette attaque, l'adversaire a à sa disposition la clé privée de la banque et des marchands et joue face à des utilisateurs honnêtes. Il peut ajouter de nouveaux utilisateurs grâce à l'oracle AddU et leur faire retirer des porte-monnaie grâce à l'oracle WithdrawB en jouant le rôle de la banque. Lorsqu'il fait appel à l'oracle Spend, il joue le rôle du marchand. Pour chaque porte-monnaie retiré par un utilisateur honnête, il peut lancer la requête SpendU autant de fois qu'il le désire. Il en résulte qu'un utilisateur peut lui fournir le numéro de série d'une pièce déjà dépensée. On note alors A_{ds} l'ensemble des numéros de série des pièces double-dépensées de cette manière. On note A_j, l'ensemble de tous les On dit que $\mathcal{A}$ a réussi son attaque, s'il produit deux couples (S, π_1) et (S, π_2) tels que :

- (S, π_1) et (S, π_2) sont des pièces valides et $S \notin A_{ds}$,

- l'algorithme $\mathsf{Identify}(\mathrm{S}, \pi_1, \pi_2)$ renvoie un couple $(pk_{\mathcal{U}}, \Pi)$ correct,
- l'algorithme $\mathsf{VerifyGuilt}(\mathrm{S}, \Pi, pk_{\mathcal{U}})$ est correct,
- $pk_{\mathcal{U}}$ est une clé publique d'un utilisateur honnête.

12.3 Description du schéma

Nous décrivons dans cette section le schéma que nous avons présenté dans [CGH07]. Ce schéma est une adaptation de [CGH06] à la monnaie électronique. De ce fait, les deux constructions sont très similaires et le déroulement suit le même principe.

Tout d'abord, l'utilisateur $\mathcal{U}$ qui souhaite retirer de l'argent auprès d'une banque $\mathcal{B}$ décide du nombre de pièces qu'il souhaite retirer et de leurs valeurs. Il interagit avec la banque pour construire son porte-monnaie. Celui-ci est constitué d'une signature de la banque sur des valeurs connues de lui seul, ainsi que d'un ensemble $\mathcal{S}$ qui représente les valeurs des pièces. La signature est faite, comme dans [CHL05] à l'aide de signature Camenisch-Lysyanskaya sur des engagements. Cependant, afin d'obtenir le porte-monnaie le plus compact possible, nous privilégions l'utilisation du schéma de signature BBS [BBS04] pour signer.

La dépense de la pièce se fait ensuite en deux étapes. Tout d'abord l'utilisateur calcule l'identifiant de sa pièce et un tag de sécurité. Il construit ensuite une signature de connaissance qu'il transmet au marchand, afin de lui prouver la validité de sa pièce et sa connaissance d'une signature de la banque.

Nous commençons tout d'abord par décrire les différents algorithmes nécessaires à la construction de notre schéma.

Soit $\mathcal{U}$ un utilisateur souhaitant retirer un porte-monnaie auprès d'une banque $\mathcal{B}$, pour les dépenser ensuite auprès d'un marchand $\mathcal{M}$.

Mise en place du système : Setup

Les paramètres du système *params* sont donnés par les valeurs suivantes :

- $k \in \mathbb{N}$ est le paramètre de sécurité.
- l est un entier servant à fixer la taille maximale d'un multi-coupons $\mathrm{M}ax = 2^l$.
- $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_\mathrm{T}$ sont trois groupes cycliques de même ordre p et de générateurs g_1, g_2 et g_T respectivement.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ est une application bilinéaire admissible.
- $\tilde{g}, \tilde{g}_1, \ldots, \tilde{g}_n$ sont des éléments choisis aléatoirement dans $\mathbb{G}_\mathrm{T}$,
- $g, h, h_0, \ldots h_{n+1}$ sont des éléments choisis aléatoirement dans $\mathbb{G}_1$.
- $\{\mathrm{V}_1, \ldots, \mathrm{V}_n\}$ sont les différentes valeurs possibles pour les pièces.
- $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p^*$ est une fonction de hachage résistante aux collisions.

$\mathsf{G}_\mathcal{B}$: la banque B calcule sa paire de clés ($sk_\mathcal{B}$, $pk_\mathcal{B}$)pour une signature BBS :

- elle choisit $\gamma \in \mathbb{Z}_p$ et pose $sk_\mathcal{B} = \gamma$,
- elle calcule ensuite $pk_\mathcal{B} = g_2^\gamma \mod p$.

$\mathsf{G}_\mathcal{U}$: l'utilisateur $\mathcal{U}$ calcule sa paire de clés :

- il choisit $u \in \mathbb{Z}_p$ et pose $sk_\mathcal{U} = u$,

- il calcule ensuite $pk_{\mathcal{U}} = \tilde{g}^u$.

$\mathsf{G}_{\mathcal{M}}$: le marchand $\mathcal{M}$ calcule sa paire de clés :
- il choisit $m \in \mathbb{Z}_p$ et pose $sk_{\mathcal{M}} = m$,
- il calcule ensuite $pk_{\mathcal{M}} = g^m$.

Retrait d'un porte-monnaie : Withdraw

Durant la phase de retrait, $\mathcal{U}$ interagit avec la banque $\mathcal{B}$ pour obtenir un nouveau porte-monnaie. $\mathcal{U}$ donne en entrée ses clés $sk_{\mathcal{U}}$ et $pk_{\mathcal{U}}$, la clé publique de la banque $pk_{\mathcal{B}}$ et les paramètres publics $params$. La banque donne en entrée ses clés $sk_{\mathcal{B}}$ et $pk_{\mathcal{B}}$, la clé publique de l'utilisateur $pk_{\mathcal{U}}$ et les paramètres publics $params$.

Le protocole de retrait permet à l'utilisateur $\mathcal{U}_1$ d'obtenir un nouveau porte-monnaie en interagissant avec $\mathcal{B}$ comme décrit dans la figure 12.3.

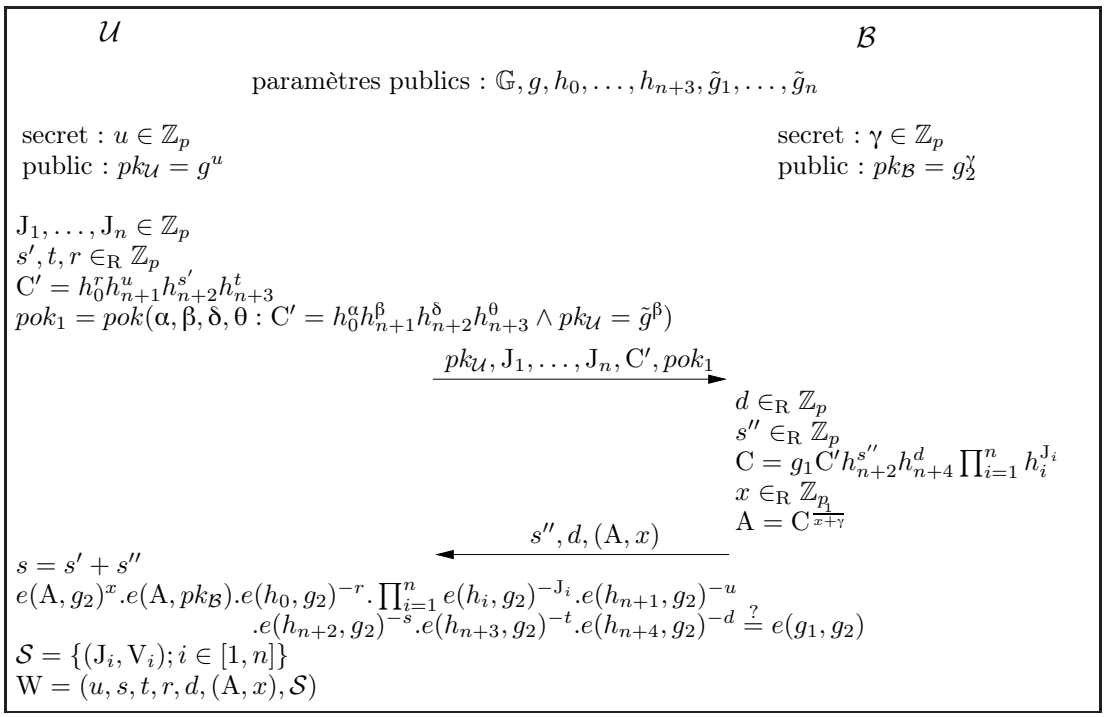

FIG. 12.3 – Protocole de Retrait

1. Tout d'abord, $\mathcal{U}$ s'authentifie auprès de la banque et celle-ci vérifie que le compte de ce dernier est suffisamment approvisionné.
2. $\mathcal{U}$ choisit ensuite les valeurs $\mathrm{J}_1, \ldots, \mathrm{J}_n$ correspondant au nombre de pièces de valeurs $\mathrm{V}_1, \ldots, \mathrm{V}_n$ qu'il souhaite retirer.
3. L'utilisateur et la banque contribuent tous deux dans la construction d'un secret s. $\mathcal{U}$ choisit aussi un secret t, associé à son nouveau porte-monnaie. $\mathcal{U}$ choisit donc des valeurs aléatoire s', t et r et s'engage sur ces valeurs ainsi que sur sa clé secrète.

4. Il prouve ensuite à la banque, sa connaissance des valeurs engagées r, s', t et u.
5. $\mathcal{U}$ et $\mathcal{B}$ exécutent ensuite une signature Camenisch-Lysyanskaya sur les valeurs $(r, \mathrm{J}_1, \ldots, \mathrm{J}_n, u, s, t, d)$ où $s = s' + s$" et s" a été choisi par $\mathcal{B}$ et d est une date de validité choisie par $\mathcal{B}$ également.
6. $\mathcal{U}$ possède alors une signature (A, x) valide si la relation suivante est vérifiée :

$$e(\mathrm{A}, g_2)^x e(\mathrm{A}, pk_{\mathcal{B}}) e(h_0, g_2)^{-r} \prod_{i=1}^{n} e(h_i, g_2)^{-\mathrm{J}_i} e(h_{n+1}, g_2)^{-u}$$
$$e(h_{n+2}, g_2)^{-s} e(h_{n+3}, g_2)^{-t} e(h_{n+4}, g)^{-d} = e(g_1, g_2)$$

7. $\mathcal{U}$ sauvegarde son porte-monnaie comme étant $\mathrm{W} = ((\mathrm{A}, x), sk_{\mathcal{U}}, r, u, s, t, d, \mathcal{S})$ où $\mathcal{S}$ est l'ensemble $\{(\mathrm{J}_i, \mathrm{V}_i); i \in [1, n]\}$.
8. La banque enregistre un débit de $\mathrm{J}_1\mathrm{V}_1 + \cdots \mathrm{J}_n\mathrm{V}_n$ sur le compte de $\mathcal{U}$.

Dépense d'une pièce : Spend

Lorsque l'utilisateur $\mathcal{U}$ souhaite dépenser une pièce de son porte-monnaie W, il commence tout d'abord par choisir la valeur V_i de la pièce qu'il souhaite utiliser. Il choisit ensuite le rang j de la pièce dans l'ensemble des pièces de valeur V_i, c'est-à-dire entre $[0, \mathrm{J}_i - 1]$.

L'utilisateur reçoit ensuite du marchand $\mathcal{M}$ une valeur $\mathrm{D} \in \mathbb{Z}_p^*$ correspondant à la date du jour, ainsi que des informations $info$ relatives à la transaction.

L'utilisateur agit ensuite de la manière suivante :

1. Il calcule $\mathrm{R} = \mathcal{H}(pk_{\mathcal{M}} || \mathrm{D} || info)$.
2. Il calcule l'identifiant S de la pièce à l'aide la fonction pseudo-aléatoire de Dodis-Yampolskiy, en prenant comme graine son secret s et comme générateur $\tilde{g}_i$ correspondant à la valeur V_i de la pièce et prenant en entrée j :

$$\mathrm{S} = \tilde{g}_i^{\frac{1}{s+j+1}}.$$

   Il calcule aussi le tag de sécurité T en utilisant le secret t comme graine, sa clé publique $pk_{\mathcal{U}}$ et la valeur aléatoire R :

$$\mathrm{T} = pk_{\mathcal{U}}(\tilde{g}_i^{\frac{1}{t+j+1}})^{\mathrm{R}}.$$

3. $\mathcal{U}$ doit ensuite prouver à la banque que son porte-monnaie ainsi que sa pièces sont valides. Pour cela, il doit prouver qu'il connaît une signature sur les secrets $r, \mathrm{J}_1, \ldots, \mathrm{J}_n, u, s, t, d, j$, que la valeur j est bien comprise dans l'intervalle $[0, \mathrm{J}_i - 1]$, sans révéler j et que la valeur d est inférieure la date du jour, sans révéler d. $\mathcal{U}$ envoie donc à $\mathcal{B}$ une signature de connaissance notée $\mathbf{\Phi}$.
4. Si la banque accepte la preuve $\mathbf{\Phi}$, alors elle accepte la pièce $(\mathrm{S}, \boldsymbol{\pi})$ où $\boldsymbol{\pi} = (\mathrm{R}, info, pk_{\mathcal{M}}, \mathbf{\Phi})$.
5. $\mathcal{U}$ met à jour son porte-monnaie.

La signature de connaissance $\mathbf{\Phi}$ est calculée de la manière suivante :

1. $\mathcal{U}$ calcule des engagements sur les valeurs secrètes $r, \mathrm{J}_1, \ldots, \mathrm{J}_n, u, s, t, d, j$:

$$\mathrm{C}_r = g_1^r g_2^{w_r}, \quad \mathrm{C}_u = g_1^u g_2^{w_u}, \quad \mathrm{C}_s = g_1^s g_2^{w_s}, \tag{12.1}$$

$$\mathrm{C}_{\mathrm{J}_1} = g_1^{\mathrm{J}_1} g_2^{w_{\mathrm{J}_1}}, \quad \cdots, \quad \mathrm{C}_{\mathrm{J}_n} = g_1^{\mathrm{J}_n} g_2^{w_{\mathrm{J}_n}}, \tag{12.2}$$

$$\mathrm{C}_t = g_1^t g_2^{w_t}, \quad \mathrm{C}_d = g_1^d g_2^{w_d}, \quad \mathrm{C}_j = g_1^j g_2^{w_j}. \tag{12.3}$$

2. La signature de connaissance Φ comprend alors
   - une preuve de connaissance de toutes les valeurs engagées, c'est-à-dire une preuve de connaissance de $\alpha_0 = r$, $\alpha_1 = \mathrm{J}_1$, $\cdots$, $\alpha_n = \mathrm{J}_n$, $\alpha_{n+1} = u$, $\alpha_{n+2} = s$, $\alpha_{n+3} = t$, $\alpha_{n+4} = d$, $\alpha_{n+5} = j$, $\eta_0 = w_r$, $\eta_1 = w_{\mathrm{J}_1}$, $\cdots$, $\eta_n = w_{\mathrm{J}_n}$, $\eta_{n+1} = w_u$, $\eta_{n+2} = w_s$, $\eta_{n+3} = w_t$, $\eta_{n+4} = w_d$ et $\eta_{n+5} = w_j$ tels que $\mathrm{C}_r, \mathrm{C}_u, \mathrm{C}_s, \mathrm{C}_{\mathrm{J}_1}, \cdots, \mathrm{C}_{\mathrm{J}_n}, \mathrm{C}_t, \mathrm{C}_d, \mathrm{C}_j$ sont consistants ;
   - une preuve de connaissance d'une signature de la banque $\mathcal{B}$ sur les valeurs $(r, \mathrm{J}_1, \ldots, \mathrm{J}_n, u, s, t, d)$. Cette preuve peut être faite comme décrite dans la section 3.4.2 et correspond à une preuve de connaissance de $\alpha_0, \cdots, \alpha_{n+4}, \beta, \delta$ tels que $(\beta, \delta) = \mathsf{Sign}(\alpha_0, \cdots, \alpha_{n+4})$ où $\alpha_0 = r$, $\alpha_1 = \mathrm{J}_1$, $\cdots$, $\alpha_n = \mathrm{J}_n$, $\alpha_{n+1} = u$, $\alpha_{n+2} = s$, $\alpha_{n+3} = t$, $\alpha_{n+4} = d$, $\beta = \mathrm{A}$ et $\delta = x$;
   - une preuve que la date courante D est inférieure ou égale à la date de validité d du porte-monnaie (sans révéler la valeur d), en utilisant la preuve qu'une valeur engagée est plus grande qu'une valeur connue décrite dans la section 3.3.2.2 ;
   - une preuve que la valeur j appartient à l'ensemble $\mathcal{J}_i = \{0, \ldots, \mathrm{J}_i - 1\}$ (sans révéler ni la valeur j ni les valeurs J_i) en utilisant la preuve qu'une valeur engagée est plus petite qu'une autre valeur engagée décrite au chapitre 3.3.2.3 ;
   - une preuve que S a été calculé en utilisant les valeurs s et j (c'est-à-dire en prouvant que la valeur s est la même que celle signée par la banque). Cette preuve est faite en utilisant le fait que $g_1 = (\mathrm{C}_s \mathrm{C}_j g_1)^{\frac{1}{s+j+1}} g_2^{-\frac{w_s+w_j}{s+j+1}}$. Ainsi, l'utilisateur prouve qu'il connaît θ et ζ tels que $\mathrm{S} = \tilde{g}_i^{\theta}$ et $g_1 = (\mathrm{C}_s \mathrm{C}_j g_1)^{\theta} g_2^{\zeta}$ où $\theta = \frac{1}{s+j+1}$ et $\zeta = -\frac{w_s+w_j}{s+j+1}$;
   - une preuve que la valeur T a été calculée en utilisant les valeurs engagées t, j et u (et que donc u et t sont les mêmes valeurs que celles signées par la banque). Cette preuve est faite en utilisant le fait que $g_1 = (\mathrm{C}_t \mathrm{C}_j g_1)^{\frac{1}{t+j+1}} g_2^{\frac{w_t+w_j}{t+j+1}}$. L'utilisateur prouve donc qu'il connaît ι et κ tels que $\mathrm{T} = g^{\alpha_{n+1}} \tilde{g}_i^{\mathrm{R}.\iota}$ et $g_1 = (\mathrm{C}_t \mathrm{C}_j g_1)^{\iota} g_2^{\kappa}$ où $\alpha_{n+1} = u$, $\iota = \frac{1}{t+j+1}$ et $\zeta = \frac{-(w_t+w_j)}{t+j+1}$.
3. La sortie du marchand correspond donc à une pièce de valeur V_i dont le numéro de série est S et une signature de connaissance

$$\pi = (\mathrm{R}, info, pk_{\mathcal{M}}, \mathrm{D}, \mathrm{T}, \mathrm{C}, \mathrm{V}_i, \mathrm{C}_r, \mathrm{C}_u, \mathrm{C}_s, \mathrm{C}_{\mathrm{J}_1}, \cdots, \mathrm{C}_{\mathrm{J}_n}, \mathrm{C}_t, \mathrm{C}_d, \mathrm{C}_j, \mathrm{P}, \Phi).$$

Dépôt d'une pièce : Deposit

Le marchand $\mathcal{M}$ souhaite déposer une pièce (S, π) auprès de la banque $\mathcal{B}$.

1. $\mathcal{M}$ envoie la pièce (S, π) à $\mathcal{B}$.
2. La banque vérifie que la preuve π contient un tag de sécurité T, une preuve de connaissance Φ et une valeur aléatoire R donnée par le marchand.

3. $\mathcal{B}$ vérifie la validité de Φ et la consistance de S.
4. Si (S, π) n'est pas une pièce valide, la banque la rejette et la procédure s'arrête.
5. Sinon, $\mathcal{B}$ vérifie s'il existe déjà dans sa base de données une entrée (S, π').
6. S'il n'en existe pas, $\mathcal{B}$ accepte la pièce et crédite le compte du marchand $\mathcal{M}$ de la valeur V_i et ajoute (S, π) dans la base des pièces dépensées.
7. S'il existe déjà une entrée pour S, la banque s'assure dans un premier temps que $\mathcal{M}$ n'est pas en train de rejouer la valeur R contenue dans π en la comparant à celle contenue dans π'. Si les deux valeurs sont identiques, alors $\mathcal{B}$ sait que le marchand est en train de frauder et refuse le dépôt de la pièce. Si R est différent de R′, $\mathcal{B}$ accepte de créditer le compte de $\mathcal{M}$ et ajoute l'entrée (S, π, π') à la liste des pièces double dépensées.

Identification des fraudeurs : Identify

Pour chaque entrée $(\mathrm{S}, \pi_1, \pi_2)$ dans la liste des double dépenses, $\mathcal{B}$ peut lancer la procédure Identify qui lui permet de retrouver les utilisateurs fraudeurs.

A partir des deux preuves π_1 et π_2, $\mathcal{B}$ obtient les valeurs $\mathrm{T}_1 \in \pi_1$ et $\mathrm{T}_2 \in \pi_2$ où nécessairement[1] $\mathrm{T}_1 = \tilde{g}^u(\tilde{g}_i^{\frac{1}{t+j+1}})^{\mathrm{R}_1}$ et $\mathrm{T}_2 = \tilde{g}^u(\tilde{g}_i^{\frac{1}{t+j+1}})^{\mathrm{R}_2}$ avec la même clé publique $pk_{\mathcal{U}}$. La banque ayant procédé aux étapes de vérification dans la procédure de dépôt, on a nécessairement que $\mathrm{R}_1 \neq \mathrm{R}_2$. Par conséquent, la banque peut calculer

$$\Big(\frac{\mathrm{T}_2^{\mathrm{R}_1}}{\mathrm{T}_1^{\mathrm{R}_2}}\Big)^{(\mathrm{R}_1-\mathrm{R}_2)^{-1}} = \Big(\frac{g^{u.\mathrm{R}_1}\tilde{g}_i^{\frac{\mathrm{R}_1.\mathrm{R}_2}{t+j+1}}}{g^{u.\mathrm{R}_2}\tilde{g}_i^{\frac{\mathrm{R}_1.\mathrm{R}_2}{t+j+1}}}\Big)^{(\mathrm{R}_1-\mathrm{R}_2)^{-1}} = \Big(\frac{g^{u.\mathrm{R}_1}}{g^{u.\mathrm{R}_2}}\Big)^{(\mathrm{R}_1-\mathrm{R}_2)^{-1}} = g^u = pk_{\mathcal{U}}$$

La preuve de culpabilité de l'utilisateur donc la clé publique est $pk_{\mathcal{U}}$ est donnée par $\Pi_{\mathrm{G}} = (\pi_1, \pi_2)$.

Vérification de la culpabilité : VerifyGuilt

A partir de la preuve de culpabilité $\Pi_{\mathrm{G}} = (\pi_1, \pi_2)$ obtenue suite à une identification, tout le monde est à même de vérifier la culpabilité de l'utilisateur possédant la clé publique $pk_{\mathcal{U}}$ en exécutant la même procédure que dans l'algorithme d'identification.

12.4 Preuves de sécurité

Théorème 24. *Le système de multi-coupons est sûr, sous les hypothèses y-DDHI et q-SDH, dans le modèle de l'oracle aléatoire.*

[1] Dans le cas contraire, cela signifierait qu'un adversaire a réussi à casser la propriété d'identification des fraudeurs, ce qui, comme nous le verrons par la suite, n'est pas possible.

12.4.1 Consistance

La consistance du schéma découle directement de l'équation de vérification $\sigma \stackrel{?}{=} \mathsf{Sign}^{\mathrm{BBS}_l^{ext}}_{sk_{\mathcal{SP}}}$ du protocole de retrait et de la vérification de la preuve de connaissance Φ du protocole de dépense.

12.4.2 Anonymat et inforgeabilité

L'anonymat et l'inforgeabilité du schéma de monnaie électronique se montrent de la même manière que pour les coupons. Par conséquent, nous renvoyons le lecteur aux preuves décrites en section 11.4.

L'anonymat se montre dans le modèle de l'oracle aléatoire et repose sur l'hypothèse y-DDHI. L'inforgeabilité se montre aussi dans le modèle de l'oracle aléatoire et repose sur l'hypothèse q-SDH.

12.4.3 Identification des fraudeurs

On rappelle que A_f représente la liste des numéros de série obtenus à partir de la clé publique $pk_{\mathcal{U}f}$ de l'utilisateur $\mathcal{U}_f$. Soit $\mathcal{A}$ un adversaire qui réussit à casser l'identification des fraudeurs de notre schéma avec probabilité non négligeable. Cela signifie que $\mathcal{A}$ a réussi à faire accepter deux pièces (S, π_1) et (S, π_2) à un marchand honnête $\mathcal{M}$ pour un S appartenant à une liste A_f. Le schéma étant inforgeable, l'adversaire n'a pu produire un numéro de série S n'appartenant à aucun A_f pour tout f ou tel que $\mathrm{S} \in \mathrm{A}_f$ et $\mathrm{S} \in \mathrm{A}_{f'}$ pour $f \neq f'$.

Les deux preuves π_1 et π_2 se décomposent en $(\mathrm{R}_1, \mathrm{T}_1, \Phi_1)$ et $(\mathrm{R}_2, \mathrm{T}_2, \Phi_2)$. Le marchand étant honnête, on peut supposer que $\mathrm{R}_1 \neq \mathrm{R}_2$ avec probabilité écrasante.

Comme S est valide, on peut l'écrire sous la forme $\mathrm{S} = \tilde{g}_i^{\frac{1}{s+j+1}}$ pour une valeur $j \in [0, \mathrm{J}_i - 1]$ et $s \in \mathbb{Z}_p$. De plus, les tags de sécurité $\mathrm{T}_1 = \tilde{g}^u \tilde{g}_i^{\frac{\mathrm{R}_1}{t+j+1}}$ et $\mathrm{T}_1 = \tilde{g}^u \tilde{g}_i^{\frac{\mathrm{R}_2}{t+j+1}}$ sont les seuls tags valides associés au numéro de série S dans ces deux transactions. La seule façon pour $\mathcal{A}$ de produire un autre tag pendant la phase de Spend est de simuler la preuve Φ, ce qui n'est possible qu'avec probabilité négligeable[2].

Ainsi, l'algorithme Identify peut toujours calculer la clé publique d'un fraudeur et produire la preuve associée. L'algorithme VerifyGuilt étant juste une ré-exécution du précédent, il s'en suit que la procédure de vérification de culpabilité sera toujours acceptée pour une sortie honnête de Identify.

12.4.4 Non-diffamation

Comme nous l'avons fait pour la non-diffamation d'identité du schéma de signature aveugle à anonymat révocable, il est possible d'utiliser un adversaire contre la non-diffamation de notre schéma pour construire un algorithme contre le problème du logarithme discret de plus. Nous détaillons rapidement ici, comment fonctionne la preuve, celle-ci étant très proche de 9.2.6.1.

[2]Les arguments utilisés sont semblables à ceux utilisés dans les preuves d'anonymat vues auparavant.

Soit $\mathcal{B}$ un algorithme contre le problème du logarithme discret de plus qui utilise l'adversaire $\mathcal{A}$ pour mener à bien son attaque. $\mathcal{B}$ a accès à un oracle Dlog et reçoit en entrée son instance, à savoir l'ensemble $(q, \tilde{g}, \mathbb{G}_{\mathrm{T}}, \tilde{g}^{x_1}, \tilde{g}^{x_2}, \ldots, \tilde{g}^{x_l})$. Il construit pour $\mathcal{A}$ les paramètres publics sauf les clés de la banque. $\mathcal{A}$ choisit $\gamma \in_{\mathrm{R}} \mathbb{Z}_p$ et pose $\Gamma = g_2^{\gamma}$.

$\mathcal{B}$ simule les réponses aux requêtes de $\mathcal{A}$ de la manière suivante :

$\mathsf{AddU}(\mathcal{U})$: lorsque $\mathcal{A}$ demande à son oracle d'ajouter un nouvel utilisateur, $\mathcal{B}$ lui renvoie une valeur $\tilde{g}^{x_i}$ de son entrée contre le logarithme discret de plus et ajoute cet utilisateur à la liste HU.

$\mathsf{USK}(pk_{\mathcal{U}})$: lorsque $\mathcal{A}$ demande à son oracle de corrompre un utilisateur, $\mathcal{B}$ vérifie que la clé publique $pk_{\mathcal{U}}$ correspond à la clé publique d'un utilisateur honnête, sinon il arrête la procédure. Il interroge son oracle Dlog en lui donnant en entrée $pk_{\mathcal{U}}$ et renvoie la réponse à $\mathcal{A}$. L'utilisateur est alors ajouté à CU.

$\mathsf{AddM}(\mathcal{M})($: lorsque $\mathcal{A}$ demande à son oracle d'ajouter un nouveau marchand, $\mathcal{B}$ choisit la clé secrète $sk_{\mathcal{M}} = m \in \mathbb{Z}_p$ pour le marchand et renvoie la clé publique $pk_{\mathcal{M}} = g^m$. Le marchand est ajouté à la liste HM.

$\mathsf{MSK}(pk_{\mathcal{M}})$: lorsque $\mathcal{A}$ souhaite corrompre un marchand, $\mathcal{B}$ est capable de lui renvoyer sa clé secrète. Le marchand est ajouté à la liste CM.

$\mathsf{WithdrawU}(pk_{\mathcal{U}})$: dans le modèle de l'oracle aléatoire, $\mathcal{B}$ est capable de simuler parfaitement les réponses aux requêtes de $\mathcal{A}$. La requête se faisant avec un utilisateur honnête, $\mathcal{B}$ ne connaît pas la clé secrète de cet utilisateur. Il doit donc simuler les preuves de connaissances en faisant appel à son oracle de hachage et en utilisant les méthodes de simulation que nous avons décrites dans les preuves précédentes. Il calcule tous les numéros de série résultants et les enregistre dans une liste A_f.

$\mathsf{SpendU}(\mathcal{U}, \mathrm{W})$: comme pour la requête précédente, $\mathcal{B}$ est capable de simuler de manière indistinguable pour $\mathcal{A}$ les réponses de l'utilisateur $\mathcal{U}$ à cette requête.

À la fin de son attaque et après avoir fait l requêtes à l'oracle USK, $\mathcal{A}$ renvoie deux pièces valides (S, π_1) et (S, π_2) qui cassent la non-diffamation du schéma. Par définition, la clé publique donnée par $\mathsf{Identify}(\mathrm{S}, \pi_1, \pi_2) = pk_{\mathcal{U}}$ est la clé publique d'un utilisateur de la liste HU. Elle n'a donc pas été donnée en entrée à l'oracle USK et donc $pk_{\mathcal{U}}$ est dans l'instance du logarithme discret de plus donnée à $\mathcal{B}$. En utilisant la significativité des preuves de connaissance, $\mathcal{B}$ est capable d'extraire les secrets (r, u, s, t, d, J_1, …, J_n) contenus dans les preuves π_1 et π_2. $\mathcal{B}$ a ainsi extrait $l + 1$ logarithmes discrets de son instance après seulement l requêtes à l'oracle Dlog. Il a donc cassé le problème du logarithme discret de plus.

Conclusion

Durant cette thèse, nous avons fourni un certain nombre de contributions relatives aux signatures pour l'anonymat et à leurs applications.

En premier lieu, nous avons construit un nouveau protocole d'authentification anonyme fondé sur les couplages et montré comment le modifier afin d'obtenir un schéma de signature d'anneau pour un anneau composé de deux membres. Il serait intéressant d'étudier si le type de construction utilisé peut être étendu à des anneaux plus grands, constitués de trois utilisateurs ou plus.

Nous nous sommes également penchés sur les signatures aveugles. En particulier, nous avons détecté une faille de sécurité s'appliquant à un certain type de tels schémas.

Puis nous avons abordé le thème principal de ce mémoire : les schémas de signature aveugle à anonymat révocable. Après avoir mis en évidence les insuffisances des modèles et schémas déjà existants, nous avons reconsidéré les propriétés que l'on pouvait en attendre et défini les algorithmes nécessaires à leur construction. Cela nous a amenés à spécifier un nouveau modèle de sécurité, couvrant toutes les propriétés que l'on peut souhaiter pour de telles signatures.

Nous avons alors conçu un nouveau schéma et prouvé sa sécurité dans notre modèle, obtenant ainsi le premier schéma de signature aveugle à anonymat révocable prouvé sûr. Notre preuve de sécurité ayant été obtenue dans le modèle de l'oracle aléatoire, la question de spécifier un schéma de signature aveugle à anonymat révocable sûr dans le modèle standard reste ouverte.

Enfin, nous nous sommes intéressés aux applications des signatures pour l'anonymat et plus particulièrement à deux d'entre elles : la monnaie et les coupons électroniques. Une limitation sévère des systèmes multi-coupons existants était que le nombre et la valeur des coupons devaient être pré-définis. Notre nouvelle construction, que nous avons étendue à la monnaie électronique, assouplit l'usage de tels systèmes en s'affranchissant de ces contraintes. Nous nous sommes également intéressés aux notions de transfert et de multi-dépenses de pièces (ou de coupons). Néanmoins, l'efficacité de nos solutions mériterait d'être encore améliorée, ce qui constitue une autre piste de recherche intéressante pour l'avenir.

Bibliographie

[Abe01] M. Abe – « A three-move blind signature scheme for polynomially many signatures. », in *EUROCRYPT'01* (B. Pfitzmann, éd.), Lecture Notes in Computer Science, vol. 2045, Springer-Verlag, 2001, p. 136–151.

[Abe02] — , « Personnal communication. », 2002.

[ACJT00] G. Ateniese, J. Camenisch, M. Joye & G. Tsudik – « A practical and provably secure coalition-resistant group signature scheme. », in *CRYPTO'00* [Bel00], p. 255–270.

[AL03] A. K. Awasthi & S. Lal – « Proxy blind signature scheme », Cryptology ePrint Archive, Report 2003/072, 2003, `http://eprint.iacr.org/`.

[AO00] M. Abe & T. Okamoto – « Provably secure partially blind signatures. », in *CRYPTO'00* [Bel00], p. 271–286.

[AO01] M. Abe & M. Ohkubo – « Provably secure fair blind signatures with tight revocation. », in *ASIACRYPT'01* [Boy01], p. 583–601.

[AOS02] M. Abe, M. Ohkubo & K. Suzuki – « 1-out-of-n signatures from a variety of keys. », in *ASIACRYPT'02* [Zhe02], p. 415–432.

[ASM07] M. H. Au, W. Susilo & Y. Mu – « Practical compact e-cash. », in *ACISP'07* (J. Pieprzyk, H. Ghodosi & E. Dawson, éds.), Lecture Notes in Computer Science, vol. 4586, Springer, 2007, p. 431–445.

[BB04] D. Boneh & X. Boyen – « Short signatures without random oracles. », in *EUROCRYPT'04* [CC04], p. 56–73.

[BBP04] M. Bellare, A. Boldyreva & A. Palacio – « An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. », in *EUROCRYPT'04* [CC04], p. 171–188.

[BBS04] D. Boneh, X. Boyen & H. Shacham – « Short group signatures. », in *CRYPTO'04* [Fra04], p. 41–55.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval & P. Rogaway – « Relations among notions of security for public-key encryption schemes », in *CRYPTO* [Kra98], p. 26–45.

[Bel00] M. Bellare (éd.) – *Advances in cryptology - crypto 2000*, Lecture Notes in Computer Science, vol. 1880, Springer, 2000.

[BG97] M. Bellare & S. Goldwasser – « Verifiable partial key escrow. », in *ACM Conference on Computer and Communications Security'97*, 1997, p. 78–91.

[BGK95] E. Brickell, P. Gemmel & D. Kravitz – « Trustee-based tracing extension to anonymous cash and the making of anonymous change. », in *6th ACM-SIAM*, ACM Press, 1995, p. 457–466.

[Bih03] E. Biham (éd.) – *Advances in cryptology - eurocrypt 2003, proceedings*, Lecture Notes in Computer Science, vol. 2656, Springer, 2003.

[BKM06] A. Bender, J. Katz & R. Morselli – « Ring signatures : Stronger definitions, and constructions without random oracles. », in *TCC'06* [HR06], p. 60–79.

[BM84] M. Blum & S. Micali – « How to generate cryptographically strong sequences of pseudo-random bits. », *SIAM J. Comput.* **13** (1984), no. 4, p. 850–864.

[BMW03] M. Bellare, D. Micciancio & B. Warinschi – « Foundations of group signatures : Formal definitions, simplified requirements, and a construction based on general assumptions. », in *EUROCRYPT'03* [Bih03], p. 614–629.

[BNPS03] M. Bellare, C. Namprempre, D. Pointcheval & M. Semanko – « The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. », *J. Cryptology* **16** (2003), no. 3, p. 185–215.

[Bol03] A. Boldyreva – « Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. », in *PKC'03* (Y. Desmedt, éd.), Lecture Notes in Computer Science, vol. 2567, Springer, 2003, p. 31–46.

[Bou00] F. Boudot – « Efficient proofs that a committed number lies in an interval. », in *EUROCRYPT'00* (B. Preneel, éd.), Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, 2000, p. 431–444.

[Boy01] C. Boyd (éd.) – *Advances in Cryptology - Asiacrypt '01, Proceedings*, Lecture Notes in Computer Science, vol. 2248, Springer-Verlag, 2001.

[BP97] N. Bari & B. Pfitzmann – « Collision-free accumulators and fail-stop signature schemes without trees. », in *EUROCRYPT'97* (W. Fumy, éd.), Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, p. 480–494.

[BR93] M. Bellare & P. Rogaway – « Random oracles are practical : A paradigm for designing efficient protocols. », in *ACM Conference on Computer and Communications Security*, 1993, p. 62–73.

[BR94] — , « Optimal asymmetric encryption. », in *EUROCRYPT'94*, 1994, p. 92–111.

[BR96] — , « The exact security of digital signatures - how to sign with rsa and rabin. », in *EUROCRYPT'96*, 1996, p. 399–416.

[Bra90] G. Brassard (éd.) – *Advances in Cryptology - Crypto '89, Proceedings*, Lecture Notes in Computer Science, vol. 435, Springer-Verlag, 1990.

[Bra93] S. Brands – « Untraceable off-line cash in wallets with observers (extended abstract). », in *CRYPTO'93* (D. R. Stinson, éd.), Lecture Notes in Computer Science, vol. 773, Springer-Verlag, 1993, p. 302–318.

[Bra95] — , « Restrictive blinding of secret-key certificates. », in *EUROCRYPT'95* [GQ95], p. 231–247.

[Bre02] E. Bresson – « Protocoles cryptographiques pour l'authentification et l'anonymat dans les groupes », Phd thesis, École polytechnique, October 2002.

[Bri93] E. F. Brickell (éd.) – *Advances in Cryptology - Crypto '92, Proceedings*, Lecture Notes in Computer Science, vol. 740, Springer-Verlag, 1993.

[BS04] D. Boneh & H. Shacham – « Group signatures with verifier-local revocation. », in *ACM Conference on Computer and Communications Security* (V. Atluri, B. Pfitzmann & P. D. McDaniel, éds.), ACM, 2004, p. 168–177.

[BSS02] E. Bresson, J. Stern & M. Szydlo – « Threshold ring signatures and applications to ad-hoc groups. », in *CRYPTO'02* [Yun02], p. 465–480.

[BSZ05] M. Bellare, H. Shi & C. Zhang – « Foundations of group signatures : The case of dynamic groups. », in *CT-RSA'05* (A. Menezes, éd.), Lecture Notes in Computer Science, vol. 3376, Springer, 2005, p. 136–153.

[BW06] X. Boyen & B. Waters – « Compact group signatures without random oracles. », in *EUROCRYPT'06* (S. Vaudenay, éd.), Lecture Notes in Computer Science, vol. 4004, Springer, 2006, p. 427–444.

[Can01] R. Canetti – « Universally composable security : A new paradigm for cryptographic protocols. », in *FOCS'01*, 2001, p. 136–145.

[CBL+07] L. Chen, A. N. E. B., H. Löhr, M. Manulis & A.-R. Sadeghi – « A privacy-protecting multi-coupon scheme with stronger protection against splitting. », in *Financial Cryptography'07*, 2007.

[CC04] C. Cachin & J. Camenisch (éds.) – *Advances in Cryptology - EUROCRYPT 2004, Proceedings*, Lecture Notes in Computer Science, vol. 3027, Springer-Verlag, 2004.

[CDS94] R. Cramer, I. Damgård & B. Schoenmakers – « Proofs of partial knowledge and simplified design of witness hiding protocols. », in *CRYPTO'94* (Y. Desmedt, éd.), Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, p. 174–187.

[CES+05] L. Chen, M. Enzmann, A.-R. Sadeghi, M. Schneider & M. Steiner – « A privacy-protecting coupon system. », in *Financial Cryptography'05* (A. S. Patrick & M. Yung, éds.), Lecture Notes in Computer Science, vol. 3570, Springer, 2005, p. 93–108.

[CFN88] D. Chaum, A. Fiat & M. Naor – « Untraceable electronic cash. », in *CRYPTO'88* (S. Goldwasser, éd.), Lecture Notes in Computer Science, vol. 403, Springer-Verlag, 1988, p. 319–327.

[CFT98] A. H. Chan, Y. Frankel & Y. Tsiounis – « Easy come - easy go divisible cash. », in *EUROCRYPT'98* [Nyb98], p. 561–575.

[CG07] S. Canard & A. Gouget – « Divisible e-cash systems can be truly anonymous. », in *EUROCRYPT'07* (M. Naor, éd.), Lecture Notes in Computer Science, vol. 4515, Springer, 2007, p. 482–497.

[CGH98] R. Canetti, O. Goldreich & S. Halevi – « The random oracle methodology, revisited (preliminary version). », in *STOC'98*, 1998, p. 209–218.

[CGH04] — , « On the random-oracle methodology as applied to length-restricted signature schemes. », in *TCC'04* (M. Naor, éd.), Lecture Notes in Computer Science, vol. 2951, Springer, 2004, p. 40–57.

[CGH06] S. Canard, A. Gouget & E. Hufschmitt – « A handy multi-coupon system. », in *ACNS'06* (J. Zhou, M. Yung & F. Bao, éds.), Lecture Notes in Computer Science, vol. 3989, 2006, p. 66–81.

[CGH07] — , « Handy compact e-cash system. », in *SAR-SSI'07*, 2007.

[CGT06] S. Canard, M. Gaud & J. Traoré – « Defeating malicious servers in a blind signatures based voting system. », in *Financial Cryptography'06*, 2006.

[Cha82] D. Chaum – « Blind signatures for untraceable payments. », in *CRYPTO'82* (R. L. R. D. Chaum & A. T. Sherman, éds.), Lecture Notes in Computer Science, Springer-Verlag, 1982, p. 153.

[Cha83] D. Chaum – « Blind signature system. », in *CRYPTO'83* (D. Chaum, éd.), Lecture Notes in Computer Science, Plenum Publishing, 1983, p. 153.

[Cha85] — , « Showing credentials without identification : Signatures transferred between unconditionally unlinkable pseudonyms. », in *EUROCRYPT'85*, 1985, p. 241–244.

[Cha87] — , « Blinding for unanticipated signatures. », in *EUROCRYPT'87*, 1987, p. 227–233.

[CHL05] J. Camenisch, S. Hohenberger & A. Lysyanskaya – « Compact e-cash. », in *EUROCRYPT'05* (R. Cramer, éd.), Lecture Notes in Computer Science, vol. 3494, Springer, 2005, p. 302–321.

[CL01] J. Camenisch & A. Lysyanskaya – « An identity escrow scheme with appointed verifiers. », in *CRYPTO'01* [Kil01], p. 388–407.

[CL02] — , « A signature scheme with efficient protocols. », in *SCN'02* (S. Cimato, C. Galdi & G. Persiano, éds.), Lecture Notes in Computer Science, vol. 2576, Springer, 2002, p. 268–289.

[CL04] — , « Signature schemes and anonymous credentials from bilinear maps. », in *CRYPTO'04* [Fra04], p. 56–72.

[CM94] J. M. Couveignes & F. Morain – « Schoof's algorithm and isogeny cycles. », in *ANTS'94* (L. M. Adleman & M.-D. A. Huang, éds.), Lecture Notes in Computer Science, vol. 877, Springer, 1994, p. 43–58.

[CM98] J. Camenisch & M. Michels – « A group signature scheme with improved efficiency. », in *ASIACRYPT'98* [OP98], p. 160–174.

[CM99] — , « Proving in zero-knowledge that a number is the product of two safe primes. », in *EUROCRYPT'99* [Ste99], p. 107–122.

[CMS96] J. Camenisch, U. M. Maurer & M. Stadler – « Digital payment systems with passive anonymity-revoking trustees », in *ESORICS'96*, vol. 1146, 1996, p. 33–43.

[CP92] D. Chaum & T. P. Pedersen – « Wallet databases with observers. », in *CRYPTO'92* [Bri93], p. 89–105.

[CP94] L. Chen & T. P. Pedersen – « New group signature schemes (extended abstract). », in *EUROCRYPT'94* (A. D. Santis, éd.), Lecture Notes in Computer Science, vol. 950, Springer-Verlag, 1994, p. 171–181.

[CR03] R. Canetti & T. Rabin – « Universal composition with joint state », in *CRYPTO* (D. Boneh, éd.), Lecture Notes in Computer Science, vol. 2729, Springer, 2003, p. 265–281.

[CR06] G. D. Crescenzo & A. Rubin (éds.) – *Financial cryptography and data security, fc 2006, proceedings*, Lecture Notes in Computer Science, Springer, 2006.

[CS97] J. Camenisch & M. Stadler – « Efficient group signature schemes for large groups (extended abstract). », in *CRYPTO'97* [Jr.97], p. 410–424.

[CT03a] S. Canard & J. Traoré – « List signature shcemes and application to electronic voting. », in *WCC'03*, 2003.

[CT03b] — , « On fair e-cash systems based on group signature schemes. », in *ACISP'03* (R. Safavi-Naini & J. Seberry, éds.), Lecture Notes in Computer Science, vol. 2727, Springer, 2003, p. 237–248.

[CvH91] D. Chaum & E. van Heyst – « Group signatures. », in *EUROCRYPT'91* (D. W. Davies, éd.), Lecture Notes in Computer Science, vol. 547, Springer-Verlag, 1991, p. 257–265.

[CZMS06] X. Chen, F. Zhang, Y. Mu & W. Susilo – « Efficient provably secure restrictive partially blind signatures from bilinear pairings. », in *Financial Cryptography'06* [CR06], p. 251–265.

[DBS04] R. Dutta, R. Barua, & P. Sarkar – « Provably secure authenticated tree based group key agreement protocol », Cryptology ePrint Archive, Report 2004/090, 2004, `http://eprint.iacr.org/`.

[DDN00] D. Dolev, C. Dwork & M. Naor – « Nonmalleable cryptography », *SIAM J. Comput.* **30** (2000), no. 2, p. 391–437.

[DH76] W. Diffie & M. Hellman – « New directions in cryptography », in *IEEE Transactions on Information Theory*, vol. 22, 1976, p. 644–654.

[DKNS04] Y. Dodis, A. Kiayias, A. Nicolosi & V. Shoup – « Anonymous identification in ad hoc groups. », in *EUROCRYPT'04* [CC04], p. 609–626.

[DP06] C. Delerablée & D. Pointcheval – « Dynamic fully anonymous short group signatures. », in *VIETCRYPT'06* (P. Q. Nguyen, éd.), Lecture Notes in Computer Science, vol. 4341, Springer, 2006, p. 193–210.

[dST98] A. die Solages & J. Traoré – « An efficient fair off-line electronic cash system with extensions to checks and wallets with observers. », in *Financial Cryptography'98* [Hir98], p. 275–295.

[DX06] B. Dou & C. Xu – « Analysis of some attacks on awasthi and lalâď ?s proxy blind signature scheme », Cryptology ePrint Archive, Report 2006/311, 2006, `http://eprint.iacr.org/`.

[DY05] Y. Dodis & A. Yampolskiy – « A verifiable random function with short proofs and keys. », in *PKC'05* (S. Vaudenay, éd.), Lecture Notes in Computer Science, vol. 3386, Springer, 2005, p. 416–431.

[EGL85] S. Even, O. Goldreich & A. Lempel – « A randomized protocol for signing contracts. », *Commun. ACM* **28** (1985), no. 6, p. 637–647.

[ElG85] T. ElGamal – « A public key cryptosystem and a signature scheme based on discrete logarithms », *IEEE Transactions on Information Theory* **31** (1985), no. 4, p. 469–472.

[FFS88] U. Feige, A. Fiat & A. Shamir – « Zero-knowledge proofs of identity. », *J. Cryptology* **1** (1988), no. 2, p. 77–94.

[Fis06] M. Fischlin – « Round-optimal composable blind signatures in the common reference string model. », in *CRYPTO'06* (C. Dwork, éd.), Lecture Notes in Computer Science, vol. 4117, Springer, 2006, p. 60–77.

[FK02] C. Fieker & D. R. Kohel (éds.) – *Algorithmic number theory, ants 2002, proceedings*, Lecture Notes in Computer Science, vol. 2369, Springer, 2002.

[FO97] E. Fujisaki & T. Okamoto – « Statistical zero knowledge protocols to prove modular polynomial relations. », in *CRYPTO'97* [Jr.97], p. 16–30.

[FOO92] A. Fujioka, T. Okamoto & K. Ohta – « A practical secret voting scheme for large scale elections. », in *ASIACRYPT'92* (J. Seberry & Y. Zheng, éds.), Lecture Notes in Computer Science, vol. 718, Springer-Verlag, 1992, p. 244–251.

[FOPS01] E. Fujisaki, T. Okamoto, D. Pointcheval & J. Stern – « Rsa-oaep is secure under the rsa assumption. », in *CRYPTO'01* [Kil01], p. 260–274.

[FP01] P.-A. Fouque & D. Pointcheval – « Threshold cryptosystems secure against chosen-ciphertext attacks. », in *ASIACRYPT'01*, Lecture Notes in Computer Science, vol. 2248, 2001, p. 351–368.

[FR94] G. Frey & H.-G. Rück – « A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves », *Math. Comput.* **62** (1994), no. 206, p. 865–874.

[Fra04] M. K. Franklin (éd.) – *Advances in Cryptology - Crypto 04*, Lecture Notes in Computer Science, vol. 3152, Springer-Verlag, 2004.

[FS86] A. Fiat & A. Shamir – « How to prove yourself : Pratical solutions of identifications and signature problems. », in *CRYPTO'86* (A. M. Odlyzko, éd.), Lecture Notes in Computer Science, vol. 263, 1986, p. 186–194.

[FSNS03] F.Zhang, R. Safavi-Naini & W. Susilo – « Efficient verifiably encrypted signature and partially blind signature from bilinear pairings », in *Indocrypt'03* (T. Johansson & S. Maitra, éds.), Lecture Notes in Computer Science, vol. 2904, Springer, 2003, p. 191–204.

[FTY96] Y. Frankel, Y. Tsiounis & M. Yung – « "indirect discourse proof" : Achieving efficient fair off-line e-cash. », in *ASIACRYPT'96* [KM96], p. 286–300.

[FTY98] — , « Fair off-line e-cash made easy. », in *ASIACRYPT'98* [OP98], p. 257–270.

[FY93] M. K. Franklin & M. Yung – « Secure and efficient off-line digital money (extended abstract). », in *ICALP'93* (A. Lingas, R. G. Karlsson & S. Carlsson, éds.), Lecture Notes in Computer Science, vol. 700, Springer, 1993, p. 265–276.

[GGM86] O. Goldreich, S. Goldwasser & S. Micali – « How to construct random functions. », *J. ACM* **33** (1986), no. 4, p. 792–807.

[GHS02] S. D. Galbraith, K. Harrison & D. Soldera – « Implementing the tate pairing. », in *ANTS'02* [FK02], p. 324–337.

[Gir90] M. Girault – « An identity-based identification scheme based on discrete logarithms modulo a composite number. », in *EUROCRYPT'90* (I. Damgård, éd.), Lecture Notes in Computer Science, vol. 473, Springer-Verlag, 1990, p. 481–486.

[GM82] S. Goldwasser & S. Micali – « Probabilistic encryption and how to play mental poker keeping secret all partial information », in *STOC*, ACM, 1982, p. 365–377.

[GM84] — , « Probabilistic encryption. », *J. Comput. Syst. Sci.* **28** (1984), no. 2, p. 270–299.

[GMR84] S. Goldwasser, S. Micali & R. L. Rivest – « A "paradoxical'"solution to the signature problem (abstract). », in *CRYPTO'84*, 1984, p. 467.

[GMR85] S. Goldwasser, S. Micali & C. Rackoff – « The knowledge complexity of interactive proof-systems (extended abstract) », in *STOC'85*, ACM, 1985, p. 291–304.

[GMR88] S. Goldwasser, S. Micali & R. L. Rivest – « A digital signature scheme secure against adaptive chosen-message attacks. », *SIAM J. Comput.* **17** (1988), no. 2, p. 281–308.

[GMR89] S. Goldwasser, S. Micali & C. Rackoff – « The knowledge complexity of interactive proof systems. », *SIAM J. Comput.* **18** (1989), no. 1, p. 186–208.

[GPS06a] S. Galbraith, K. Paterson & N. Smart – « Pairings for cryptographers », Cryptology ePrint Archive, Report 2006/165, 2006.

[GPS06b] M. Girault, G. Poupard & J. Stern – « On the fly authentication and signature schemes based on groups of unknown order », *J. Cryptology* **19** (2006), no. 4, p. 463–487.

[GQ95] L. C. Guillou & J. J. Quisquater (éds.) – *Advances in Cryptology - EUROCRYPT 1995, Proceedings*, Lecture Notes in Computer Science, vol. 921, Springer-Verlag, 1995.

[GT03] M. Gaud & J. Traoré – « On the anonymity of fair offline e-cash systems. », in *Financial Cryptography'03* (R. N. Wright, éd.), Lecture Notes in Computer Science, vol. 2742, Springer-Verlag, 2003, p. 34–50.

[Har77] R. Hartshorne – *Algebraic geometry*, Springer-Verlag, 1977.

[Hir98] R. Hirschfeld (éd.) – *Financial Cryptography, FC 1998, Proceedings*, Lecture Notes in Computer Science, vol. 1465, Springer-Verlag, 1998.

[HKKL07] C. Hazay, J. Katz, C.-Y. Koo & Y. Lindell – « Concurrently-secure blind signatures without random oracles or setup assumptions », in *TCC'07*, 2007.

[HL06] J. Herranz & F. Laguillaumie – « Blind ring signatures secure under the chosen-target-cdh assumption. », in *ISC'06* (S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis & B. Preneel, éds.), Lecture Notes in Computer Science, vol. 4176, Springer, 2006, p. 117–130.

[HLS05] E. Hufschmitt, D. Lefranc & H. Sibert – « A zero-knoweldge identifications scheme in gap diffie-hellman groups », 2005.

[HR06] S. Halevi & T. Rabin (éds.) – *Theory of cryptography, tcc 2006, proceedings*, Lecture Notes in Computer Science, vol. 3876, Springer, 2006.

[HS04] J. Herranz & G. Sáez – « Ring signature schemes for general ad-hoc access structures. », in *ESAS* (C. Castelluccia, H. Hartenstein, C. Paar & D. Westhoff, éds.), Lecture Notes in Computer Science, vol. 3313, Springer, 2004, p. 54–65.

[HT07] E. Hufschmitt & J. Traoré – « Fair blind signatures revisited », in *Pairing 2007*, 2007.

[JLO97] A. Juels, M. Luby & R. Ostrovsky – « Security of blind digital signatures (extended abstract). », in *CRYPTO'97* [Jr.97], p. 150–164.

[Jou00] A. Joux – « A one round protocol for tripartite diffie-hellman. », in *ANTS'00* (W. Bosma, éd.), Lecture Notes in Computer Science, vol. 1838, Springer, 2000, p. 385–394.

[Jr.97] B. S. K. Jr. (éd.) – *Advances in Cryptology - Crypto '97*, Lecture Notes in Computer Science, vol. 1294, Springer-Verlag, 1997.

[Kil01] J. Kilian (éd.) – *Advances in Cryptology - Crypto '01*, Lecture Notes in Computer Science, vol. 2139, Springer-Verlag, 2001.

[KK02a] M. Kim & K. Kim – « A New Identification Scheme Based on the Bilinear Diffie-Hellman Problem », in *The 7th Australian Conference on Information Security and Privacy, ACISP '02*, Springer-Verlag, 2002, p. 362–378.

[KK02b] — , « A New Identification Scheme Based on the GAP Diffie-Hellman Problem », in *The Third Symposium on Cryptography and Information Security*, 2002.

[KKL01] J. Kim, K. Kim & C. Lee – « An efficient and provably secure threshold blind signature. », in *ICISC'01* (K. Kim, éd.), Lecture Notes in Computer Science, vol. 2288, Springer, 2001, p. 318–327.

[KM96] K. Kim & T. Matsumoto (éds.) – *Advances in Cryptology - Asiacrypt '96, Proceedings*, Lecture Notes in Computer Science, vol. 1163, Springer-Verlag, 1996.

[Kob87] N. Koblitz – « Elliptic curve cryptosytems », in *Mathematics of Computation*, vol. 48, 1987, p. 203–209.

[KR96] J. Kilian & P. Rogaway – « How to protect des against exhaustive key search », in *CRYPTO* (N. Koblitz, éd.), Lecture Notes in Computer Science, vol. 1109, Springer, 1996, p. 252–267.

[Kra98] H. Krawczyk (éd.) – *Advances in cryptology - crypto '98*, Lecture Notes in Computer Science, vol. 1462, Springer, 1998.

[KY03] A. Kiayias & M. Yung – « Extracting group signatures from traitor tracing schemes. », in *EUROCRYPT'03* [Bih03], p. 630–648.

[KZ06] A. Kiayias & H.-S. Zhou – « Concurrent blind signatures without random oracles. », in *SCN'06* (R. D. Prisco & M. Yung, éds.), Lecture Notes in Computer Science, vol. 4116, Springer, 2006, p. 49–62.

[Lag05] F. Laguillaumie – « On the security of pairing-based signatures with controlled verification », Thèse, Université de Caen Bass-Normandie, 2005.

[LL97] C. H. Lim & P. J. Lee – « A key recovery attack on discrete log-based schemes using a prime order subgroup », in *CRYPTO'97* [Jr.97], p. 249–63.

[LL04] J. I. Lim & D. H. Lee (éds.) – *Information security and cryptology, icisc 2003, proceedings*, Lecture Notes in Computer Science, vol. 2971, Springer, 2004.

[LM95] R. Lercier & F. Morain – « Counting the number of points on elliptic curves over finite fields : Strategies and performance. », in *EUROCRYPT'95*, 1995, p. 79–94.

[LR98] A. Lysyanskaya & Z. Ramzan – « Group blind digital signatures : A scalable solution to electronic cash », in *Financial Cryptography'98* [Hir98], p. 184–197.

[LRSW99] A. Lysyanskaya, R. L. Rivest, A. Sahai & S. Wolf – « Pseudonym systems. », in *Selected Areas in Cryptography* (H. M. Heys & C. M. Adams, éds.), Lecture Notes in Computer Science, vol. 1758, Springer, 1999, p. 184–199.

[LW03] J. Lv & X. Wang – « Verifiable ring signature. », in *CANS'03*, Proc. of DMS 2003, 2003, p. 663–667.

[LWW03] J. K. Liu, V. K. Wei & D. S. Wong – « A separable threshold ring signature scheme. », in *ICISC'03* [LL04], p. 12–26.

[LWW04] — , « Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). », in *ACISP'04* (H. Wang, J. Pieprzyk & V. Varadharajan, éds.), Lecture Notes in Computer Science, vol. 3108, Springer, 2004, p. 325–335.

[MB01] G. Maitland & C. Boyd – « Fair electronic cash based on a group signature scheme. », in *ICICS'01* [QOZ01], p. 461–465.

[MB02] — , « A provably secure restrictive partially blind signature scheme. », in *PKC'02* (D. Naccache & P. Paillier, éds.), Lecture Notes in Computer Science, vol. 2274, Springer, 2002, p. 99–114.

[Men94] A. J. Menezes – *Elliptic curve public key cryptosystems*, Kluwer Academic Publishers, 1994, Foreword By-Neal Koblitz.

[Mil85] V. S. Miller – « Use of elliptic curves in cryptography. », in *CRYPTO'85* (H. C. Williams, éd.), Lecture Notes in Computer Science, vol. 218, Springer-Verlag, 1985, p. 417–426.

[Mil86] — , « Short program for functions on curves », 1986, `hrrp://crypto.standford.edu/miller/miller.ps`.

[MOV91] A. Menezes, T. Okamoto & S. A. Vanstone – « Reducing elliptic curve logarithms to logarithms in a finite field », in *STOC'91*, ACM, 1991, p. 80–89.

[MOV93] — , « Reducing elliptic curve logarithms to logarithms in a finite field. », *IEEE Transactions on Information Theory* **39** (1993), no. 5, p. 1639–1646.

[MvOV01] A. Menezes, P. van Ooschot & S. Vanstone – *Handbook of Applied Cryptography*, 5 éd., CRC Press, 2001.

[Nao02] M. Naor – « Deniable ring authentication. », in *CRYPTO'02* [Yun02], p. 481–498.

[Ngu06] L. Nguyen – « Privacy-protecting coupon system revisited », in *FC'06* [CR06].

[NS00] T. Nakanishi & Y. Sugiyama – « Unlinkable divisible electronic cash. », in *ISW'00* (J. Pieprzyk, E. Okamoto & J. Seberry, éds.), Lecture Notes in Computer Science, vol. 1975, Springer, 2000, p. 121–134.

[NSN04] L. Nguyen & R. Safavi-Naini – « Efficient and provably secure trapdoor-free group signature schemes from bilinear pairings. », in *ASIACRYPT'04* (P. J. Lee, éd.), Lecture Notes in Computer Science, vol. 3329, Springer, 2004, p. 372–386.

[NT00] K. Q. Nguyen & J. Traoré – « An online public auction protocol protecting bidder privacy. », in *ACISP'00* (E. Dawson, A. Clark & C. Boyd, éds.), Lecture Notes in Computer Science, vol. 1841, Springer, 2000, p. 427–442.

[Nyb98] K. Nyberg (éd.) – *Advances in Cryptology - EUROCRYPT 1998, Proceedings*, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, 1998.

[Oka92] T. Okamoto – « Provably secure and practical identification schemes and corresponding signature schemes. », in *CRYPTO'92* [Bri93], p. 31–53.

[Oka95] — , « An efficient divisible electronic cash scheme. », in *CRYPTO'95* (D. Coppersmith, éd.), Lecture Notes in Computer Science, vol. 963, Springer-Verlag, 1995, p. 438–451.

[Oka06] — , « Efficient blind and partially blind signatures without random oracles. », in *TCC'06* [HR06], p. 80–99.

[OO89a] K. Ohta & T. Okamoto – « Disposable zero-knowledge authentications and their applications to untraceable electronic cash », in *CRYPTO'89* [Bra90], p. 481–496.

[OO89b] — , « Divertible zero knowledge interactive proofs and commutative random self-reducibility. », in *EUROCRYPT'89* (J. J. Quisquater & J. Vandewalle, éds.), Lecture Notes in Computer Science, vol. 434, Springer-Verlag, 1989, p. 134–148.

[OO98] — , « On concrete security treatment of signatures derived from identification. », in *CRYPTO'98* [Kra98], p. 354–369.

[OP98] K. Ohta & D. Pei (éds.) – *Advances in Cryptology - Asiacrypt '98, Proceedings*, Lecture Notes in Computer Science, vol. 1514, Springer-Verlag, 1998.

[OP01] T. Okamoto & D. Pointcheval – « React : Rapid enhanced-security asymmetric cryptosystem transform. », in *CT-RSA'01* (D. Naccache, éd.), Lecture Notes in Computer Science, vol. 2020, Springer, 2001, p. 159–175.

[Pai99] P. Paillier – « Public-key cryptosystems based on composite degree residuoisity classes. », in *EUROCRYPT'99* [Ste99], p. 129–140.

[Ped91] T. P. Pedersen – « Non-interactive and information-theoretic secure verifiable secret sharing. », in *CRYPTO'91* (J. Feigenbaum, éd.), Lecture Notes in Computer Science, vol. 576, Springer-Verlag, 1991, p. 129–140.

[Pet97] H. Petersen – « How to convert any digital signature scheme into a group signature scheme. », in *Security Protocols Workshop* (B. Christianson, B. Crispo, T. M. A. Lomas & M. Roe, éds.), Lecture Notes in Computer Science, vol. 1361, Springer, 1997, p. 177–190.

[Poi96] D. Pointcheval – « Les preuves de connaissances et leurs preuves de sécurité », Thèse, Université de Caen, 1996.

[Poi01] D. Pointcheval – « Practical security in public-key cryptography. », in *Information Security and Cryptology* (K. Kim, éd.), Lecture Notes in Computer Science, vol. 2288, Springer, 2001, p. 1–17.

[Poi02] — , *Le chiffrement asymétrique et la sécurité prouvée*, École Normale Supérieure, 2002, Habilitation à Diriger des Recherches.

[PS96a] D. Pointcheval & J. Stern – « Provably secure blind signature schemes. », in *ASIACRYPT'96* [KM96], p. 252–265.

[PS96b] — , « Security proofs for signature schemes. », in *EUROCRYPT'96* (U. M. Maurer, éd.), Lecture Notes in Computer Science, vol. 1070, Springer-Verlag, 1996, p. 387–398.

[PS98] G. Poupard & J. Stern – « Security analysis of a practical "on the fly" authentication and signature generation. », in *EUROCRYPT'98* [Nyb98], p. 422–436.

[PS00] D. Pointcheval & J. Stern – « Security arguments for digital signatures and blind signatures. », *J. Cryptology* **13** (2000), no. 3, p. 361–396.

[QOZ01] S. Qing, T. Okamoto & J. Zhou (éds.) – *Information and communications security, icics 2001, proceedings*, Lecture Notes in Computer Science, vol. 2229, Springer, 2001.

[Rom90] J. Rompel – « One-way functions are necessary and sufficient for secure signatures », in *STOC'90*, ACM, 1990, p. 387–394.

[RSA78] R. L. Rivest, A. Shamir & L. M. Adleman – « A method for obtaining digital signatures and public-key cryptosystems. », *Commun. ACM* **21** (1978), no. 2, p. 120–126.

[RST01] R. L. Rivest, A. Shamir & Y. Tauman – « How to leak a secret. », in *ASIACRYPT'01* [Boy01], p. 552–565.

[Sat02] T. Satoh – « On p-adic point counting algorithms for elliptic curves over finite fields. », in *ANTS'02* [FK02], p. 43–66.

[Sch89] C.-P. Schnorr – « Efficient identification and signatures for smart cards. », in *CRYPTO'89* [Bra90], p. 239–252.

[Sch91] — , « Efficient signature generation by smart cards. », *Journal of Cryptology* **4** (1991), no. 3, p. 161–174.

[Sch95] R. Schoof – « Counting points on ellptic curves over finite fields », *Journal de théorie des nombres de Bordeaux* (1995), p. 219–254.

[Sch01] C.-P. Schnorr – « Security of blind discrete log signatures against interactive attacks. », in *ICICS'01* [QOZ01], p. 1–12.

[SCPY94] A. D. Santis, G. D. Crescenzo, G. Persiano & M. Yung – « On monotone formula closure of szk », in *FOCS'94*, IEEE, 1994, p. 454–465.

[Sho00] V. Shoup – « Using hash functions as a hedge against chosen ciphertext attack. », in *EUROCRYPT'00*, 2000, p. 275–288.

[Sho01] — , « Oaep reconsidered. », in *CRYPTO'01* [Kil01], p. 239–259.

[Sho06] — , « Sequences of games : a tool for taming complexity in security proofs. », 2006.

[Sil86] J. Silverman – *The arithmetic of elliptic curves*, Springer-Verlag, 1986.

[SLC04] J. Shao, R. Lu & Z. Cao – « A New Efficient Identification Scheme Based on the Strong Diffie-Hellman Assumption », in *International Symposium on Future Software Technology*, 2004.

[SM03] W. Susilo & Y. Mu – « Non-interactive deniable ring authentication. », in *ICISC'03* [LL04], p. 386–401.

[SPC95] M. Stadler, J.-M. Piveteau & J. Camenisch – « Fair blind signatures. », in *EUROCRYPT95'95* [GQ95], p. 209–219.

[Ste99] J. Stern (éd.) – *Advances in Cryptology - EUROCRYPT 1999, Proceedings*, Lecture Notes in Computer Science, vol. 1592, Springer-Verlag, 1999.

[Tra95] J. Traoré – « An untraceable off-line electronic cash system based on the factorization problem. », 1995, soumis à AAECC'95.

[Tra97] — , « Making unfair a "fair" blind signature scheme. », in *ICICS'97*, 1997, p. 386–397.

[Tra99] J. Traoré – « Group signatures and their relevance to privacy-protecting off-line electronic cash systems. », in *ACISP'99* (J. Pieprzyk, R. Safavi-Naini & J. Seberry, éds.), Lecture Notes in Computer Science, vol. 1587, Springer, 1999, p. 228–243.

[Tur37] A. Turing – « On computable numbers, with an application to the entscheidung's problem. », in *Proceedings of the London Mathematical Society*, vol. 42, 1937, p. 230–265.

[TWC+04] P. P. Tsang, V. K. Wei, T. K. Chan, M. H. Au, J. K. Liu & D. S. Wong – « Separable linkable threshold ring signatures. », in *INDOCRYPT'04* (A. Canteaut & K. Viswanathan, éds.), Lecture Notes in Computer Science, vol. 3348, Springer, 2004, p. 384–398.

[vSN92] S. H. von Solms & D. Naccache – « On blind signatures and perfect crimes. », *Computers & Security* **11** (1992), no. 6, p. 581–583.

[Wan07] G. Wang – « Bibliography on blind signatures », 2007, http ://icsd.i2r.a-star.edu.sg/staff/guilin/bible/blind-sign.htm.

[Wei40] A. Weil – « Sur les fonctions algébriques de constantes finies », in *Comptes rendu de l'Académie des sciences*, vol. 210, 1940, p. 592–594.

[XY04] S. Xu & M. Yung – « Accountable ring signatures : A smart card approach. », in *CARDIS'04* (J.-J. Quisquater, P. Paradinas, Y. Deswarte & A. A. E. Kalam, éds.), Kluwer, 2004, p. 271–286.

[Yun02] M. Yung (éd.) – *Advances in cryptology - crypto 2002*, Lecture Notes in Computer Science, vol. 2442, Springer, 2002.

[YWW03] G. Yao, G. Wang & Y. Wang – « An Improved Identification Scheme », in *Progress in Computer Science and Applied Logic*, Berkhauser-Verlag, November 2003.

[Zhe02] Y. Zheng (éd.) – *Advances in Cryptology - Asiacrypt '02, Proceedings*, Lecture Notes in Computer Science, vol. 2501, Springer-Verlag, 2002.

[ZK02] F. Zhang & K. Kim – « Id-based blind signature and ring signature from pairings », in *ASIACRYPT'02* [Zhe02], p. 533–547.

[ZSNL03] F. Zhang, R. Safavi-Naini & C.-Y. Lin – « New proxy signature, proxy blind signature and proxy ring signature schemes from bilinear pairing », Cryptology ePrint Archive, Report 2003/104, 2003, `http://eprint.iacr.org/`.

Résumé de la thèse. Les questions d'anonymat surgissent dans de nombreux contextes et tout particulièrement dans celui des transactions électroniques. Il est souvent souhaitable de protéger l'identité des participants afin d'éviter la constitution de profils de consommateurs ou de bases de données de renseignements commerciaux. De nombreuses solutions cryptographiques ont été apportées afin de renforcer la confiance des utilisateurs dans ces applications. Une nouvelle approche dans l'élaboration de mécanismes d'anonymat sûrs et performants s'appuie sur des applications bilinéaires (couplages de Weil et de Tate sur les courbes elliptiques).

Dans cette thèse nous présentons tout d'abord un état de l'art des différentes signatures utilisées pour l'anonymat en cryptographie, en particulier les signatures de groupe, les signatures aveugles et les signatures d'anneau. Dans ce contexte nous décrivons un nouveau protocole d'authentification et montrons comment il peut être converti en signature d'anneau. Notre étude porte ensuite sur les signatures aveugles à anonymat révocable. Il s'agit de signatures aveugles dont l'anonymat et l'intraçabilité peuvent être révoqués par une autorité. Nous proposons le premier véritable modèle de sécurité pour ces signatures, ainsi qu'une nouvelle construction basée sur les couplages dont nous prouvons la sécurité dans ce modèle. Nos derniers travaux portent sur les systèmes de multi-coupons et de monnaie électronique. L'utilisation des couplages nous permet d'introduire de nouvelles propriétés destinées à faciliter leur usage. Pour chacun de ces systèmes nous proposons un modèle de sécurité, puis décrivons un schéma dont nous prouvons la sécurité dans ce modèle.

English title. Signatures for pairing-based anonymity and applications.

English abstract. Anonymity issues arise in several situations, in the Internet context and specifically for electronic transactions. It is then advisable to protect the users' identities to avoid the construction of commercial information databases or the constitution of consumers' profiles. Many cryptographic solutions have been proposed to strengthen the users' confidence in these systems. A new approach in the elaboration of secure and efficient anonymous mechanisms is based on bilinear maps (Weil and Tate's pairings on elliptic curves).

In this thesis, we first present a literature review of different signatures used for anonymity in cryptography, specially group signatures, blind signatures and ring signatures. Following this presentation, we describe a new protocol for authentication and show how it can be turned into a ring signature. We then focus our study on fair blind signatures. These are blind signatures enabling an authority to revoke their anonymity and untraceability. We propose the first actual security model for these signatures and a new construction based on bilinear maps, we then prove its security in this model. Our final works concern multi-coupon and electronic cash systems. The use of pairings allows us to introduce new properties aimed to simplify their utilization. For each of these systems we describe a security model, then describe a scheme and prove its security in this model.

Mots-clés. Cryptographie à clé publique, signatures électroniques, authentification, anonymat, monnaie électronique.

Discipline. Informatique.

Orange Labs R&D, 42 rue des coutures 14000 CAEN.
Laboratoire GREYC, CNRS UMR 6072, UFR des Sciences - Campus II Côte de Nacre Boulevard du Maréchal Juin, 14000 Caen