École Normale Supérieure **Université Paris 7**

Département d'Informatique
Groupe de Recherche En Complexité et Cryptographie

Le partage de clés cryptographiques : Théorie et Pratique

THÈSE

présentée et soutenue publiquement le 4 Octobre 2001

pour l'obtention du

Doctorat de Paris 7

par

Pierre-Alain Fouque

Composition du jury

Rapporteurs :	Rosario Gennaro	(I.B.M. T.J. Watson)
	Marc Girault	(France Télécom R&D)
	Moti Yung	(Université de Columbia)
Examinateurs :	Anca Musholl	(LIAFA - Paris 7)
	Jacques Patarin	(Schlumberger - Université de Versailles)
	David Pointcheval	(ENS - CNRS)
	Jacques Stern	(ENS - Directeur de Thèse)

Département d'Informatique de l'École Normale Supérieure, 45 Rue d'Ulm 75005 PARIS

Remerciements

Je remercie tout d'abord vivement Jacques Stern pour son encadrement scientifique rigoureux, pour la chance qu'il m'a offerte de faire de la recherche dans d'aussi bonnes conditions et pour son amitié. Je remercie ensuite chaleureusement tous les membres du GRECC et plus particulièrement Guillaume et David pour leurs nombreuses explications, leurs grandes disponibilités, les travaux effectués ensemble, les moments passés dans le bureau ou ailleurs (piscines, ...), et finalement le travail de relecture.
L'équipe de Complexité et Cryptographie de l'ENS, son directeur Jacques Stern, ses chercheurs permanents David Pointcheval, Louis Granboulan, Phong Nguyen, ses anciens membres Serge Vaudenay, ses nombreux thésards que j'ai pu cotoyer Guillaume Poupard, Thomas Pornin, Olivier Baudron, Emmanuel Bresson, et ses stagiaires Yves Verhoeven, ... sont un milieu idéal pour faire de la recherche en Cryptographie.

Je suis aussi redevable à l'entreprise CS Communication & Systèmes puis à sa filiale TrustyCom de m'avoir permis de mettre en pratique la cryptographie et de travailler sur différents projets et produits intéressants comme les PKIs, la signature électronique ou la sécurisation des protocoles réseaux. Je remercie toutes les personnes avec qui j'ai eu le plaisir de travailler Sylvain Blonde, Jérôme Lubrez, Jean-Pierre Garnier, Moïse Moyal, Caroline Gerrebout, Pierre Tsagouria, Oualid Ammar, Jean-François Wiorek, Mustapha Allaf, Patrick Dessarps, Didier Virlogeux, Véronique Delebarre, Widad Chatraoui, Julien Olivain, Jean-Pierre Gauthier, ceux que j'oublie. Je remercie en particulier Marc Milan, le "mentor" de l'activité sécurité pendant de longues années.

J'exprime ma gratitude à Rosario Gennaro, Marc Girault et Moti Yung pour avoir effectué le lourd travail de rapporteurs et je remercie Anca Muscholl et Jacques Patarin d'avoir accepté de participer à mon jury.

Merci à Joëlle et Valérie pour leur soutien quotidien.

Enfin, j'exprime mes remerciements les plus chers à Gwenaëlle pour son attention, ses nombreux encouragements et conseils et pour le bonheur et la joie qu'elle me procure.

À mes parents, Catherine et Anne, et Gwenaëlle,

Table des matières

Table des figures

Notations

Les notations suivantes seront employées dans la suite de cette thèse :

- Nous noterons $\lfloor x \rfloor$ la partie entière inférieure de x et $\lceil x \rceil$ la partie entière supérieure de x.
- La taille en bits de tout entier x sera notée $|x|$ ou $L(x)$, et est égale à $\lfloor \log_2(x) \rfloor + 1$.
- Nous utiliserons les notations classiques de Landau pour comparer asymptotiquement les fonctions : $o(.)$, $\mathcal{O}(.)$ et $\Theta(.)$. En particulier, soient f et g des fonctions de $\mathbb{N}$ dans $\mathbb{N}$, on a :
  - $f(n) = \mathcal{O}(g(n))$, s'il existe une constante $c > 0$ et un entier $n_0 > 0$, tels que $f(n) \leq c.g(n)$ pour tout $n > n_0$.
  - $f(n) = \Omega(g(n))$, s'il existe une constante $c > 0$ et un entier $n_0 > 0$, tels que $f(n) \geq c.g(n)$ pour tout $n > n_0$.
  - $f(n) = \Theta(g(n))$, s'il existe deux constantes $c_1 > 0$ et $c_2 > 0$ et un entier $n_0 > 0$, tels que $c_1.g(n) \leq f(n) \leq c_2.g(n)$ pour tout $n > n_0$.
  - $f(n) = g(n)^{\mathcal{O}(1)}$, s'il existe une constante $c > 0$ et un entier $n_0 > 0$, tels que $f(n) \leq g(n)^c$ pour tout $n > n_0$.
  - $f(n) = g(n)^{\Omega(1)}$, s'il existe une constante $c > 0$ et un entier $n_0 > 0$, tels que $f(n) \geq g(n)^c$ pour tout $n > n_0$.
  - $f(n) = o(g(n))$, si pour toute constante $c > 0$, il existe un entier $n_c > 0$, tel que $0 \leq f(n) < c.g(n)$ pour tout $n > n_c$.
  - $f(n) = \omega(g(n))$, si pour toute constante $c > 0$, il existe un entier $n_c > 0$, tel que $0 \leq c.g(n) < f(n)$ pour tout $n > n_c$. En d'autres termes, $f(n) = \omega(g(n))$ si et seulement si $g(n) = o(f(n))$.

  Intuitivement, $f(n) = \mathcal{O}(g(n))$ signifie que f ne croît pas asymptotiquement plus vite que g à une constante multiplicative près ; $f(n) = \Omega(g(n))$ signifie que f croît au moins aussi rapidement que g à une constante multiplicative près ; $f(n) = o(g(n))$ signifie que g est une borne supérieure pour f qui n'est pas asymptotiquement fine, en d'autres mots, la fonction f devient insignifiante par rapport à g quand n croît, ou $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. L'expression $f(n) = o(1)$ est souvent utilisée pour dire qu'une fonction f tend vers 0 quand n tend vers l'infini.
- La notation $x \in_R X$ indique que l'élément x est choisi au hasard dans un ensemble fini X, selon une distribution uniforme.
- Les intervalles entiers seront notés entre crochets, $[0, X[$ désignant par exemple l'ensemble des entiers $\{0, 1, \ldots, X-1\}$.

– Le pgcd de deux entiers x et y sera noté soit $\mathrm{pgcd}(x, y)$, soit (x, y).

– On désignera souvent un nombre premier, *i.e.* divisible uniquement par 1 et par lui-même, par p ou q, et par N un nombre composé (non premier) de la forme pq.

– $\mathbb{Z}_N$ est l'anneau défini sur les éléments $[0, N[$ muni des lois d'addition et de multiplication modulo N.

– $\mathbb{Z}_N^*$ représente le groupe des éléments inversibles de $\mathbb{Z}_N$, i.e., l'ensemble des éléments x tels que $\mathrm{pgcd}(x, N) = 1$.

– L'ordre multiplicatif d'un élément g dans le groupe $\mathbb{Z}_N^*$ est noté $\mathrm{ord}_N(g)$.

– Q_N représente le groupe des carrés dans $\mathbb{Z}_N^*$.

– $\varphi(N)$ représente la fonction d'Euler, i.e. la cardinalité de $\mathbb{Z}_N^*$.

– $\lambda(N)$ représente la fonction lambda de Carmichael définie comme étant le plus grand ordre des éléments de $\mathbb{Z}_N^*$. Si $N = \prod_i {p_i}^{e_i}$, où pour tout i, $p_i > 2$, alors $\lambda(N) = \mathrm{ppcm}({p_i}^{e_i - 1}(p_i - 1))$

– $(x|p)$ est le symbole de Legendre de x modulo un nombre premier p. On peut le calculer par la formule $x^{\frac{p-1}{2}} \mod p$. Il vaut 0 en 0, 1 si x est un carré modulo p (résidu quadratique), et -1 sinon. De la même manière, $(x|N)$ est le symbole de Jacobi lorsque N est composé.

– L'abréviation PPTM(t) désigne une machine de Turing fonctionnant en temps polynomial en t et disposant d'un ruban spécial, infini, contenant des bits choisis au hasard selon une distribution uniforme.

– n représente le nombre de serveurs ou d'acteurs dans un protocole partagé et t représente le "seuil" de serveurs qui peuvent être corrompus.

– Les probabilités sont notées $\Pr_{x \in D}[X]$ et désignent la probabilité de l'événement X prise pour $x \in D$.

– Le sous-groupe généré par les éléments $g_1, \ldots g_k$ d'un groupe $\mathcal{G}$ est noté $\langle g_1, \ldots g_k \rangle$.

Introduction

La cryptographie a pour but d'assurer la sécurité des communications et des données stockées en présence d'un adversaire. Elle propose un ensemble de techniques permettant d'offrir des services de confidentialité, d'authentification et d'intégrité. La cryptologie, appelée aussi la Science du Secret dans [179], regroupe la *cryptographie* et la *cryptanalyse*. Alors que le rôle des cryptographes est de construire et prouver, entre autres, des systèmes de chiffrement ou de signature, l'objectif des cryptanalystes est de "casser" ces systèmes. L'histoire de la cryptologie a vu tour à tour les victoires des uns et des autres (cf. [112, 177, 179]).

L'histoire de la cryptographie a été pendant longtemps l'histoire des *codes secrets* et depuis l'antiquité, ils ont décidé du sort des hommes et des nations. En effet, jusque dans les années 70, l'unique objectif de la cryptographie était de construire des systèmes de chiffrement. Grâce à la cryptanalyse, les militaires et les cabinets noirs des diplomates ont pu mener leurs guerres dans l'ombre en découvrant les correspondances de leurs ennemis et en contrôlant les réseaux de communications. Les codes sauvèrent les Grecs des Perses, accompagnèrent César dans ses conquêtes, firent arrêter et décapiter Marie Stuart, décidèrent Wilson à rejoindre les alliés, et permirent d'épargner des milliers de vie pendant la seconde guerre mondiale.

La révolution d'Internet et l'utilisation de plus en plus massive d'informations sous forme numérique facilitent les communications et rendent de ce fait plus fragiles les informations que l'on détient. En effet, les réseaux "ouverts" créent des brèches de sécurité et il est plus aisé à un adversaire d'accéder aux informations. De même, le remplacement de l'homme par des machines rend les relations beaucoup plus anonymes alors qu'en même temps l'accès aux données demande des moyens d'authentification forts. De plus, la dématérialisation change les moyens de preuves juridiques : les signatures numériques, ou plus généralement les preuves de l'*écrit électronique* doivent remplir certaines exigences que nous ne connaissions pas avec les signatures manuscrites. Ainsi, la révolution numérique des communications et de l'information a ouvert de nombreux champs d'investigation à la cryptographie, de sorte que celle-ci a envahi notre vie quotidienne : carte à puce, transaction bancaire, internet, téléphone cellulaire...

Commençons par voir quels sont les services de sécurité que peut garantir la cryptographie et ses applications dans la vie "réelle".

1 Objectifs de sécurité

La cryptographie ne permet pas de résoudre tous les problèmes de sécurité informatique. Cependant, elle apporte des garanties et des briques de base sur lesquelles des produits de sécurité peuvent être construits. Il est bien connu que la sécurité d'un système de sécurité se mesure à son maillon le plus faible. En général, le maillon le plus faible d'un système de sécurité informatique n'est pas le système cryptographique mais par exemple son implémentation informatique.

En outre, il existe diverses attaques contre lesquelles la cryptographie n'est pas d'un grand secours, comme les virus informatiques ou les chevaux de Troie logiciels qui profitent de la confiance exagérée des utilisateurs dans les messages ou logiciels qu'ils reçoivent.

La cryptographie participe à la sécurité informatique en proposant des primitives qui permettent d'atteindre les objectifs d'authentification, de confidentialité et de protection en intégrité. Au-delà de cette trilogie, la cryptographie apporte aussi certaines réponses à des problématiques de *disponibilité de service* et de *résistance aux fautes* sur certains protocoles cryptographiques. Ce dernier objectif de sécurité est au cœur de la présente thèse.

2 Authentification

L'authentification d'un écrit est un problème ancien qui a trouvé des réponses dans les analyses graphologiques et d'études des divers types de support. La cryptographie apporte des techniques permettant d'authentifier l'émetteur d'un document ou d'un écrit électronique [1].

En cryptographie, l'authentification regroupe l'*authentification de données* qui permet de garantir que les données transmises proviennent bien d'un émetteur désigné, et l'*identification de personne* qui consiste à prouver son identité.

C'est par exemple le cas lors d'un retrait d'argent. Lorsque vous mettez votre carte à puce dans le distributeur ou un terminal de paiement électronique, un protocole d'authentification vise à vérifier qu'il s'agit bien d'une carte bancaire. Pour ce faire, une signature basée sur des informations contenues dans la carte est vérifiée. Une signature est une donnée publiquement vérifiable permettant d'identifier sans ambiguïté le signataire. Les informations de la carte sont signées avec la clé GIE CB [2]. Une vérification correcte de cette signature prouve que la carte est bien une carte bancaire car seul le GIE CB peut générer de telles signatures. Puis, une identification permet d'assurer que l'utilisateur est bien le détenteur légitime de la carte. Pour ce faire, le possesseur connaît un code personnel d'identification, appelé code PIN [3], composé de 4 chiffres. Lorsqu'il tape son code PIN sur le clavier, le code est envoyé à la carte qui en possède un double. La puce effectue une comparaison entre le code envoyé et celui sauvegardé et enfin répond par un bit d'information indiquant si le code est bon ou mauvais. Ce faisant, le système "carte bleue" a identifié la carte et le fait que le porteur de la carte est bien son détenteur légal.

Il existe d'autres systèmes d'identification permettant par exemple l'entrée sur un système informatique ou l'accès à certaines informations sur un site web. Dans ce cas, l'utilisateur rentre son nom et son mot de passe. En tapant son login, l'utilisateur déclare une identité. Le système acceptera l'utilisateur si ce dernier peut fournir une preuve de cette identité en révélant par exemple un mot de passe correct. Le mot de passe sera alors comparé à celui conservé par le système soit de manière centralisée, soit de manière locale comme dans la carte à puce dans le paragraphe précédent. Lors d'une authentification centralisée, un adversaire espionnant le réseau peut intercepter le login et le mot de passe et se faire passer pour cette personne. Ultérieurement, afin de protéger l'identification d'un utilisateur, la cryptographie a apporté une réponse originale en proposant le concept de preuve d'identité "zero knowledge". Il s'agit de mécanismes qui permettent à une personne de prouver son identité sans avoir à divulger son secret. On peut prouver que le système qui authentifie ne peut apprendre aucune information sur le secret qui représente la personne lors d'une instance de ce protocole. Ceci est fondamental dans de nombreux protocoles cryptographiques. On peut dire que la cryptographie sépare le pouvoir de conviction

1. L'écrit électronique est une notion juridique. Il apparaît d'après certains juristes, comme le Professeur Raynouard, que la législation française en matière de signature électronique aurait mieux fait de proposer une législation sur l'écrit électronique plutôt que de faire une particularisation du droit de la preuve juridique à la signature électronique. Pour un aperçu pour la profession notariale, lire l'article "Les technologies de l'écrit électronique" [70].

2. Groupement d'Intérêt Économique de la Carte Bancaire

3. Personal Identification Number

du pouvoir d'explication[4]. C'est-à-dire, on peut convaincre un interlocuteur que l'on connaît une preuve d'un théorème sans avoir besoin de donner la démonstration ! La conséquence de cette technique est de briser la chaîne de transmission d'une preuve. La personne convaincue est incapable de convaincre une autre personne du théorème mathématique. C'est exactement le type de propriété que l'on souhaite pour construire un système d'identification.

3 Intégrité

Le problème de l'intégrité est également un problème ancien qui s'est accentué avec l'émergence de l'écrit sous forme électronique. En effet, la copie, la modification puis la diffusion du document sont des opérations facilement réalisables. De même, l'intégrité de certaines données de sécurité est sensible. C'est par exemple, le cas du mot de passe qui doit être stocké sur la carte à puce ou dans l'ordinateur. Si ce mot de passe venait à être modifié, la personne ne pourrait plus s'authentifier sur le système. En outre, lorsque l'on télécharge sur Internet un document, on veut être certain que le document reçu correspond bien à celui que le site web présentait. De manière plus générale, le problème de l'intégrité peut apparaître dans le cas du partage de ressources et dans le cas de l'envoi de messages sur un canal non sécurisé.

Pour résoudre ce problème, on utilise des fonctions de hachage qui réduisent des données de longueur variable en une "empreinte" de petite taille, appelé *haché* ou *condensat*. Cette empreinte peut être stockée sur un autre support que l'ordinateur par exemple sur papier, et permet à l'utilisateur à chaque fois qu'il utilise sa machine de vérifier si une personne a modifié les données ou de vérifier que l'information envoyée n'a pas été altérée en route. Les fonctions de hachage à usage cryptographique doivent satisfaire les propriétés suivantes :

1. Fonction à sens unique : étant donné y, il est difficile de trouver un x tel que $H(x) = y$ en un temps raisonnable. Cette propriété permet d'empêcher une personne malintentionnée de modifier le contenu du disque dur ou d'un fichier envoyé.

2. Fonction résistante aux collisions : il est difficile de trouver x et x' $(x \neq x')$ tels que $H(x) = H(x')$ en temps raisonnable. Cette propriété permet d'empêcher l'émetteur de créer des fichiers ayant même "empreinte". Un émetteur pourrait renier le premier document en disant qu'il a envoyé le second. Cette notion est aussi importante dans le modèle *hash-and-sign* qui consiste à signer uniquement une empreinte du fichier. Ceci permet en outre de construire des schémas de signature qui prennent leur entrée dans un espace fixe et permettent de signer des documents de taille variable.

3. Fonction résistante à un deuxième antécédent : étant donné $y = H(x)$, il est difficile de trouver en temps raisonnable $x' \neq x$ tel que $H(x') = y$. Dans le modèle de signature *hash-and-sign*, ceci empêche un attaquant de trouver un deuxième message ayant même signature qu'un message signé donné.

Les exemples de fonctions de hachage sont MD5 (Message Digest numéro 5) et SHA-1 (deuxième version de la norme américaine Secure Hash Algorithm du NIST[5]). Dobbertin [63, 59] a trouvé des faiblesses sur la fonction de compression de MD5 et aujourd'hui seule la fonction SHA-1 est recommandée.

4. cf. ENS, Le Courrier, Numéro 46, Février 1999, Extrait d'un entretien entre Jacques Stern et Max Marcuzzi.
5. National Institute of Standards and Technology

4 Confidentialité

La confidentialité est le problème le plus ancien auquel la cryptographie ait tenté de trouver une réponse. Le problème à résoudre est le suivant : deux personnes veulent communiquer et utilisent pour cela un canal public qui peut être espionné par un adversaire. La solution consiste à transformer les données de sorte à les rendre illisibles à l'adversaire tout en assurant que le destinataire pourra appliquer la transformation inverse pour retrouver les données originelles.

Le *chiffrement* est l'opération qui consiste à transformer un message à transmettre, dit "message clair", en un autre message inintelligible pour un tiers, dit "message chiffré", en vue d'assurer le secret de la transmission. Le *déchiffrement* est l'opération inverse du chiffrement et consiste en une transformation des données chiffrées en une forme intelligible. Chiffrement et déchiffrement exigent généralement l'utilisation d'une information secrète appelée la *clé*.

On distingue le chiffrement symétrique qui utilise la même clé pour chiffrer et déchiffrer et le chiffrement asymétrique ou à clé publique où les clés utilisées sont différentes mais reliées entre elles par une relation mathématique. Dans ce dernier système de chiffrement, la clé de chiffrement est dite "publique" et peut être connue de tout le monde, alors que la clé "privée" doit être maintenue de manière sûre par son possesseur. Dans ce modèle, toute personne qui a accès à la clé publique d'un utilisateur peut lui envoyer des messages, alors que ce dernier sera le seul à pouvoir les déchiffrer. On peut remarquer que la sécurité d'un système ne provient pas de la difficulté à chiffrer, n'importe qui doit pouvoir chiffrer, mais de la *difficulté à déchiffrer*. C'est cette asymétrie qui est au cœur du concept de cryptographie à clé publique [62].

L'inconvénient des cryptosystèmes symétriques est que les personnes doivent se rencontrer pour échanger la clé de chiffrement avant de pouvoir l'utiliser. Dans une communauté fermée d'utilisateurs, ceci peut être réalisé, mais dans le cas d'Internet, où les personnes ne se connaissent pas forcément avant d'établir une communication et souhaitent tout de même pouvoir échanger des informations confidentiellement, ce type de système n'est pas adapté. De plus, chaque utilisateur doit posséder une clé avec toutes les personnes avec qui il souhaite communiquer. On voit que la cryptographie à clé publique permet de résoudre ces problèmes car les clés publiques peuvent être publiées dans des annuaires informatisés. Ce sont l'équivalent d'annuaires téléphoniques qui associent une clé publique à chaque personne, au lieu d'un numéro de téléphone. Les systèmes à clés publiques résolvent le problème de l'échange de clé en utilisant des annuaires publiquement consultables. Historiquement, c'est à partir du problème crucial de la distribution de clé de chiffrement symétrique qu'a été découvert le concept de cryptographie à clé publique (cf.[62, 177]). Cependant, la cryptographie à clé publique souffre d'un défaut majeur quand la quantité d'informations à transmettre devient grand. En effet, ces systèmes sont plus lents que les cryptosystèmes symétriques et sont plus gourmands en temps de calcul, ce qui pénalise les performances quand beaucoup d'opérations de chiffrement et déchiffrement doivent être effectuées (comme sur des serveurs). Enfin, un système "hybride" tire profit de la cryptographie à clé publique en échangeant une clé de session de petite taille, 128 bits par exemple, et de la cryptographie symétrique pour chiffrer les données à transmettre en utilisant la clé de session. Les systèmes hybrides résolvent le problème de la "lenteur" de la cryptographie à clé publique pour chiffrer de longs messages et de l'échange de clé pour les système de chiffrement symétrique.

Comme exemples de cryptosystèmes symétriques, on peut citer le DES (Data Encryption Standard) qui depuis 1977 est l'algorithme le plus utilisé dans le monde. En octobre 2000, le NIST a proposé un successeur au DES. Le DES a résisté pendant 20 ans aux assauts de la recherche publique. Cependant la taille des clés DES fixée à 56 bits a permis à la recherche exhaustive et à la conception de puce de venir à bout de l'espace de recherche. L'AES (Advanced Encryption Standard), alias Rijndael, est le remplaçant du DES. Il s'agit d'un système de chiffrement symétrique utilisant des blocs de 128 bits et pouvant utiliser des clés de 128, 192, et 256 bits. Ayant participé à de nombreuses opérations autour de l'algorithme CS-

Cipher (CS-Cipher challenge [6], soumission NESSIE [7], implémentation de l'algorithme en FPGA dans un commutateur ATM [8] de la société CS Télécom, implémentation en C++ dans les produits de la société CS Systèmes d'Information), je ne peux manquer de parler de cet algorithme développé par la société CS Communication et Systèmes en collaboration avec Jacques Stern et Serge Vaudenay [180, 183].

Enfin, parmi les exemples de cryptosystèmes à clé publique, on peut citer le RSA qui fut le premier système à clé publique, proposé par Rivest, Shamir et Adleman en 1977 [163], et les systèmes El Gamal [66], Paillier [139], NTRU (basé sur le problème de trouver des vecteurs courts dans certains réseaux), ceux basés sur le problème du sac-à-dos et enfin celui de Mac Eliece (problèmes de codes correcteurs d'erreurs).

5 Disponibilité de service

La disponibilité de service est souvent oubliée dans les avantages que peut offrir la cryptographie. En effet, seules les applications cryptographiques peuvent profiter de cette fonction de sécurité. Cependant, ce service est crucial pour certaines applications pratiques qui exigent un fort niveau de sécurité et qui doivent tourner en permanence. En effet, l'indisponibilité d'un service comme le réseau de carte bancaire ou l'accès à des sites web peut avoir des conséquences économiques importantes.

La cryptographie apporte une solution qui ne permet pas de répondre entièrement à la problématique du déni de service qui est actuellement le problème le plus épineux sur l'Internet avec l'invasion des virus. Le déni de service a fait tomber de nombreux sites commerciaux comme Amazone.com ou E-Bay.com en février 2000. Ces attaques de déni de service distribuées consistent à inonder de demandes un site web en envoyant plus de demandes de connexions que ne peut en servir le site. Les conséquences économiques ont été énormes pour ces sites. Cependant, il n'existe aujourd'hui pas de solution satisfaisante pour résister à ce type d'attaques.

La solution cryptographique ne résout pas le problème du déni de service mais résout partiellement le problème de la *tolérance aux pannes*. Elle garantit que si parmi n serveurs, seule une minorité $t < \frac{n}{2}$ est attaquée, alors le système cryptographique (de déchiffrement ou de signature) sera toujours opérationnel. Ces solutions de redondance sont importantes dans les systèmes très sensibles comme par exemple la clé de signature d'une autorité de certification ou la clé de déchiffrement d'une autorité de recouvrement. Imaginons donc un service de recouvrement qui consiste à retrouver la clé privée d'un utilisateur. Ces clés sont conservées sous forme chiffrée dans une base de données. La clé de déchiffrement de l'autorité est sensible car elle permet de retrouver toutes les clés privées des utilisateurs. Par conséquent, si la clé était confiée entièrement à un employé malintentionné, ce dernier pourrait déchiffrer la clé de tous les utilisateurs et lire les messages de tout le monde. Afin, d'éviter une telle tentation, la clé est séparée en plusieurs parts de sorte que l'intervention d'au moins la majorité des parts soit nécessaire pour déchiffrer la clé d'un utilisateur. Ce genre de partage est aussi utilisé dans de nombreuses banques où l'intervention du directeur de la banque avec un des deux sous-directeurs par exemple est nécessaire. La cryptographie force alors l'intervention de plusieurs personnes en faisant comme hypothèse qu'il n'y aura pas trop de personnes malintentionnées qui se coalisent pour attaquer le système.

De même, des serveurs en panne peuvent être vus comme des personnes absentes et tant qu'il y a une majorité de serveurs en marche, le service de recouvrement par exemple peut être assuré. Ce type de partage a été implémenté dans le produit de recouvrement de clé de la société CS Systèmes d'Information.

Cette thèse a pout but de proposer différentes techniques qui visent à garantir cet objectif pour divers

6. `http://www.distributed.net/`
7. New European Schemes for Signatures, Integrity and Encryption, `http://www.cryptonessie.org/`
8. Asynchronous Tranfer Mode

types de systèmes de chiffrement ou de signature. Nous montrerons dans la troisième partie des applications de ces techniques à un schéma de loterie électronique, de vote électronique et au recouvrement de clé. Toutes ces applications requièrent un fort niveau de sécurité sur l'opération de déchiffrement ou de signature.

6 La cryptographie et la vie réelle

Les applications de la cryptographie sont la sécurisation des postes de travail, des connexions à des serveurs (identification des accès et transmission sécurisée des données), des canaux publics et enfin des communications par messagerie.

La protection des connexions du type client/serveur, telles que le protocole HTTP (HyperText Transfer Protocol) par exemple, utilise le protocole de sécurité SSL (Secure Socket Layer) ou TLS (Transport Layer Security). La sécurisation des liens de communication sur un canal public utilise le protocole IPSec (IP Security) qui permet de chiffrer tout protocole véhiculé sur IP (Internet Protocol), cas de la plupart des flux aujourd'hui. Enfin, les produits de signature et de chiffrement de fichiers ou de messages utilisent le protocole CMS (Cryptographic Message Syntax) et le protocole S/MIME (Secure / Multipurpose Internet Mail Extensions) dans le cas des messageries.

Pendant longtemps la cryptographie était principalement utilisée par les militaires qui avaient recours exclusivement à la cryptographie symétrique. De même, les applications centralisées, comme les réseaux téléphoniques des opérateurs de télécommunication (cf. la téléphonie mobile), ont opté pour des systèmes symétriques plus rapides une fois implémenté en matériel.

Comme on l'a vu, la révolution de l'électronique utilise les avantages que peut procurer la cryptographie à clé publique pour signer ou communiquer avec une personne sans avoir besoin d'établir un contact préliminaire pour échanger une clé de session. Ainsi, la cryptographie à clé publique devient une nécessité. Le principal frein au développement d'un système asymétrique est qu'il faut construire une infrastructure à clé publique permettant de distribuer et de gérer les clés des utilisateurs. En effet, à l'instar du système bancaire pour lequel un mécanisme permet de faire opposition en cas de perte ou de vol de la carte, dans le cas de la cryptographie asymétrique, un mécanisme similaire permet de révoquer en cas de perte de la clé privée. Le problème principal de la cryptographie à clé publique est que rien ne permet de dire quand on voit une clé publique si cette clé appartient bien à une personne donnée. Ce problème a été résolu par l'utilisation des certificats de clé publique. Les clés sont encapsulées dans des *certificats numériques* qui sont l'équivalent d'un passeport numérique. Ils attestent que la clé publique qu'une personne présente lui a bien été délivrée par une autorité à laquelle les deux parties qui souhaitent communiquer font confiance. Cette autorité fonctionne comme une préfecture : elle commence par enregistrer des pièces prouvant l'identité d'un individu et génère ensuite un certificat en signant numériquement la clé publique et l'identité de la personne de la même manière que l'administration met un tampon sur une photo et le nom de la personne.

La mise en place de ces autorités, dites *autorités de certification*, représente le principal inconvénient à la technologie de clé publique. Aujourd'hui de nombreuses infrastructures à clé publique (ICP, en anglais PKI) sont déployées aussi bien dans des entreprises que des administrations. En Finlande, une expérience est menée pour délivrer à tous les habitants une carte à puce avec un certificat de clé publique. Cette carte sera la carte d'identité de chaque citoyen. De même, en France de nombreux projets sont en cours pour permettre aux différentes professions du monde de la santé (médecins, pharmaciens, infirmières, ...) d'accéder à des informations en toute sécurité.

Aujourd'hui le marché de la cryptographie concerne surtout les grands comptes qui sont plus sensibilisés que les petites entreprises au problème de la sécurité. Pour ces dernières, la sécurité est perçue à travers le problème des virus. Cependant, la législation sur la signature électronique est effective en France

et de nombreuses lois comme la télé-déclaration de TVA forcent les entreprises françaises réalisant un chiffre d'affaire supérieur à 100 millions de francs (15,6 millions d'euros) à signer numériquement leur télé-déclaration et leur télé-paiement. Espérons pour les sociétés éditrices de logiciels de sécurité que cela permettra le décollage des produits de sécurité !

Première partie

Introduction à la cryptologie partagée

1

Généralités de cryptographie moderne

La cryptographie est sortie des âges artisanal et technique (cf. [179]) pour entrer dans l'âge moderne au milieu des années 70. La cryptographie moderne est aujourd'hui une discipline de l'Informatique.

Dans ce chapitre, nous présentons l'approche moderne de la cryptographie, puis nous introduirons le décor théorique, et enfin, nous décrivons les modèles de sécurité pour les primitives standards que sont le chiffrement et la signature.

Sommaire

1.1 Introduction à la cryptographie moderne

Un des objectifs majeurs de la cryptographie moderne est de construire des schémas *prouvés sûrs* qui peuvent être utilisés en *pratique*. Initialement, une tâche de la cryptographie a été de démontrer des résultats d'*existence* assurant qu'il est possible de concevoir de telles primitives (sûres et pratiques). Aujourd'hui, les cryptographes cherchent à construire ces primitives de manière efficace en utilisant des

hypothèses basées sur la difficulté pour une machine à effectuer certains calculs ou à résoudre certains problèmes.

Un des objectifs de la cryptographie moderne est donc de concevoir des primitives de sécurité comme des schémas de signature et de chiffrement efficaces, et si possible prouver leur sécurité. Cette mission peut être divisée en trois tâches :

- *définir* des notions appropriées de sécurité. Ceci comprend la description d'un modèle formel décrivant comment l'adversaire interagit avec le système et ce que signifie "casser" un système.
- *concevoir* des schémas cryptographiques.
- *prouver* la sécurité de ces schémas cryptographiques dans le modèle précédemment défini.

La première tâche est aujourd'hui en partie résolue pour les primitives de base de la cryptographie que sont les cryptosystèmes à clé publique ou les schémas de signature. Cependant, on ne peut prouver la sécurité qu'en fonction des objectifs et de la description de l'adversaire. Si toutes les attaques n'ont pas été prévues, il se peut que le système soit démontré sûr selon la modélisation d'un type d'attaquant, mais que ce système soit "cassable" car dans la vie réelle ce dernier peut monter des attaques qui n'auraient pas été envisagées dans le modèle initial. C'est par exemple le cas si l'on considère que l'adversaire peut avoir connaissance d'une partie de la clé ou que l'équipement matériel ne soit pas aussi résistant que ce que l'on avait prévu. Ces nouveaux types d'attaques ont été récemment considérés dans [32, 116, 117].

Les termes de "sécurité prouvée" signifient que le but final est de montrer que le schéma ne peut pas être cassé. Alors que cette troisième tâche peut être atteinte pour certains problèmes cryptographiques, les solutions ne sont généralement pas pratiques et requièrent de faire appel à un ensemble d'hypothèses physiques spéciales. On est dans le modèle de la cryptographie sûre au sens de la théorie de l'information et les cryptographes parlent de sécurité inconditionnelle. Cette notion remonte à Shannon [171, 172] qui a démontré la sécurité inconditionnelle du système de chiffrement de Vernam ou "one-time pad". Nous montrerons plus loin que le schéma de partage de secret de Shamir [170] a aussi cette propriété.

L'étape de conception de systèmes pratiques consiste à prouver la sécurité en montrant qu'un schéma ne peut pas être cassé sans l'aide d'une quantité importante de ressources de calcul. Malheureusement, étant donné l'état des connaissances mathématiques, on ne peut pas espérer prouver la sécurité d'un schéma dans ce sens. En revanche, quand les cryptographes parlent de "schémas prouvés sûrs", ils parlent de sécurité conditionnelle, basée sur des problèmes difficiles "raisonnables et naturelles" comme par exemple le problème de la factorisation de grands nombres et utilisent l'état de l'art des meilleurs algorithmes connus qui résolvent ces problèmes difficiles pour estimer les paramètres de sécurité de leurs schémas.

Même si la sécurité prouvée conditionnellement à des hypothèses n'est pas une notion aussi forte que ce que l'on souhaiterait, elle demeure une notion puissante. Elle garantit qu'il ne peut pas y avoir d'alternative pour casser le système cryptographique, *i.e.* un adversaire qui tenterait de casser le système doit attaquer les problèmes sous-jacents difficiles. En effet, il existe des exemples célèbres où des attaques ont été trouvées pour casser le schéma (cf. [20, 52]) sans attaquer le problème sur lequel était supposé reposer la sécurité. La sécurité *ad hoc*, c'est-à-dire basée sur une solution sans preuve de sécurité et résistante tant que le schéma n'a pas été cassé, n'est plus un gage de sécurité aujourd'hui.

Pour définir ces notions de sécurité conditionnelle, nous avons besoin de faire quelques rappels de complexité.

1.2 Rappels de complexité

La Théorie de la Complexité ou Complexité des Calculs est un domaine central de l'Informatique. Cette discipline a pour objectif d'étudier la complexité *intrinsèque* de certains calculs et problèmes et

cherche à ranger les problèmes en différentes classes : elle s'intéresse aux ressources naturelles pour effectuer le calcul et considère les effets de la limitation de ces ressources sur la classe de problèmes qui peuvent être résolus. *Analyser* un algorithme signifie prévoir les ressources nécessaires à cet algorithme. Les ressources pertinentes sont la mémoire utilisée, la bande passante d'une communication, le nombre de bits aléatoires ou le nombre de portes logiques, mais le plus souvent on s'intéresse uniquement au temps de calcul.

1.2.1 Problèmes et langages

On peut définir un *problème abstrait* Q comme une relation binaire entre un ensemble I d'instances d'un problème et un ensemble S de solutions à un problème. Soient par exemple $G = (S, A)$ un graphe, donné par un ensemble S de sommets et un ensemble A d'arêtes, deux sommets u et v de G, et une longueur k. Le problème abstrait CHEMIN est le suivant : existe-t-il un chemin dans G de u vers v de longueur au plus k? Une instance de ce problème est la donnée explicite d'un graphe, de deux sommets, et d'une longueur.

Les problèmes susceptibles d'un traitement par une machine se répartissent en deux classes :

1. La classe des problèmes conduisant à une valeur (numérique ou non); ils sont modélisés par la notion de *fonction* de A^* dans B^*, où A et B sont des alphabets, ou plus généralement de $(A^*)^n \rightarrow B^*$.

2. La classe des problèmes conduisant à une réponse "oui" ou "non"; ce sont les problèmes de décision. Ils sont modélisés par la notion de langage: à tout *langage* L sur un alphabet A est associé le problème de décision consistant à déterminer si un mot w de A^* appartient ou non à L, où A^* représente l'ensemble des suites de lettres de A.

Définition 1 ***(Langage)*** *Soit $L : \{0,1\}^k \rightarrow \{0,1\}$ une fonctions. On peut voir L comme un langage, où, pour tout $w \in \{0,1\}^k$, $w \in L$ si $L(w) = 1$ et $w \notin L$ si $L(w) = 0$.*

Le problème CHEMIN est un problème décisionnel qui peut être traduit par un langage. En effet, on voit que l'ensemble des solutions est $S = \{0,1\} = \{\text{non}, \text{oui}\}$.

1.2.2 Machines de Turing et algorithmes

Deux modèles de calcul sont couramment utilisés en informatique. Ils permettent de représenter les calculs possibles sur un ordinateur. Le premier est la *machine de Turing* dont la définition a précédé de quelques années l'apparition des premiers ordinateurs; l'autre est la *machine à accès direct* qui est en revanche un outil abstrait issu de la pratique informatique. Il est remarquable que ces deux modèles conduisent à la même notion de calculabilité.

Une machine de Turing (TM) possède un système de contrôle, un ou plusieurs rubans de longueur infinie à droite et éventuellement un ruban d'entrée et/ou de sortie. Le système de contrôle est aussi appelé le programme de la machine. Les opérations disponibles permettent au programme de bouger le curseur d'un ruban à droite ou à gauche, d'écrire en position courante sur un ruban et de passer dans un état dépendant de la valeur du symbole lu et de l'état courant. Il s'agit d'une machine abstraite capable d'effectuer n'importe quel calcul. Il est remarquable qu'une machine aussi simple puisse résoudre de nombreux problèmes. L'avantage principal par rapport à d'autres machines comme les automates est l'apparition du ruban qui permet d'ajouter une capacité de mémorisation à une machine. Contrairement aux automates qui n'ont pas de structure de données aussi sophistiquée, ce simple ajout confère une puissance considérable à ce type de machine et permet de réaliser de nombreux calculs. On représente

une TM par un quadruplet (E, A, δ, i), où E est un ensemble fini d'états, $i \in E$ est l'état initial, A est l'alphabet, et $\delta : E \times A \rightarrow (E \cup \{f, \text{oui}, \text{non}\}) \times A \times \{G, D, R\}$ est le programme de la machine qui possède un unique ruban (f est l'état final, "oui" est l'état acceptant, "non" est l'état refusant, G pour gauche, D pour droite, R pour reste sur place).

Le fonctionnement d'une machine de Turing $M = (E, A, \delta, i)$ est spécifié par le programme (cf. [142]). Dans l'état initial i, le ruban d'entrée contient un ensemble fini de symboles différents du symbole blanc et le curseur est positionné en face de la case la plus à gauche du ruban de travail. De manière séquentielle, en fonction du symbole s lu sur le curseur, et de l'état e, le programme va déterminer un nouveau symbole s', écrire s' à la place de s, passer dans l'état e' et déplacer le curseur suivant la valeur de d, à gauche, à droite ou rester sur place en fonction du symbole G, D ou R, où $(e', s', d) = \delta(e, s)$. Cette procédure est répétée jusqu'à ce que la machine se trouve dans l'état final ou que le curseur sorte du ruban (nécessairement par la gauche). Une machine de Turing peut cependant ne jamais s'arrêter.

Une machine à accès direct, en anglais Random Access Machine (RAM), est constituée comme une machine de Turing d'un programme et d'une structure de données (tableau de registres capables de contenir des entiers de n'importe quelle taille). Les instructions d'une RAM ressemblent à l'ensemble d'instructions des langages de programmation des ordinateurs actuels. Un programme $\Pi = (\pi_1, \ldots, \pi_m)$ est une suite finie d'instructions, un compteur de programme représentant l'évolution du programme et pointant sur l'instruction à exécuter. De plus, un registre spécial, appelé *accumulateur*, permet d'effectuer les calculs arithmétiques et logiques. On voit que ce type de machine est très proche de la structure des ordinateurs actuels. Étant donné que ces deux types de machines permettent de résoudre les mêmes problèmes[9], on peut raisonner sur les machines de Turing et les résultats s'appliqueront aussi aux ordinateurs.

Enfin, on définit de manière informelle, un *algorithme* comme une suite d'étapes ou d'instructions permettant de résoudre un problème. Selon la thèse de Church[10], on peut identifier les machines de Turing avec les algorithmes. Ceci permet de définir le coût d'un algorithme, la *complexité en temps*, comme le nombre d'instructions effectuées par la machine de Turing pour atteindre l'état final à partir de l'état initial. Nous ne nous intéressons ici qu'à la complexité en temps. On la calcule en fonction de la taille de l'entrée. La complexité d'un problème est définie en fonction de l'instance la plus difficile à résoudre, on dit aussi *dans le pire cas*.

1.2.3 Classes de complexité

On peut définir de manière formelle les classes de complexité $\mathcal{P}$, $\mathcal{NP}$, et $\mathcal{BPP}$. On considère ici uniquement la complexité en temps. La complexité est analysée en terme de comportement asymptotique. Cette convention permet d'éliminer certaines entrées de petite taille sur lesquelles la machine pourrait se comporter exceptionnellement bien et nous pouvons alors nous focaliser sur le comportement "général" de la machine.

La classe de complexité $\mathcal{P}$

On dit qu'une machine de Turing s'exécute en *temps polynomial* s'il existe un polynôme P, tel que le nombre d'instructions est inférieur à $P(n)$ où n est la taille de l'entrée w (le nombre de bits du mot w par exemple). La classe de complexité $\mathcal{P}$ (Polynomial) représente l'ensemble des langages L qui peuvent être reconnus de manière "efficace", *i.e.*, par une machine de Turing déterministe en temps polynomial.

9. En effet, on peut simuler tout calcul sur une TM par un calcul sur une RAM et réciproquement. De plus, cette simulation est réalisée en temps polynomial en fonction de la taille de l'entrée.

10. Ce n'est pas une thèse au sens d'un résultat prouvé, mais plutôt une thèse au sens philosophique de ce terme.

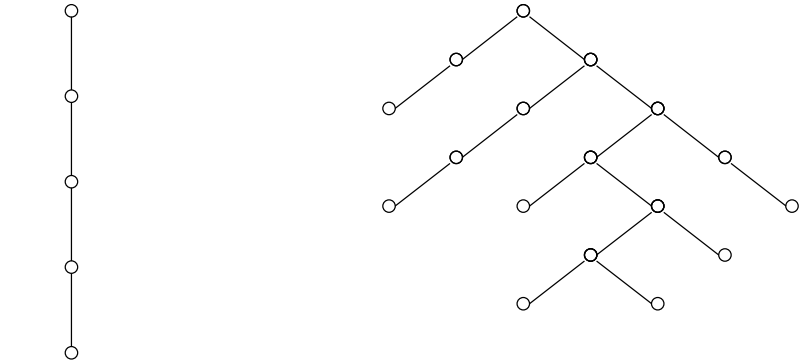
Une telle machine n'a qu'un seul prochain état qui dépend de contenu de l'état courant e et du symbole s sous la tête de lecture. Un langage L est dans $\mathcal{P}$ si et seulement s'il existe une machine de Turing M et un polynôme $Q(.)$ tel que pour tout w suffisamment grand,

1. $w \in L$ ssi la machine M reconnaît le mot w : on note $M(w) = 1$,
2. M termine après au plus $Q(|w|)$ étapes.

On dit que alors la machine M *reconnaît* ou *décide* le langage L.

La classe de complexité $\mathcal{NP}$

La classe de complexité $\mathcal{NP}$ (Non-déterministe en temps Polynomial) représente l'ensemble des langages qui peuvent être acceptés[11] de manière "efficace" par une machine de Turing *non-déterministe*. Une machine de Turing non déterministe est une machine de Turing où la fonction δ définie à la section précédente peut prendre un ensemble de valeurs en fonction de l'état e et du symbole s. Dans l'arbre de calcul, il y a au moins une branche qui mène à un état final acceptant si le mot d'entrée est dans le langage.



Arbre de calcul d'un
Algorithme deterministe

Arbre de calcul d'un
Algorithme non deterministe

FIG. 1.1: *Arbres de calculs.*

Cependant, la découverte de cette branche n'est pas toujours facile à trouver car le nombre de branches dans l'arbre des calculs peut être exponentiel. Ainsi, en temps polynomial, la machine ne peut pas explorer toutes les branches.

De manière informelle, on peut aussi voir $\mathcal{NP}$ comme la classe de tous les langages qui admettent un "court" certificat d'appartenance à un langage. Ce certificat est par exemple, la représentation de la branche qui mène à un état acceptant. Étant donné ce certificat, appelé aussi un *témoin*, l'appartenance à un langage peut être efficacement vérifiée, *i.e.*, en temps polynomial.

Définition 2 *La classe de complexité $\mathcal{NP}$ est la classe de tous les langages L pour lesquels il existe une relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, telle que*

1. *R est décidable en temps polynomial,*

11. Quand une machine de Turing "reconnaît" ou "décide" un langage, elle atteint toujours l'état final. Si elle "accepte" un langage et que le mot n'est pas dans le langage, alors la machine peut ne jamais finir.

2. *il existe un polynôme p tel que $w \in L$ si et seulement s'il existe un témoin t, $|t| \leq p(|w|)$ pour lequel $(w, t) \in R$*

Il est clair que $\mathcal{P} \subseteq \mathcal{NP}$. La question que tente de résoudre la théorie de la complexité depuis le début des années 70 est "est-ce que $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$".

Si $\mathcal{NP}$ est la classe des langages qui ont des certificats de "petite" taille, alors co-$\mathcal{NP}$[12] doit contenir les problèmes qui ont des *disqualifications* de petite taille. C'est-à-dire qu'un langage L est dans co-$\mathcal{NP}$, si les mots qui n'appartiennent pas à L ont des preuves courtes qu'ils ne sont pas dans L; et seules les mots qui n'appartiennent pas à L ont de telles preuves.

Considérons le problème PRIMES qui consiste à déterminer si un entier N codé en binaire (un mot donné) est un nombre premier. Ce problème est dans co-$\mathcal{NP}$. En effet, un diviseur propre de N (différent de 1 et de N) suffit pour disqualifier N de l'ensemble des nombres premiers. Enfin, on peut prouver que PRIMES $\in \mathcal{NP} \cap$ co-$\mathcal{NP}$. En effet, le théorème suivant permet de construire récursivement des certificats de primalité pour n'importe quel nombre premier. Ces certificats sont de taille polynomiale et le temps pour les vérifier est polynomial (cf. [142], 10.2, pages 222–223).

Théorème 1. Un entier $p > 1$ est premier ssi il existe r $(1 < r < p)$ tel que $r^{p-1} = 1 \bmod p$ et $r^{\frac{p-1}{q_i}} \neq 1 \bmod p$ pour tout diviseur q_i de $p - 1$.

Alors $C(r; q_1, \ldots, q_k)$ où q_i parcours l'ensemble des diviseurs de $p - 1$ est un certificat de primalité de p et par conséquent PRIMES $\in \mathcal{NP}$.

La classe de complexité $\mathcal{BPP}$

Une machine de Turing *probabiliste* en temps polynomial PPTM est une TM[13] qui possède en plus un ruban de bits aléatoires, et qui, sur l'entrée w de taille n, s'exécute en un nombre polynomial $P(n)$ d'étapes. La plupart des algorithmes en cryptographie sont représentés par des machines de Turing probabilistes s'exécutant en temps polynomial (PPTM). La classe de problème $\mathcal{BPP}$ contient les problèmes qui sont décidables par un algorithme probabiliste. Il s'agit de machines de Turing qui ont accès à un ruban aléatoire. Ce ruban aléatoire permet à la machine d'indiquer la branche choisie dans l'arbre des calculs. La machine n'est donc pas déterministe car à chaque étape, elle peut prendre un ensemble fini d'états différents en fonction du ruban aléatoire.

Un langage L est dans $\mathcal{BPP}$ (Probabilité d'erreur Bornée en temps Polynomial) si et seulement s'il existe une TM, $M(w, r)$ où r est le contenu du ruban aléatoire, et des polynômes p et ℓ tels que ssur le mot d'entrée w,

1. $w \in L \Rightarrow \underset{|r|<\ell(|w|)}{\Pr} [M(w, r)\text{accepte}] \geq \frac{2}{3}$.
2. $w \notin L \Rightarrow \underset{|r|<\ell(|w|)}{\Pr} [M(w, r)\text{accepte}] \leq \frac{1}{3}$.
3. $M(w, r)$ termine toujours après au plus $p(|w|)$ étapes.

Il est clair que $\mathcal{BPP} =$ co-$\mathcal{BPP}$ car la définition est symétrique. Nous savons aussi que $\mathcal{P} \subseteq \mathcal{BPP}$ car il suffit d'ignorer le ruban aléatoire. Cependant, nous ne savons pas si cette inclusion est stricte ou

12. Soit C une classe de problèmes, alors co-C est l'ensemble des problèmes L tels que $\bar{L} \in C$ où $\bar{L}$ représente l'ensemble des mots qui ne sont pas dans le langage L. Autrement dit, on réfléchit sur les mots qui ne sont pas dans le langage et non plus sur les mots qui sont dans le langage.

13. On rappelle que les machines de Turing peuvent avoir plusieurs rubans.

non, bien qu'une telle séparation soit conjecturée. Un exemple de langage connu pour être dans $\mathcal{BPP}$ et conjecturé non dans $\mathcal{P}$ est le langage PRIMES. Le test de primalité de Miller-Rabin présente un algorithme probabiliste en temps polynomial prouvant que PRIMES $\in \mathcal{BPP}$. Mais, aucun algorithme connu à ce jour permet de *décider* en temps polynomial si un mot est un nombre premier. On ne sait pas non plus si $\mathcal{BPP}$ est un sous-ensemble strict de $\mathcal{NP}$.

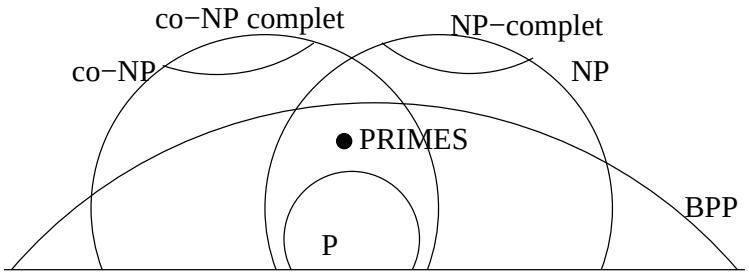


FIG. 1.2: *Conjecture entre les classes de complexité.*

Remarque 1 ***(Motivation pour étudier asymptotiquement les algorithmes lorsque les entrées ont une taille non majorée)*** *La complexité d'un algorithme prend tout son sens quand on considère des entrées de taille infiniment grande. En effet, dans le cas contraire, si l'ensemble E des valeurs d'entrée est fini (il existe k tel que $E \subset \{0,1\}^k$), alors sur cet ensemble, l'algorithme A s'arrête en un temps T au plus. Par conséquent, il existe toujours un polynôme P tel que $P(k) > T$. En effet, un algorithme dans $\mathcal{P}$ se termine toujours, et dans ce cas, tout algorithme qui s'exécute en temps au plus T appartiendrait à $\mathcal{P}$.*

1.2.4 Machines de Turing à oracle et réductibilité

Les oracles peuvent être vus comme des composants que l'on installe sur une machine pour en augmenter la puissance. Les oracles que nous utilisons n'ont pas une puissance de calcul infinie mais savent résoudre certains problèmes. On notera $\mathcal{S}^{\mathcal{A}}$ un algorithme (ou une machine de Turing) $\mathcal{S}$ qui peut faire appel à un oracle $\mathcal{A}$. L'interface entre la machine $\mathcal{S}$ et l'oracle $\mathcal{A}$ doit être formalisée. Une machine de Turing $M^?$ à oracle est une machine de Turing déterministe (probabiliste ou non) à plusieurs rubans qui a un ruban spécial appelée *le mot question*, et trois états spéciaux $q_?$, q_{YES}, q_{NON}. On a défini ces machines indépendamment de l'oracle utilisé $?$. La classe $\mathcal{C}^A$ est la classe de complexité qui regroupe les langages qui peuvent être décidés par des machines qui décident les langages dans $\mathcal{C}$ en ayant accès à un oracle A. Les machines de Turing à oracle sont des machines très puissantes et on peut montrer que $\mathcal{NP} \subseteq \mathcal{RP}^{\oplus\mathcal{P}}$: c'est-à-dire qu'une machine de Turing non-déterministe n'est pas plus puissante qu'une machine de Turing à oracle dans $\mathcal{RP}$, *i.e.*, probabiliste qui rejette correctement si un mot n'est pas dans le langage L et si le mot est dans le langage décide avec probabilité $\geq 1/2$ et $\oplus\mathcal{P}$ est la classe des langages pour lesquels, il existe une machine de Turing qui calcule si le nombre de solutions est impair.

La construction d'une machine à oracle est souvent utilisée pour effectuer des réductions. Intuitivement, un problème Q peut se réduire à un autre problème Q' si une instance de Q peut "facilement être formulée" comme une instance de Q' dont la solution fournira une solution pour l'instance de Q. Pour les problèmes de décision, on dit que le langage L_1 est réductible au langage L_2 en temps polynomial, s'il existe une fonction *calculable en temps polynomial* $f : \{0,1\}^* \to \{0,1\}^*$ telle que pour tout $x \in \{0,1\}^*$, $x \in L_1$ si et seulement si $f(x) \in L_2$. On appelle la fonction f, fonction de réduction, et un algorithme polynomial F qui calcule f est appelé algorithme de réduction. L'algorithme A_1 est capable

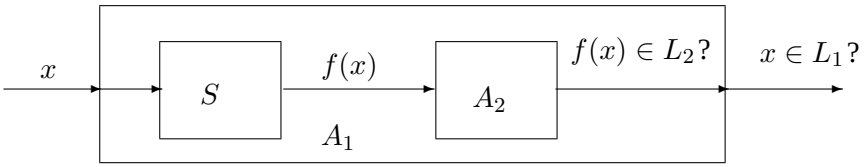


FIG. 1.3: *Réduction de Karp.*

de décider si $x \in L_1$ en utilisant F pour transformer une entrée x quelconque en $f(x)$ puis en utilisant A_2 pour décider si $f(x) \in L_2$.

On dit que le problème A est au moins aussi difficile que le problème B si B se réduit à A. Comme les réductions sont transitives, car si f_1 et f_2 sont deux fonctions calculables en temps polynomiale, alors $f_2 \circ f_1$ est encore une fonction calculable en temps polynomiale, il est possible d'ordonner les problèmes par rapport à leur difficulté. Soit $\mathcal{C}$ une classe de complexité. On dit que le langage L est *$\mathcal{C}$-complet* si tout langage $L' \in \mathcal{C}$ peut être réduit à L. La classe des problèmes $\mathcal{NP}$-complet représente la classe des problèmes les plus difficiles de la classe $\mathcal{NP}$. Si on prouve qu'un seul de ces problèmes est dans $\mathcal{P}$, c'est-à-dire qu'il existe un algorithme en temps polynomial pour en résoudre un, alors on aura prouver que $\mathcal{P} = \mathcal{NP}$. De plus, le problème bien connu du voyageur de commerce fait partie de cette classe. Soient un graphe complet avec une fonction coût sur chaque arête et un entier k, existe-t-il une tournée (i.e., un cycle qui passe par chaque sommet), telle que le coût total soit inférieur à k? Le lecteur désirant connaître plus d'information sur divers problèmes $\mathcal{NP}$-complet pourra lire [86].

On utilise dans cette thèse les réductions au sens de Cook ou de Turing, c'est-à-dire que l'algorithme A_1 peut interroger un nombre polynomial de fois l'algorithme A_2 pour résoudre le problème $x \in L_1$?. Dans les réductions au sens de Karp [14], un seul accès à l'algorithme A_2 est autorisé et à la fin. Il est évident que deux problèmes décisionnels équivalents au sens de Karp seront Turing-équivalent, mais on ne sait pas si la réciproque est vraie [15]. Dans les preuves de sécurité, nous aurons besoin dans la suite de cette thèse de la notion de *réduction randomisée au sens de Turing*. En effet, pour prouver la sécurité d'un schéma $\mathcal{S}$, on suppose qu'il existe un adversaire $\mathcal{A}$, machine de Turing probabiliste, qui sait casser la sécurité de $\mathcal{S}$ et on construit un attaquant $\mathcal{B}$, machine de Turing probabiliste [16], contre un problème réputé difficile. Or il n'existe pas d'attaquant $\mathcal{B}$ car le problème est difficile. Par conséquent, l'adversaire $\mathcal{A}$ ne peut pas exister. Dans ces preuves, l'attaquant $\mathcal{B}$ peut interroger un nombre polynomial de fois l'adversaire $\mathcal{A}$. On construit ainsi des réductions au sens de Cook ou de Turing entre le jeu que tente de résoudre l'adversaire pour casser la sécurité du schéma $\mathcal{S}$ et le problème conjecturé difficile. Nous décrirons dans la suite de ce chapitre des exemples de problèmes conjecturés difficiles et les jeux que tente de résoudre un adversaire pour casser la sécurité d'un système de chiffrement ou de signature.

Le rôle de $\mathcal{B}$ dans cette réduction est de simuler les entrées du problème difficile en entrée pour l'adversaire et vice-versa en sortie de sorte que l'adversaire ne puisse pas détecter s'il est en présence d'entrées réelles ou simulées. Si l'adversaire pouvait distinguer les entrées "réelles" des entrées "simulées", il pourrait décider de ne pas répondre sur les entrées simulées. Par conséquent, l'attaquant, aussi appelé simulateur, ne pourrait plus utiliser l'adversaire pour résoudre le problème difficile. Ainsi, il est

14. appelées aussi, *many-one en temps polynomial*, ou *transformation polynomiale*.

15. Il existe un type de réductions intermédiaire entre ces deux types. On les appelle *réduction de tables de vérité en temps polynomial*. Dans ces réductions, on peut demander plusieurs questions "$x \in L'$? " *mais elles doivent toutes être posées avant que l'on ait la réponse d'une d'entre elles.* Ainsi, on obtient la réponse finale comme une fonction booléenne des réponses, d'où le nom.

16. Cette machine possède un ruban de bits aléatoires et peut donc poser des questions randomisées à la machine $\mathcal{A}$.

primordial de définir l'interface du type d'entrée qu'accepte l'oracle pour savoir si la simulation des entrées est indistinguable[17] pour l'adversaire d'entrées réelles.

Dans le cas des langages réductibles au sens de Karp, on n'a pas besoin d'appeler F en sortie pour résoudre le problème que tente de résoudre A_1.

Récemment, Okamoto et Pointcheval [135] ont considéré et défini des types de problèmes mettant en jeu des oracles. Ces problèmes sont appelés les *gap-problèmes* et peuvent être formalisés intuitivement de la manière suivante : résoudre un problème d'inversion à l'aide d'un oracle pour un problème de décision relié. Un *gap-problème* pour f est, étant donné x et une relation f, trouver y satisfaisant $f(x, y) = 1$, à l'aide d'un oracle qui, sur la donnée (x', y'), répond si $f(x', y') = 1$ ou non. Dans la sous-section suivante, nous étudierons une classe de fonctions pour lesquelles retrouver un x à partir de y tel que $y = f(x)$ est difficile. Dans ce cas, les gap-problèmes précédents sont une classe de fonction intermédiaire.

1.2.5 Fonctions à sens unique

La conjecture $\mathcal{P} \neq \mathcal{NP}$ implique qu'il existe des problèmes calculatoires d'un grand intérêt qui sont difficiles. En anglais, les fonctions à sens unique sont appelées One-Way Functions. Les fonctions à sens unique[18], fonctions "faciles" à calculer et "difficile" à inverser, sont des primitives très utiles en cryptographie. L'existence des fonctions à sens unique est une condition nécessaire à l'existence de la plupart des primitives cryptographiques connues (chiffrement et signature numérique). L'état actuel des connaissances en théorie de la complexité ne permet pas de prouver l'existence des fonctions à sens unique. On doit donc se contanter de supposer leur existence.

La cryptographie moderne cherche à construire des algorithmes efficaces pour les utilisateurs légitimes et à rendre impossible pour un adversaire la recherche des informations protégées. Dans les systèmes de chiffrement, l'utilisateur légitime est capable de déchiffrer les messages (en utilisant des informations privées à sa disposition), alors que pour un adversaire, qui n'a pas cette information privée, la tâche de *décrypter*[19] (i.e., "cassage" du chiffrement) le message chiffré devra être impossible. Il est clair que le cassage peut être exécuté par une machine non-déterministe en temps polynomial. En effet, si on donne accès à la variable auxiliaire représentant la clé secrète, la machine de l'adversaire peut déchiffrer en temps polynomial comme l'utilisateur légitime. Cependant, l'exigence de sécurité impose que le cassage ne doit pas être faisable, *i.e.* ne doit pas pouvoir être effectué par une machine probabiliste en temps polynomial. Par conséquent, l'existence de schémas de chiffrement sûrs implique qu'il existe des tâches réalisables par des machines non-déterministes en temps polynomial mais qui ne peuvent pas être exécutées par des machines déterministes (ou même probabilistes) fonctionnant en temps polynomial. En d'autres mots, une condition nécessaire pour l'existence de systèmes de chiffrement sûrs est que $\mathcal{NP}$ ne soit pas contenu dans $\mathcal{BPP}$ (et donc que $\mathcal{P} \neq \mathcal{NP}$). Cependant, la condition nécessaire $\mathcal{P} \neq \mathcal{NP}$ n'est pas suffisante. $\mathcal{P} \neq \mathcal{NP}$ implique seulement que le système de chiffrement est difficile à casser dans le pire des cas. Il ne dit rien sur le fait que le système soit facile à casser dans presque tous les cas. Par conséquent, la difficulté dans le pire cas n'est pas une bonne mesure pour la sécurité et il est possible de construire des schémas pour lesquels casser le problème est $\mathcal{NP}$-complet alors qu'il existe un algorithme efficace pour le casser dans 99% des cas. Cependant, on peut montrer que l'existence de

17. Il existe plusieurs niveaux d'indistinguabilité. Deux distributions peuvent être égales, statistiquement indistinguables (la distance entre ces deux distributions est négligeable), polynomialement indistinguable (aucune machine de Turing polynomiale ne pourra distinguer les deux distributions) ou "calculatoirement indistinguables" (les distributions sont, en principe, indistinguables).

18. La théorie de la $\mathcal{NP}$-complétude réfléchit sur les problèmes décisionnels alors que la cryptographie utilise aussi bien les problèmes décisionnels que les fonctions.

19. Il ne faut pas confondre, l'opération de décrypter qui consiste à retrouver un message clair à partir d'un message chiffré sans connaître la clé de déchiffrement, et l'opération de déchiffrement, qui consiste à utiliser la clé de déchiffrement.

fonction à sens unique implique que $\mathcal{P} \neq \mathcal{NP}$. On peut définir formellement les fonctions à sens unique de la manière suivante.

Définition 3 ***(Fonction négligeable)*** *Une fonction $\mu : \mathbb{N} \mapsto [0,1]$ est dite négligeable si pour tout polynôme positif p, il existe N, tel que pour tout $n > N$, $\mu(n) < 1/p(n)$ ($\mu(k) = k^{-\omega(1)}$).*

Définition 4 ***(Fonction à sens unique)*** *Une fonction $f : \{0,1\}^* \to \{0,1\}^*$ est dite à sens unique si les deux conditions suivantes sont vérifiées :*

1. *facile à calculer : la fonction f est calculable en temps polynomial.*
2. *difficile à inverser : pour toute machine probabiliste M s'exécutant en temps polynomial, pour tout polynôme positif p, et tout n suffisamment grand,*

$$\Pr_x \left[M(1^n, f(x)) \in f^{-1}(f(x)) \right] < \frac{1}{p(n)}$$

où x est uniformément distribué dans $\{0,1\}^n$

Une hypothèse calculatoire suffisante pour toutes les constructions utilisées ci-après est l'existence de permutations à trappe. Il y a des ensembles de permutations à sens unique, $\{f_\alpha\}$, ayant la propriété supplémentaire que f_α est efficacement inversible une fois que l'on a une entrée auxiliaire, une "trappe" pour l'indice α. La trappe de l'indice α, notée $t(\alpha)$, *ne peut pas* être efficacement calculée à partir de α, mais on peut efficacement générer des paires correspondantes $(\alpha, t(\alpha))$.

1.3 Fonctions conjecturées à sens unique

Choisir des hypothèses difficiles qui semblent "raisonnables et naturelles" sur lesquelles baser la sécurité d'un schéma est une tâche difficile. En effet, quand on choisit une telle hypothèse, on espère que le problème "difficile" a été bien étudié. Quand on a besoin d'une hypothèse particulière pour analyser la sécurité d'un schéma, on cherche à comparer cette nouvelle hypothèse à l'un des deux types d'hypothèses standards que l'on va définir dans cette section. Pour ce faire, on cherche des réductions polynomiales entre les problèmes. Voici des exemples de fonctions conjecturées comme des fonctions à sens unique. Dans cette section, p et q sont des nombres premiers de longueur k, où k représente le paramètre de sécurité.

1.3.1 Problèmes liés à la factorisation

Définition 5 ***(Le problème de la factorisation)*** *Étant donné un entier N, calculer des facteurs non triviaux de N (différent de 1 et N)*[20].

Ce problème est facile si N est de forme spéciale par exemple pair. Les instances difficiles de ce problème semblent être les nombres de la forme $N = pq$ où p et q sont de grande taille équivalente. On peut donc conjecturer que la fonction $f(p,q) = pq$ est à sens unique : *i.e.*, il est facile de calculer $N = pq$, étant donnés p et q, en temps $\mathcal{O}(k^2)$, mais il n'existe pas d'algorithme probabiliste en temps polynomial qui, sur une entrée aléatoire N de la forme pq, permet de calculer p et q en temps raisonnable[21] en moyenne

20. Factoriser N ne veut pas forcément dire factoriser complètement N en facteurs premiers.
21. *i.e.* pour une machine de Turing M probabiliste en temps polynomial, $(M \in \mathcal{BPP})$.

pour des paires (p, q) choisies au hasard parmi les grands nombres premiers. De même, il est facile de générer p et q et de tester leur primalité en temps $\mathcal{O}(k^3 \log(k))$.

Définition 6 ***Le problème RSA.*** *Étant donné (N, e, y) où $y \in_R \mathbb{Z}_N^*$ et $N = pq$, trouver x tel que $y = x^e \bmod N$.*

L'hypothèse RSA est la suivante. Pour tout $y \in \mathbb{Z}_N^*$ choisi au hasard, il est difficile de calculer en temps "raisonnable" x tel que $y = x^e \bmod N$.

Le cryptosystème RSA, proposé par Rivest, Shamir et Adleman, est basé sur cette hypothèse calculatoire. La fonction $f(x) = x^e \bmod N$ est un exemple de permutation à trappe. En effet, soit un entier d tel que $ed = 1 \bmod \varphi(N)$, où $\varphi(N) = (p-1) \times (q-1)$ si $N = pq$. Cette condition peut être écrite sous la forme : il existe un entier u tel que $ed = 1 + u\varphi(N)$. Alors, le théorème d'Euler dit que, pour tout $x \in \mathbb{Z}_N^*$, $x^{\varphi(N)} = 1 \bmod N$. Par conséquent, $y^d = (x^e)^d = x^{ed} = x^{1+k\varphi(N)} = x \times (x^{\varphi(N)})^k = x \times 1^k = x \bmod N$. L'entier d est donc une trappe qui permet de résoudre le problème RSA.

Si on sait factoriser N[22], on peut trouver x car on peut calculer $\varphi(N)$ et le calcul de la trappe d peut être réalisé efficacement en $\log(N)$ étapes grâce à l'algorithme d'Euclide étendu. En revanche, si on sait résoudre RSA, on ne sait pas si l'on peut factoriser N. Cette direction est primordiale car il se pourrait que l'on sache résoudre RSA sans avoir besoin de factoriser N. Dans ce cas, l'estimation des algorithmes de factorisation ne servirait à rien ! Cependant, le problème RSA est un problème bien étudié depuis 20 ans [26] et il ne semble pas y avoir de méthodes pour résoudre ce problème autres que la factorisation.

Pour factoriser un nombre, on peut utiliser l'algorithme prouvé des carrés aléatoires de Dixon qui s'exécute en $\mathcal{O}(e^{(1+o(1))\sqrt{\ln N \ln\ln N}})$, où N est le module RSA. En pratique, on peut en considérer d'autres. Soit $\ell = |p|$, p, le plus petit diviseur premier de N. L'algorithme utilisant les courbes elliptiques a une complexité en moyenne en $\mathcal{O}(e^{(1+o(1))\sqrt{2\ell \ln \ell}})$. La complexité du crible quadratique est en moyenne en $\mathcal{O}(e^{(1+o(1))\sqrt{\ln N \ln\ln N}})$. Le meilleur algorithme est le crible algébrique [119] où la complexité heuristique en moyenne est en $\mathcal{O}(e^{(c+o(1))(\ln N)^{1/3}(\ln\ln N)^{2/3}})$, avec $c = (64/9)^{1/3} \approx 1.923$.

Au-delà de ces valeurs asymptotiques, on peut se demander en pratique, quelle est la taille des paramètres que l'on doit considérer. On prend en général comme paramètre de sécurité 2^{80} opérations. Pour fixer un ordre de grandeur sur ces valeurs "immenses", on a estimé l'âge de l'univers à 4.10^{11} années, ce qui représente environ 2^{87} instructions sur un ordinateur cadencé à 1 Ghz. La taille recommandée pour les systèmes basés sur la factorisation est de considérer des modules $N = pq$ de 1024 bits, où p et q sont de taille 512 bits. Dans la formule asymptotique, en remplaçant $\ln N$ par $1024 \times \ln 2$, on obtient 2^{87}. Pour un module RSA de 512 bits $\ln N \approx 512$, il faut environ $2.10^{19} \approx 2^{63}$. Lors de la factorisation d'un module de 512 bits il y a deux ans (août 1999), il a été observé que le coût était en fait de $3.10^{17} \approx 2^{56}$ opérations. Il apparaît donc que la formule du crible algébrique surestime le temps de calcul. Cet algorithme fonctionne mieux en pratique qu'en théorie. Pour une approximation du coût de la factorisation et de la longueur des clés recommandées, voir [120].

1.3.2 Problèmes liés au calcul du logarithme discret

Soit g un générateur d'ordre q d'un sous-groupe G de $\mathbb{Z}_p^*$. D'après le théorème de Lagrange q est un diviseur de $p-1$ car p est premier.

Définition 7 ***Le problème du logarithme discret (DL).*** *Étant donnés p, g et $y \in_R G$, trouver x tel que $y = g^x$.*

22. Un nombre N de la forme pq où p et q sont deux grands nombres premiers est couramment appelé un module RSA.

L'hypothèse dite du logarithme discret est donc qu'il est difficile de calculer x en temps raisonnable si y est une instance aléatoire dans $\mathbb{Z}_p^*$.

Les valeurs p et g ne sont pas nécessairement choisies au hasard. Le nombre premier p est sélectionné pour avoir certaines propriétés qui rendent en pratique le calcul du logarithme discret difficile. Par exemple, p doit être choisi tel que $p-1$ ait de grands diviseurs premiers. Dans le cas où $|q| = 160$, les calculs modulo q de petite taille permettent accélérer les exponentiations modulaires. En effet, l'algorithme d'exponentiation modulaire effectue un nombre de multiplications linéaire en la taille de l'exposant.

Le meilleur algorithme connu jusqu'à présent pour calculer des logarithmes discrets est l'algorithme de calcul d'indexe. Le temps en moyenne de cet algorithme est polynomial en $e^{c\sqrt{\ln p \ln \ln p}}$ où $c > 0$. Il existe une variante de l'algorithme du crible algébrique pour calculer les logarithmes discrets qui a la même complexité que pour la factorisation.

Le record de calcul d'un logarithme discret est depuis le 17 avril 2001[23] de 120 chiffres décimaux, ce qui fait un peu moins de 400 bits. Il est à noter qu'il existe une différence entre le record pour la factorisation et le record pour le calcul du logarithme discret de plus de cent bits. Cette différence peut s'expliquer en étudiant rapidement les deux étapes essentielles du crible algébrique. Dans un premier temps, le crible algébrique réalise un crible et détermine un ensemble d'équations. Cette phase peut être distribuée. La deuxième phase consiste à inverser une matrice. La différence se situe dans cette phase. Dans le cas de la factorisation, une matrice binaire doit être inversée alors que dans le cas du logarithme discret, il s'agit d'une matrice à coefficients dans $\mathbb{Z}_p$. On peut remarquer que le crible algébrique ne tient pas compte de la taille du groupe engendré par g. Enfin, contrairement à la factorisation où le crible permet d'attaquer un seul module RSA, la phase de crible dans le cas du logarithme discret peut être réutilisée pour calculer d'autres exposants pour un même groupe.

La taille recommandée d'un module DL est de 1024 bits puisque la formule de complexité du crible algébrique est identique et l'inversion de la matrice peut être améliorée.

Enfin, on peut citer deux problèmes classiques reliés au problème du calcul du logarithme discret.

Définition 8 ***Le problème Diffie-Hellman Calculatoire (CDH).*** *Étant donnés g, $g^x \in_R G$ et $g^y \in_R G$, calculer g^{xy}.*

L'hypothèse CDH est qu'il est difficile de résoudre le problème CDH en temps "raisonnable".

Il est clair que si l'on sait résoudre le problème du logarithme discret, on sait résoudre le problème CDH en calculant x à partir de g^x et en calculant $(g^y)^x$, on obtient g^{xy}. Cependant, comme dans le cas RSA, on ignore si l'on sait résoudre le problème CDH à partir de la résolution du logarithme discret. Maurer a étudié des réductions entre ces deux problèmes (cf. [122, 123]). Il existe une version décisionnelle de ce problème très utilisée.

Définition 9 ***Le problème Diffie-Hellman Décisionnel (DDH).*** *Étant donné $(g, g^x, g^y, g^z) \in_R G^4$, décider si $z = xy \bmod ord(g)$?*

L'hypothèse DDH est qu'il est difficile de résoudre le problème DDH en temps "raisonnable".

Ce problème est plus simple que le problème Diffie-Hellman calculatoire qui consiste à calculer g^z, mais il est apparemment difficile à résoudre (cf. [25]).

1.4 Modèle de sécurité

Le modèle de sécurité permet de modéliser la vie réelle et les interactions des différents acteurs. Il convient également de modéliser l'attaque que cherche à monter l'adversaire. Par exemple, quand on

23. Antoine Joux (DCSSI) et Reynald Lercier (CELAR)

dit que l'adversaire tente de casser un schéma de chiffrement, quel est son but ? Retrouver la clé secrète de chiffrement ? Retrouver le clair d'un chiffré ? Connaître le bit de poids faible ou de poids fort du message clair ? La modélisation de ce que signifie "casser" un système de chiffrement a été une tâche difficile. Pour les primitives usuelles de cryptographie comme le chiffrement et la signature, nous avons aujourd'hui des modèles bien définis que nous verrons dans les sous-sections 1.4.4 et 1.4.5.

1.4.1 Hypothèses sur le canal de communication

On fera comme hypothèse que l'on utilise un réseau de communication *public* comme l'Internet. Les messages envoyés pourront être interceptés par l'adversaire et ce dernier pourra envoyer des messages à la place d'un utilisateur légal.

1.4.2 Classification des adversaires

En cryptographie moderne, un adversaire tente de résoudre un problème difficile. Il est représenté par une machine de Turing probabiliste en temps polynomial (PPTM). On suppose que les langages que sait résoudre ou décider cet adversaire sont dans la classe de langages $\mathcal{BPP}$.

1.4.3 Arguments de sécurité dans le modèle de l'oracle aléatoire

Il existe de nombreux exemples dans la littérature cryptographique de systèmes qui ont des preuves de sécurité mais ne sont pas utilisables en pratique et de systèmes qui sont efficaces en pratique mais pour lesquels il n'existe pas de preuve de sécurité. Il y a peu de schémas à la fois pratiques et sûrs. A cause de cette situation, une nouvelle direction en cryptographie s'est développée et propose de faire des preuves de sécurité dans un modèle particulier, dit *le modèle de l'oracle aléatoire* [9]. Les fonctions de hachage (MD5 et SHA-1 par exemple) sont considérées comme si elles fonctionnaient comme des oracles aléatoires, *i.e.*, comme des boîtes noires qui contiennent une fonction aléatoire qui ne peut être évaluée qu'en faisant des requêtes explicites. Les analyses de sécurité dans ce modèle ne sont plus appelées des *preuves* de sécurité mais des *arguments* de sécurité.

Certains résultats [36] ont montré qu'il existe des schémas de signature et de chiffrement qui sont sûrs dans le modèle de l'oracle aléatoire, mais pour lesquels n'importe quelle implémentation de l'oracle aléatoire implique des schémas non sûrs. Néanmoins, il reste que ce modèle est utile pour évaluer la sécurité des schémas même si on ne peut pas faire à proprement parler de preuve. Comme nous le verrons dans le chapitre 2, ce modèle permet aussi de transformer des preuves de connaissance en preuves non-interactives grâce à l'heuristique due à Fiat-Shamir [69] ce qui nous sera bien utile pour prouver que les joueurs ont correctement effectué leur tâche dans les protocoles partagés.

Dans ce modèle, on autorise l'oracle à imposer la valeur de la fonction en certains points non encore été définis par lui. La seule contrainte est donc que si l'on questionne deux fois l'oracle aux mêmes points on obtienne la même réponse. On peut voir l'oracle aléatoire comme un tableau infini qui à chaque valeur d'entrée associe une valeur aléatoire dans $\{0,1\}^L$ où L est la taille de sortie de la fonction de hachage.

L'oracle peut choisir les valeurs de sortie de deux manières possibles. La première est de définir la sortie de telles fonctions avec des valeurs aléatoires. La seconde permet d'imposer la valeur de sortie pourvu qu'elle apparaîsse aléatoire.

1.4.4 Sécurité d'un système de chiffrement

Dans cette section, nous définissons un système de chiffrement et nous en donnons deux exemples très importants. Puis, nous définissons différentes notions de sécurité et nous montrons la sécurité sémantique du cryptosystème El Gamal.

Description d'un schéma de chiffrement

Définition 10 ***Système de chiffrement à clé publique.*** *Un schéma de chiffrement à clé publique se compose d'un triplet d'algorithmes s'exécutant en temps polynomial* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$.

- *L'algorithme de génération des clés* $\mathcal{K}$ *est un algorithme probabiliste en temps polynomial qui prend en entrée un paramètre de sécurité* k *(que l'on note souvent en notation unaire* 1^k*) et produit une paire* $(\mathsf{pk}, \mathsf{sk})$ *où* pk *est appelée la clé publique, et* sk *la clé secrète correpondante. On notera* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}(1^k)$*. On dira aussi que la paire* $(\mathsf{pk}, \mathsf{sk})$ *correspond à la paire de clés de chiffrement/déchiffrement.*

- *L'algorithme de chiffrement* $\mathcal{E}$ *est un algorithme probabiliste en temps polynomial qui prend en entrée un paramètre de sécurité* 1^k*, une clé publique* pk *choisie par* $\mathcal{K}(1^k)$*, et une chaîne* $m \in \{0,1\}^k$*, appelée le message, et qui produit une chaîne en sortie* $c \in \{0,1\}^*$ *appelé le chiffré. On utilisera la notation* $c \in \mathcal{E}_{\mathsf{pk}}(m; r)$ *pour dire que* c *est un chiffrement du message* m *en utilisant la clé publique* pk *avec un paramètre* k *de sécurité et un random* r*.*

- *L'algorithme de déchiffrement* $\mathcal{D}$ *est un algorithme en temps polynomial qui prend en entrée un paramètre de sécurité* 1^k*, une clé secrète* sk *choisie dans* $\mathcal{K}(1^k)$*, quelques fois un random* s *et un chiffré* $c \in \mathcal{E}_{\mathsf{pk}}(m; r)$ *et qui produit une chaîne* $m' \in \{0,1\}^*$ *telle que pour toute paire* $(\mathsf{pk}, \mathsf{sk})$ *dans* $\mathcal{K}(1^k)$*, pour tout message* m*, et pour tout chiffré* $c \in \mathcal{E}_{\mathsf{pk}}(m; r)$*, la probabilité* $Pr\left[\mathcal{D}_{\mathsf{sk}}(c; s) \neq m'\right]$ *est négligeable.*

Comment utiliser un tel système ?

Pour utiliser un système de chiffrement à clé publique $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ avec comme paramètre de sécurité 1^k, l'utilisateur Alice exécute l'algorithme de génération de clés $\mathcal{K}(1^k)$ pour obtenir une paire $(\mathsf{pk}, \mathsf{sk})$ de clés de chiffrement/déchiffrement. Alice publie alors pk dans un fichier public (annuaire), et conserve secrète la clé sk. Si Bob souhaite envoyer un message à Alice, alors il a besoin d'obtenir pk et exécute $\mathcal{E}_{\mathsf{pk}}(m; r)$. Ensuite, Alice calcule le message $m = \mathcal{D}_{\mathsf{sk}}(c)$.

Description du schéma de chiffrement RSA

Le système de chiffrement à clé publique le plus utilisé à l'heure actuelle est le chiffrement RSA [163]. La sécurité de ce cryptosystème est basée sur le problème RSA. La clé publique d'Alice est constituée de $\mathsf{pk} = (N, e)$, où $N = pq$ est un module RSA et e est l'exposant public premier avec $\varphi(N) = (p-1)(q-1)$. La clé secrète d'Alice est constituée de l'exposant privé $\mathsf{sk} = d$ tel que $ed = 1 \bmod \varphi(N)$. Il est aisé de calculer l'inverse d de e modulo $\varphi(N)$ en utilisant l'algorithme d'Euclide étendu. Cependant, à partir de N, il est difficile de calculer d. Pour envoyer un message chiffré à Alice, Bob obtient la clé publique et calcule $c = m^e \bmod N$. Pour déchiffrer le chiffré c, Alice calcule $c^d \bmod N$. La validité de l'opération utilise le théorème d'Euler. Pour tout $x \in \mathbb{Z}_N^*$, $x^{\varphi(N)} = 1 \bmod N$. Ainsi, quand Alice calcule $c^d \bmod N$, elle obtient

$$m^{ed} = m^{1+k\varphi(N)} = m \times (m^{\varphi(N)})^k = m \times (1)^k = m \bmod N$$

Remarque 2 *On peut remarquer que la relation* $m^{ed} = m \bmod$ *est valable pour tout* $m \in \mathbb{Z}_N$*.*

Description du schéma de chiffrement El Gamal

Le système de chiffrement El Gamal [66] est basé sur le problème du logarithme discret. Dans ce cryptosystème, les paramètres sont (g, q, p) tels que $q|p-1$ où p et q sont des nombres premiers et g est un générateur d'un sous-groupe G de $\mathbb{Z}_p^*$ d'ordre premier q. La clé publique pk est un entier $y = g^x \in G$ et la clé privée est $\mathsf{sk} = x \in \mathbb{Z}_q$.

Pour chiffrer un message m pour Alice, Bob génère un random $r \in \mathbb{Z}_q^*$ et calcule $(A, B) = (g^r, my^r)$. Pour déchiffrer c, Alice calcule $B/A^x = m \times \frac{y^r}{(g^r)^x} = m \times \frac{(g^x)^r}{(g^r)^x} = m$.

Attaques contre les systèmes de chiffrement

Par ordre croissant de force, on distingue :

– Dans les attaques à clairs choisis (CPA = Chosen Plaintext Attack), l'adversaire obtient les textes chiffrés des textes clairs de son choix. Cette attaque ne peut pas être évitée dans le cadre d'un chiffrement à clé publique car on donne la clé publique à l'adversaire.

– Dans les attaques à chiffrés choisis non-adaptatives (CCA1 = non-adaptive Chosen-Ciphertext Attack), aussi appelée "lunch-time attacks", formalisées par Naor et Yung [129], l'adversaire obtient en plus de la clé publique, l'accès à un oracle de déchiffrement. L'adversaire peut utiliser cet oracle *uniquement* pendant la période précédant la réception du chiffré cible c. Le terme non-adaptative fait référence au fait que les questions à l'oracle de déchiffement ne peuvent pas dépendre du chiffré c.

– Dans les attaques à chiffrés choisis adaptatives (CCA2 = adaptive Chosen-Ciphertext Attack), dues à Rackoff et Simon [160], l'adversaire obtient en plus de la clé publique l'accès à un oracle de déchiffrement, mais cette fois-ci, l'adversaire peut l'utiliser même après la réception du chiffré cible c avec la seule restriction qu'il ne peut pas demander le déchiffrement de c. L'attaque est dite "adaptative" parce que les questions à l'oracle de déchiffrement peuvent dépendre du chiffré cible c.

Notions de sécurité

Il existe trois notions de sécurité pour un système de chiffrement : la "one-wayness", la "sécurité sémantique", et la "non-malléabilité".

La notion de sécurité de base requise pour un système de chiffrement est la *one-wayness*, qui signifie grosso modo qu'à partir du chiffré, on ne peut pas retrouver le message clair en entier.

Définition 11 ***One-Wayness.*** *Un système de chiffrement à clé publique est dit* one-way *si aucun attaquant probabiliste en temps polynomial ne peut retrouver le message clair à partir d'un chiffré donné, avec une probabilité non négligeable. De manière formelle, on écrit qu'un système de chiffrement asymétrique est (t, ε)-OW si pour tout adversaire $\mathcal{A}$ ayant un temps t borné, sa probabilité d'inversion est inférieure à ε:*

$$\mathsf{Succ}^{ow}(\mathcal{A}) \stackrel{def}{=} \Pr_{m \stackrel{R}{\leftarrow} \mathcal{M}, r}\left[(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{K}(1^k) : \mathcal{A}(\mathcal{E}_{\mathsf{pk}}(m; r)) \stackrel{?}{=} m\right] < \varepsilon$$

où la probabilité est aussi prise sur les randoms r de chiffrement de l'adversaire.

Une propriété de plus en plus demandée aujourd'hui est la *sécurité sémantique* [98] aussi connue sous le nom d'*indistinguabilité des chiffrés* ou *sécurité polynomiale* car elle correspond à la sécurité parfaite dans le modèle calculatoire.

Définition 12 ***Sécurité sémantique.*** *Un système de chiffrement à clé publique est dit* sémantiquement sûr *si aucun attaquant probabiliste en temps polynomial ne peut apprendre un seul bit d'information sur le clair à partir du chiffré, excepté sa longueur. De manière formelle, on écrit qu'un système de chiffrement asymétrique est (t, ε)-IND si pour tout adversaire $\mathcal{A} = (A_1, A_2)$ ayant un temps t borné,*

$$\mathsf{Adv}^{ind}(\mathcal{A}) \stackrel{def}{=} 2\times Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{K}(1^k),\\ (m_0, m_1, s) \leftarrow A_1(\mathsf{pk}),\\ c \leftarrow \mathcal{E}_{\mathsf{pk}}(m_b; r) : A_2(c, s) \stackrel{?}{=} b\end{array}\right] - 1 < \varepsilon$$

où la probabilité est prise sur les randoms de l'adversaire et les messages m_0 et m_1 de taille identique dans l'espace des messages $\mathcal{M}$.

Remarque 3 *Cette notion n'est valable que pour des schémas de chiffrement probabilistes car dans le cas déterministe, il est aisé de distinguer un chiffré de m_0 et un chiffré de m_1 en effectuant le chiffrement.*

Une autre notion de sécurité a aussi été définie : la *non-malléabilité* [64].

Définition 13 ***La non-malléabilité.*** *La non-malléabilité (NM) formalise l'incapacité d'un adversaire à obtenir un nouveau chiffré y' à partir d'un chiffré y de telle sorte que les clairs x et x' des chiffrés y et y' soient significativement reliés.*

Une relation entre les clairs peut être $x' = x+1$, par exemple. Cette notion capture la notion de résistance du chiffré à une tentative de modification. Nous ne détaillons pas cette notion car elle a été prouvée équivalente à la sécurité sémantique contre des attaques parallèles [12].

Le jeu de la sécurité sémantique

On peut représenter le jeu d'un adversaire CPA à travers le schéma de la figure 1.4 :

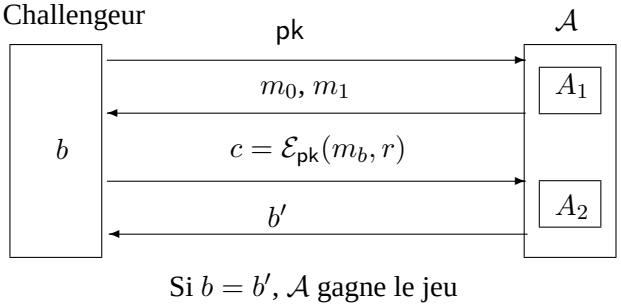


FIG. 1.4: *Jeu de la sécurité sémantique contre des attaques CPA.*

Le jeu de la sécurité sémantique est le suivant. L'adversaire $\mathcal{A}$ est composé de deux algorithmes (A_1, A_2) et obtient la clé publique du système. L'adversaire exécute l'algorithme A_1 qui retourne deux messages m_0 et m_1. Un challengeur tire au hasard un bit b et chiffre m_b avec E pour obtenir c. Il envoie ensuite (m_0, m_1, c) à l'adversaire qui exécute A_2 sur (m_0, m_1, c). L'algorithme A_2 retourne un bit b'. Si la probabilité de succès de l'adversaire diminuée de la probabilité de deviner au hasard (à pile ou

face = $1/2$) est non négligeable (supérieure à l'inverse de la valeur d'un polynôme Q en le paramètre de sécurité), on dit alors que le chiffrement n'est pas sûr. Dans ce cas, l'algorithme $\mathcal{A} = (A_1, A_2)$ est un distingueur des chiffrés de m_0 et m_1.

En d'autres termes, la sécurité sémantique dit qu'il est impossible en temps polynomial en k qu'un algorithme trouve deux messages m_0, m_1 sur lesquels il puisse en temps polynomial distinguer entre $c \in \mathcal{E}_{\mathsf{pk}}(m_0; r)$ et $c \in \mathcal{E}_{\mathsf{pk}}(m_1; r)$. Il est équivalent de dire que le chiffrement de m_0 est indistinguable du chiffrement de m_1 au sens de l'indistinguabilité polynomiale[24].

Relation entre les notions de sécurité pour les cryptosystèmes

Nous présentons maintenant les relations entre les attaques et les notions de sécurité. Ceci permettra de mieux appréhender la sécurité que nous voulons atteindre.

Un moyen commode d'organiser la définition des chiffrements sécurisés est de considérer séparément les buts et les modes d'attaques. On obtient ainsi chaque définition comme le couple d'un but et d'un mode d'attaque particulier.

On peut mixer les buts NM, IND, OW et les modes d'attaques CPA, CCA1, CCA2 pour obtenir les neuf combinaisons : OW-CPA, OW-CCA1, OW-CCA2, IND-CPA, IND-CCA1, IND-CCA2, NM-CPA, NM-CCA1, NM-CCA2.

Le schéma 1.5 présente les théorèmes suivants dus principalement à Bellare, Desai, Pointcheval et Rogaway [5]. Le premier théorème important est que la non-malléabilité contre une attaque adaptative à chiffrés choisis est équivalente à la sécurité sémantique. Aujourd'hui, on considère que la notion de sécurité la plus forte est IND-CCA2. Une flèche pleine signifie qu'il y a une implication et une flèche

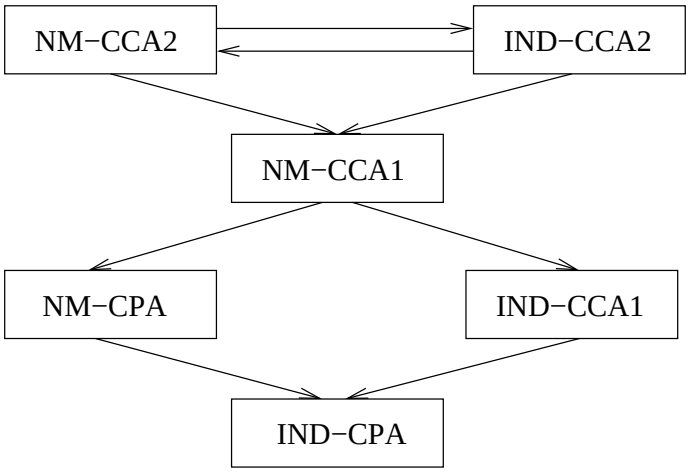


FIG. 1.5: *Relation entre les objectifs et la force des adversaires.*

barrée signifie que l'implication est fausse.

Par exemple, si un schéma est sûr contre les attaques à chiffrés choisis tentant de malléer le clair, alors le schéma est aussi sémantiquement sûr contre les attaques à chiffrés choisis. Ou bien, si un schéma est sûr contre les attaques à clairs choisis cherchant à malléer le clair, alors le schéma n'est pas obligatoirement sémantiquement sûr contre les attaques à chiffrés choisis non-adaptatives.

24. On verra la notion d'indistinguabilité au chapitre 2.

Pour éclaicir ces notions, donnons quelques exemples basés sur le chiffrement RSA sans ajouter de padding particulier comme le propose PKCS#1 v1.5 ou PKCS#1 v2 avec OAEP [10].

Remarque 4 ***Le chiffrement RSA ne cache pas d'information partielle.*** *Si un attaquant connaît l'espace des messages dans lequel Alice choisit ses clairs et que celui-ci est petit, alors un attaquant peut retrouver le clair. En effet, si Alice doit choisir entre* 0 *ou* 1*, un adversaire, Eve, peut refaire le même chiffrement et en comparant les chiffrés, elle retrouve le clair. Ceci provient du fait que RSA est un système de chiffrement déterministe.*

Remarque 5 ***Le chiffrement RSA est malléable.*** *Supposons qu'Alice veuille envoyer à Bob un ordre de virement. Alors en voyant passer le chiffré* c *de* m*, Eve peut le malléer en chiffrant* $1/2$*. L'ordre de virement que recevra Bob sera alors* $c' = c(1/2)^e$ *et Eve aura réussi à malléer* m *en* $m' = m/2$*.*

De même, dans un protocole d'enchères, les utilisateurs chiffrent un montant et un protocole détermine le montant maximum sans déchiffrer toutes les enchères. Dans ce cas, tous les utilisateurs voient passer les montants chiffrés et on ne doit pas pouvoir calculer $c' = \mathcal{E}(m+1)$ *après avoir vu* $c = \mathcal{E}(m)$*.*

Remarque 6 ***Le chiffrement RSA n'est pas sécurisé contre les attaques à chiffrés choisis.*** *Dans ces attaques, un adversaire veut déchiffrer un chiffré "cible"* c*. Cependant, Bob ne veut pas donner le clair à Eve pour certaines raisons. Pourtant, Bob accepte de déchiffrer pour Eve d'autres chiffrés. En particulier, Eve peut générer un chiffré* $c' = c \times x^e \bmod N$*, pour un nombre* x *et elle peut demander à Bob de le déchiffrer pour elle. Elle obtient ainsi* m' *et peut calculer* $m = m'/x \bmod N$*.*

Les deux dernières remarques proviennent de la multiplicativité du cryptosystème RSA.

Ces attaques peuvent toutefois être contrées en utilisant un chiffrement randomisé comme RSA PKCS#1 qui revient à encoder les données sous un certain format avant d'appliquer l'algorithme RSA. Le formatage des données brise la relation de multiplicativité car le format n'est pas multiplicatif : $f(x) \times f(y) \neq f(xy)$. Le clair se présente alors sous la forme $f(m, r)$, où r est une chaîne aléatoire et f une fonction non cryptographique facile à calculer pour tout le monde. On n'utilise jamais RSA tel quel car l'hypothèse de "one-wayness" du problème RSA dit qu'il est difficile en temps "raisonnable" à partir d'un $y \in \mathbb{Z}_N^*$ quelconque de trouver un x en entier tel que $y = x^e \bmod N$, mais ne dit rien sur la difficulté de calculer un bit de x par exemple ce qui casserait la sécurité sémantique. Le but du padding RSA est de randomiser l'instance sur laquelle appliquer la fonction RSA, de façon à pouvoir utiliser le problème RSA pour un y pris au hasard dans $\mathbb{Z}_N^*$ et non dans un sous-ensemble de petite taille.

Exemple : la sécurité sémantique du cryptosystème El Gamal

Dans cette section, nous allons montrer que le cryptosystème El Gamal est sémantiquement sûr sous l'hypothèse que le problème DDH est difficile. Pour ce faire, nous allons utiliser une preuve par réduction, en montrant qu'un adversaire $\mathcal{A}$ qui sait casser la sécurité sémantique du schéma El Gamal peut être utilisé pour construire un attaquant $\mathcal{B}$ qui sait casser le problème DDH réputé difficile. On conclut en disant que comme le problème DDH est un problème difficile, un tel attaquant $\mathcal{B}$ n'existe pas, en conséquence de quoi l'adversaire $\mathcal{A}$ n'existe pas non plus.

On peut conclure car la réduction du problème DDH au jeu de la sécurité sémantique du cryptosystème El Gamal peut s'effectuer en temps polynomial. Ainsi, si on avait un tel adversaire qui sache résoudre le jeu de la sécurité sémantique, alors, on pourrait l'utiliser pour construire un algorithme qui s'exécuterait en temps polynomial et qui utiliserait l'adversaire $\mathcal{A}$ comme une boîte noire.

La figure 1.6 représente comment utiliser l'adversaire $\mathcal{A}$ afin de construire un attaquant $\mathcal{B}$ pour résoudre le problème DDH. Il est ensuite immédiat de vérifier que cette réduction fonctionne en temps polynomial. Le temps d'exécution de $\mathcal{B}$ est borné par la même constante T que le temps de $\mathcal{A}$.

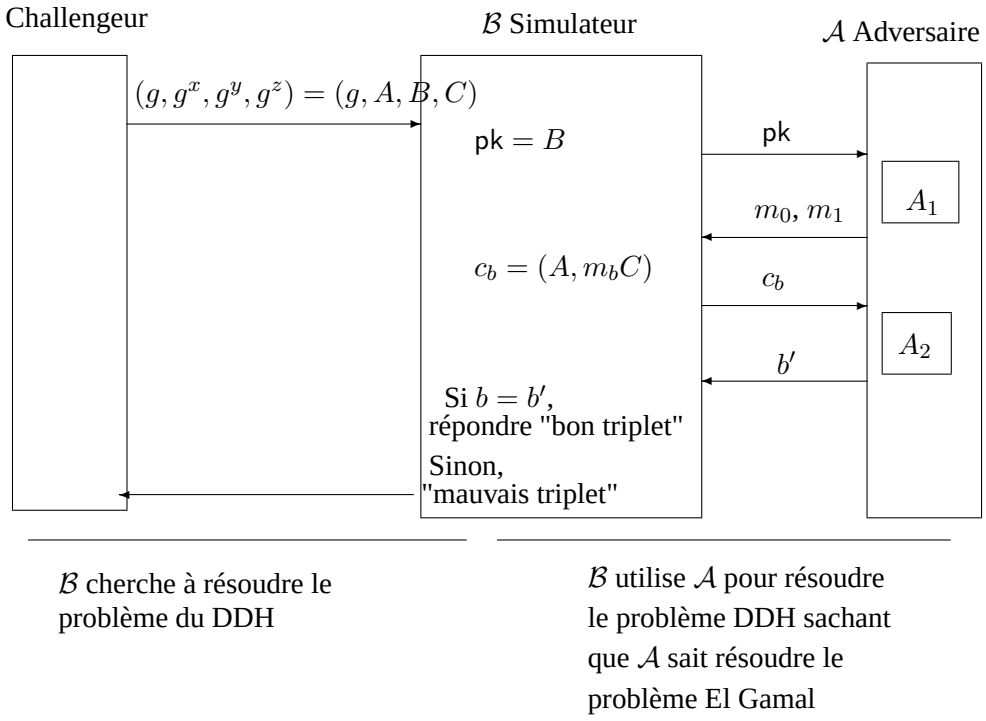


FIG. 1.6: *Preuve de sécurité du cryptosystème El Gamal.*

Soit ε l'avantage de l'adversaire $\mathcal{A}$ pour casser la sécurité sémantique du cryptosystème El Gamal. Cherchons avec quelle probabilité en voyant un triplet DDH ou non, l'attaquant $\mathcal{B}$ est un bon distingueur. Si le triplet est un vrai triplet DDH, alors le simulateur $\mathcal{B}$ gagne le jeu DDH avec la même probabilité que $\mathcal{A}$. Dans le cas contraire, le distingueur $\mathcal{B}$ se trompe en répondant "bon triplet" lorsque le triplet est effectivement un triplet DDH mais pour lequel l'adversaire n'est pas arrivé à casser la sécurité sémantique. Ainsi, $\mathcal{B}$ est un distingueur de triplet DDH avec la même probabilité ε.

Remarque sur la sécurité sémantique du cryptosystème El Gamal

On peut remarquer que si l'on n'impose pas que les clairs soient pris dans le sous-groupe $\langle g \rangle$, alors le cryptosystème El Gamal peut être cassé sans que le problème DDH le soit.

En effet, soit un groupe d'ordre $2q$ engendré par g dans $\mathbb{Z}_p^*$. L'élément g^2 engendre un sous-groupe cyclique d'ordre q. Désignons par $(A, B) = (g^r, y^r.m)$ un chiffré El Gamal. L'attaquant choisit un message m_0 qui est un carré modulo p et m_1 qui n'est pas un carré. Le sous-groupe des éléments quadratiques et les éléments non-quadratiques génèrent tout le groupe $\langle g \rangle = \mathbb{Z}_p^*$. Pour tester si un nombre est un non-residu quadratique il suffit de l'élever à la puissance q modulo p. Si l'on obtient 1, le nombre est clairement un carré, sinon c'est un non-residu quadratique car pour tout $x \in \mathbb{Z}_p^*$, $x^{p-1} = 1 \bmod p$.

L'attaquant gagne le jeu de sécurité sémantique s'il arrive à distinguer si $c = \mathcal{E}(m_b; r)$ contient le chiffré de m_0 ou celui de m_1. On peut distinguer plusieurs cas :

1. Si y est un non-residu quadratique et
   - si r est pair (que l'on peut tester en calculant le symbole de Legendre[25] de $A(= g^r)$), alors

25. Le symbole de Legendre de x modulo p, noté $(x|p)$ vaut $x^{\frac{p-1}{2}} \bmod p$ et indique si x est un résidu quadratique $(x|p) = 1$

y^r est un carré. Dans ce cas, si $B(= m_b.y^r)$ chiffre m_0, alors B est un carré, sinon B n'en est pas un. On teste alors en calculant le symbole de Legendre de B.

- si r est impair (que l'on peut tester en calculant le symbole de Legendre de A), alors y^r n'est pas un carré et si $B = m_0 y^r$ n'est pas un carré alors que si $B = m_1 y^r$, alors B est un carré.

2. Si y est un résidu quadratique alors y^r est un carré et si B chiffre m_0, alors B est un carré, sinon B n'en est pas un.

En conclusion, en testant la résiduosité de B et de A on peut distinguer entre le chiffré de m_0 et celui de m_1 et donc gagner le jeu de la sécurité sémantique.

1.4.5 Sécurité d'un schéma de signature

Dans cette sous-section, nous définissons un schéma de signature et nous en donnons quelques exemples.

Définition d'un schéma de signature

Définition 14 ***Schéma de signature.*** *Un schéma de signature se compose d'un triplet d'algorithmes* $(\mathcal{K}, \mathcal{S}, \mathcal{V})$.

- *L'algorithme de génération des clés* $\mathcal{K}$ *est un algorithme probabiliste en temps polynomial qui prend en entrée un paramètre de sécurité* k*, que l'on écrit souvent en notation unaire* 1^k*, et produit une paire* $(\mathsf{pk}, \mathsf{sk})$ *où* pk *est appelé la clé publique et* sk *la clé secrète. On écrit* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}(1^k)$ *pour indiquer que la paire* $(\mathsf{pk}, \mathsf{sk})$ *est produite par l'algorithme* $\mathcal{K}$.

- *L'algorithme de génération de signature* $\mathcal{S}$ *est un algorithme probabiliste en temps polynomial qui prend en entrée un paramètre de sécurité* 1^k*, une clé secrète* sk *choisie dans* $\mathcal{K}(1^k)$*, et un message* $m \in \{0,1\}^k$*, et produit une chaîne* s *que l'on appelle la signature de* m*. On utilise la notation* $s = \mathcal{S}_{\mathsf{sk}}(m; r)$ *si l'algorithme de signature est probabiliste et* $s = \mathcal{S}_{\mathsf{sk}}(m)$ *sinon.*

- *L'algorithme de vérification de signature* $\mathcal{V}$ *est un algorithme déterministe et en temps polynomial qui prend en entrée une clé publique* pk*, une signature* s*, et un message* m*, et retourne* 1 *(i.e. "vrai") ou* 0 *(i.e. "faux") pour indiquer si la signature est valide ou non. On demande que* $\mathcal{V}_{\mathsf{pk}}(s, m) = 1$ *si* $s \in \mathcal{S}_{\mathsf{sk}}(m)$ *et* 0 *sinon.*

Un schéma de signature est caractérisé par sa sécurité contre un adversaire probabiliste en temps polynomial.

Attaques contre les schémas de signature

On distingue trois types d'attaques que l'on peut classer suivant l'ordre croissant de force :

- *Attaque avec la clé publique uniquement* : Dans ce type d'attaque, l'adversaire connaît seulement la clé publique du signataire et a par conséquent la capacité de vérifier les signature des messages qu'il reçoit.

- *Attaque avec signature connue* : L'adversaire connaît la clé publique du signataire et a vu plusieurs couples messages/signatures choisis et produits par le signataire légal.

ou non $(x|p) = -1$. En effet, si x est un carré modulo p, il existe y tel que $x = y^2 \bmod p$, et $x^{\frac{p-1}{2}} = y^{2(\frac{p-1}{2})} = y^{p-1} = 1$ d'après le petit théorème de Fermat. Le calcul du symbole de Legendre peut être effectué rapidement en $\mathcal{O}(n^3)$, où $n = |p|$.

– *Attaque avec des messages choisis* : L'adversaire peut demander au signataire de signer un nombre de messages de son choix. Le choix de ces messages peut dépendre des signatures précédemment obtenues (cf. [101]).

Que signifie forger une signature avec succès ?

On peut distinguer plusieurs niveaux de succès pour un adversaire :

– *Forge existentielle* : L'adversaire peut réussir à forger la signature d'un message, mais pas nécessairement de son choix.

– *Forge sélective* : L'adversaire peut réussir à forger la signature de quelques messages de son choix.

– *Forge universelle* : L'adversaire, bien qu'incapable de trouver la clé secrète du signataire légal, est capable de forger la signature de n'importe quel message.

– *Cassage total* : L'adversaire peut calculer la clé secrète de signature.

Prouver la sécurité d'un schéma de signature

Quand on cherche à construire un schéma de signature, on souhaite prouver sa sécurité en utilisant la notion de sécurité la plus forte pour les schémas de signature. Il convient donc de proposer un schéma résistant aux *forges existentielles* contre une attaque à messages choisis.

On peut définir cette notion avec le jeu suivant qu'essaie de résoudre l'adversaire. Un couple de clé publique/privée $(\mathsf{pk}, \mathsf{sk})$ pour le schéma de signature est généré et l'adversaire obtient la clé publique pk. L'adversaire $\mathcal{A}$ fait une suite de requêtes de signature sur des messages m_i de son choix qui peuvent être adaptativement choisis, *i.e.* dépendre des messages précédents. L'adversaire gagne le jeu s'il peut forger une signature, c'est-à-dire, trouver une signature pour un message m différent des messages m_i qui ont été posés.

Pour montrer la sécurité du schéma construit, on cherchera comme toujours à faire une réduction entre un adversaire et un problème difficile. Dans la réduction, on pourra utiliser l'adversaire comme un oracle et lui demander de forger une signature. Ensuite, avec cette signature, on essaiera de résoudre une instance du problème difficile.

Exemple : le schéma de signature RSA

Dans le schéma de signature RSA, la clé publique est un couple d'entiers (N, e), où N est le produit de deux grands nombres premiers et e est relativement premier avec $\varphi(N)$. La clé secrète d est telle que $ed = 1 \bmod \varphi(N)$. Signer consiste à calculer $\sigma(m) = m^d \bmod N$. Vérifier consiste à élever la signature à la puissance e et comparer le résultat avec le message originel.

Remarque 7 *Le schéma de signature RSA est universellement forgeable par une attaque à messages choisis et existentiellement forgeable par un attaque à messages connus.*

Remarque 8 ***Sécurité du schéma de signature RSA-FDH ou RSA-PSS.*** *Il est important de voir que le schéma de signature RSA n'est donc jamais utilisé tel quel vu la remarque 7. En général, on utilise le modèle hash-and-sign qui consiste à calculer initialement un haché sur le message à signer avec une fonction de hachage, telle que SHA-1. La taille du message à signer est alors de taille fixe, 160 bits. Si on applique RSA^{-1}, $RSA^{-1}(y) = y^d \bmod N$, sur ce message de petite taille, est-ce que le schéma résultant est sûr ? Autrement dit, si on a un adversaire attaquant le schéma de signature avec une attaque*

à messages choisis, peut-on l'utiliser pour casser le problème RSA ? Il ne semble pas être possible de prouver la sécurité de ce schéma et même une adaptation des attaques [52] tend à montrer que ce type de schéma n'est pas sûr.

La société RSA Security a proposé un padding, appelé PKCS # 1 pour la signature qui est similaire au hash-and-sign model, mais où les bits de poids fort du message à signer sont positionnés à "1". Personne n'a pu montrer la sécurité de ce format. Cependant, il semble difficile de faire une preuve de sécurité car le domaine des messages à signer, S a une taille de 2^{160} et la fonction RSA a un domaine de taille supérieure à 2^{1023}. Ainsi, S est de taille petite par rapport à $\mathbb{Z}_N^*$. Par conséquent, il se pourrait que le problème RSA soit "facile", c'est-à-dire résoluble en temps "raisonnable" sur une instance prise au hasard dans cet espace, S, mais soit "difficile" pour une instance quelconque. En effet, la probabilité de tomber sur un message à signer est négligeable, ($\frac{2^{160}}{2^{1023}} = 2^{-863} < N^{-1/2}$).

Dans le cas de la signature RSA avec padding FDH (Full Domain Hashing) ou padding PSS (Probabilistic Signature Scheme), on montre dans le modèle de l'oracle aléatoire qu'un adversaire qui sait forger une signature avec une attaque à messages choisis adaptative, peut être utilisé pour résoudre le problème RSA (cf. [11, 51]).

Le padding FDH consiste à utiliser une fonction de hachage qui a une sortie dans l'ensemble $\{0,1\}^k$ si k est la taille du module RSA, alors que le padding PSS est plus compliqué. Cependant, dans ce dernier cas, la réduction entre un forgeur qui sait casser la sécurité du schéma de signature avec une attaque à messages choisis d'une part, et d'autre part, la résolution d'une instance du problème RSA, a le même taux de réussite. Dans le cas de la réduction avec le schéma de signature FDH, il y a une perte qui est de l'ordre du nombre de signatures créées.

2

Généralités sur la cryptographie partagée

Le but de la cryptographie partagée est de proposer des protocoles permettant de protéger un schéma cryptographique contre des adversaires plus forts que ceux considérés dans le chapitre précédent et d'éviter qu'une seule personne ait le pouvoir de déchiffrer ou signer.

Les adversaires considérés dans cette thèse peuvent attaquer des machines, *i.e.* avoir accès au contenu de la mémoire et donc à la clé secrète si cette dernière y est conservée. En effet, dans la pratique les intrusions extérieures (attaques via le réseau informatique) ou intérieures (corruption de certaines personnes internes à l'entreprise ou via le réseau interne) peuvent donner accès à la clé secrète. Une solution consiste à partager la clé secrète entre plusieurs acteurs ou machines de sorte que le secret ne soit jamais contenu dans une seule machine à un instant donné. Si l'adversaire veut obtenir la clé secrète, il devra attaquer plusieurs machines. De même que les adversaires étudiés précédemment, ces adversaires tentent de contredire la sécurité du protocole d'une part. D'autre part, ils cherchent à empêcher que les services cryptographiques de génération de clé, de signature ou de chiffrement, se terminent correctement. En effet, dans le cas distribué, les acteurs doivent s'échanger des messages et certains adversaires ont le pouvoir de modifier le contenu de la mémoire si ces derniers ont aussi le droit d'écriture. Le but de la cryptographie partagée est alors de concevoir des protocoles résistant à ces adversaires qui, en plus d'un attaquant *classique*, ont la possibilité de corrompre certains acteurs et d'avoir accès à certaines parties de la clé secrète. Ces protocoles permettent aussi de protéger une fonction cryptographique contre la révélation d'une partie de la clé puisque l'on autorise l'adversaire à connaître plusieurs parts de la clé secrète distribuée.

Dans une première section, nous montrerons que la cryptographie partagée s'inscrit dans un domaine plus vaste de la cryptographie qui concerne le calcul multiparties. Puis, nous décrivons des outils pour résister à différents types d'adversaire et nous donnerons les modèles de sécurité de la cryptographie partagée. Enfin, nous aborderons le cas des attaquants mobiles.

Sommaire

2.1 Introduction à la cryptographie partagée

Comme on l'a définie, la cryptographie a pour but d'assurer la sécurité des communications et des données stockées en présence d'un adversaire.

Le premier avantage de la cryptographie partagée est de résister à des adversaires plus forts que ceux considérés dans le premier chapitre et qui peuvent avoir accès au contenu de la mémoire de certaines machines. La deuxième caractéristique de ces adversaires est qu'ils peuvent attaquer un certain nombre de machines, disons t parmi n. Le nombre t est parfois appelé le "seuil" du système et on parle alors de cryptographie à seuil. Il est réaliste de considérer ces adversaires puisque les intrusions réseaux sont faciles à monter principalement à cause de l'architecture "ouverte" du réseau Internet. Ainsi, un adversaire peut remonter le réseau et entrer dans le réseau interne d'une entreprise. Ensuite, suivant les systèmes d'exploitation utilisés, il peut pénétrer sans trop de difficulté dans une machine et obtenir toutes les clés cryptographiques présentes. Par ailleurs, des produits de détection d'intrusion (Intrusion Detection System, IDS) existent. Malheureusement ces outils ne sont pas *préventifs* mais plutôt *curatifs*, c'est-à-dire que l'attaque peut déjà avoir eu lieu lorsque l'on détecte la présence d'un adversaire. En revanche, si le nombre de machines attaquées est inférieur au seuil t, on peut concevoir des systèmes tels que même si un adversaire obtient t parts de la clé, il n'obtient aucune information sur la clé. Enfin, la cryptographie à seuil est limitée car si l'attaquant a suffisamment de temps, il peut entrer dans les machines une à une, et compromettre la sécurité du système. Ce risque est particulièrement problématique dans les systèmes qui doivent rester sûrs pendant de longues périodes de temps comme les autorités de certification. On peut alors considérer des adversaires mobiles qui peuvent attaquer toutes les machines mais seulement t par période de temps, disons tous les mois. Ces adversaires sont appelés *mobiles* car à chaque période de temps, ils peuvent se déplacer et attaquer des machines différentes. On parle de cryptographie proactive car avant que le seuil des machines attaquées ne soit atteint, le système va repartager la clé. La sécurité proactive offre des mécanismes de protection à long terme d'un système contre de telles attaques. Elle utilise une technique de partage avec un rafraîchissement périodique du partage de la clé de telle sorte que l'information apprise pendant les périodes passées n'aide pas l'attaquant pendant la période suivante.

Le deuxième avantage des schémas partagés est qu'ils sont utilisés pour des applications critiques où l'on souhaite éviter qu'une seule personne puisse déchiffrer ou signer. Les applications peuvent être le protocole de dépouillement d'un schéma de vote électronique (déchiffrement du résultat), le recouvrement des clés des utilisateurs lorsqu'ils ont perdu leur clé privée (déchiffrement de la clé privée), ou encore le protocole de signature d'une autorité de certification où la clé racine représente la clé de voûte du système de confiance.

Initialement, la cryptographie partagée a permis de *distribuer* une clé cryptographique particulièrement "sensible" entre n personnes de telle sorte qu'aucune d'entre elles ne connaisse entièrement la

clé, mais certains sous-ensembles de ces personnes puissent la *reconstruire*. Les premiers schémas de partage de secret sont apparus à la fin des années 70 [170, 19]. L'objectif de sécurité visé est le même que celui pour résoudre le problème de l'ouverture du coffre fort dans une banque. Soit par exemple une banque qui emploie trois personnes, mais ne souhaite confier la combinaison du coffre à aucune d'entre elles. Le but est d'avoir un système d'accès tel que toute association de deux employés soit capable d'ouvrir le coffre, mais qu'aucun d'entre eux ne le puisse individuellement pour éviter les tentatives de vol. Dans ce cas, la cryptographie offre un moyen de forcer la réunion de personnes pour effectuer certaines actions.

Les schémas proposés à l'origine permettent certes de reconstruire la clé secrète à partir d'un partage initial de la clé, mais ne permettent pas toujours d'initialiser correctement le processus, c'est-à-dire de *distribuer* correctement la clé. En général, ces schémas utilisent un distributeur unique qui génère la clé secrète, la partage et envoie une part de la clé à toutes les autres personnes. Ce joueur particulier est appelé *distributeur de confiance* ou *trusted dealer*, car tous les utilisateurs lui font confiance pour qu'il ne dévoile pas la clé qu'il a générée. On suppose alors que l'adversaire ne peut pas attaquer cette machine. On peut la déconnecter du réseau pour éviter les intrusions externes et internes et on suppose alors que le distributeur ne peut pas être corrompu. Il transmet ensuite les parts à chaque machine au moyen d'une disquette par exemple. Le problème de cette technique est que rien ne garantit les autres acteurs qu'ils pourront reconstruire la clé si le distributeur triche et génère un mauvais partage. Ainsi, la disponibilité du service risque d'être mise en défaut. La solution consiste à rajouter dans le protocole des messages permettant aux acteurs de vérifier si le distributeur agit correctement dans la phase de partage. Il doit alors prouver que les parts transmises à chaque joueur permettent de reconstruire la clé. Par conséquent, cet acteur doit être une machine connectée au réseau à moins que la preuve puisse être vérifiée sans interaction. Ces schémas sont appelés partages de secret vérifiables. Ils prennent le nom de partages de secret publiquement vérifiables quand non seulement les participants peuvent vérifier que le distributeur de confiance a correctement agi mais qu'un observateur externe quelconque le peut également. Les premiers protocoles ayant ces propriétés sont [47, 68]. Ils permettent donc de partager une clé cryptographique de manière sûre.

Le dernier problème à résoudre est le suivant. Bien que ces protocoles permettent de *partager* et *reconstruire* une clé, en cryptographie, une clé secrète sert à signer et déchiffrer ou comme germe d'un générateur aléatoire. La reconstruction de cette clé dans une seule machine risque de compromettre sa protection. Même si des schémas de reconstruction et d'utilisation de la clé sont possibles, la clé sera "en danger" à chaque utilisation. En effet, l'attaquant qui peut corrompre des machines (en ayant accès à la mémoire) n'en a qu'une à attaquer. De même que précédemment, si cette machine qui reconstruit la clé est sur le réseau, alors un risque d'intrusion réseau est possible. Si elle n'est pas sur le réseau, les risques peuvent venir d'une *coalition* de personnes ou d'une intrusion locale sur la mémoire de la machine par un programme déjà présent. Il ne faut pas croire que le redémarrage de la machine soit suffisant pour contrer une attaque sur la mémoire vive car des attaques par cheval de Troie[26] sont toujours possibles. Une solution consiste à contrôler tous les processus exécutés sur la machine. De plus, il faudrait être certain que les appels systèmes du programme de reconstruction sont ceux d'origine et qu'aucun d'entre eux ne comporte de cheval de Troie. Bref, toutes ces suppositions font qu'une autre solution est souhaitable pour garantir la sécurité d'un système. Une solution apportée par la cryptographie partagée est le partage de fonction. En effet, tous les processus de cryptographie à clé publique peuvent être vus comme des évaluations de fonctions. Une signature, un déchiffrement, la génération d'une clé sont des fonctions mathématiques qui prennent en entrée la clé secrète, des données et une chaîne de bits aléatoires et qui

26. Un cheval de Troie est un morceau de code malicieux présent dans un programme informatique apparemment inoffensif. Son nom fait référence au cheval de Troie mythologique, cadeau des Grecs aux Troyens et qui a permis la prise de la ville de Troie.

retournent une signature ou un message déchiffré sous forme d'entiers.

La théorie du calcul *multiparties* ou *interactif* a montré que n'importe quelle fonction pouvait être calculée par plusieurs joueurs[27] de manière sûre. C'est-à-dire, pour tout adversaire, si n joueurs ont chacun une entrée x_i, il existe un protocole tel que pour tout i, le joueur i peut obtenir $y_i = f_i(x_1, \ldots, x_n)$ et sans qu'aucune autre information sur les entrées x_j des autres joueurs ne soit révélée. Un cas particulier est celui où les joueurs obtiennent $y = f(x_1, \ldots, x_n)$, $y_1 = \ldots = y_n = y$, et aucun joueur i n'apprend plus d'information sur les x_j, $(j \neq i)$ au-delà de y.

Le premier protocole est utile pour générer une clé cryptographique de manière partagée. En effet, initialement chaque joueur tire un random r_i et pose $x_i = r_i$ et à la fin du protocole chaque joueur obtient une partie de la clé y_i. Le second est utile pour calculer une fonction de signature ou de déchiffrement. Initialement, chaque joueur a une partie x_i de la clé de signature et à la fin du protocole, tous les joueurs peuvent calculer la signature y. Tous les joueurs peuvent connaître la signature ou le message déchiffré, mais aucune autre information sur la part de la clé x_i des autres joueurs n'est dévoilée.

Soit par exemple un protocole d'authentification où une personne, représentée par une clé secrète, cherche à prouver son identité (connaissance de la clé secrète) à un serveur (distributeur automatique de banque) de sorte qu'aucune information sur le secret au-delà de sa connaissance ne puisse être obtenue par le serveur. Si l'utilisateur doit transmettre son identifiant pour s'authentifier, un mauvais serveur pourrait exploiter cette information pour ultérieurement se faire passer pour l'utilisateur légal. Par exemple, considérons le mécanisme d'authentification "login/password". Il est aisé de développer un programme reproduisant l'invite d'une station de travail. Un attaquant va alors laisser ce programme sur une machine sans se délogger. Un utilisateur cherche à s'authentifier à ce programme de leurre en rentrant son login et son mot de passe. Le programme intercepte alors le mot de passe et délogge l'utilisateur précédent. L'utilisateur non attentif voit revenir l'invite classique. Il croira qu'il a mal tapé son mot de passe et le rentrera à nouveau. Cet exemple prouve que dans un processus d'authentification, si l'utilisateur doit transmettre sa clé pour prouver son identité, il donne alors au vérifieur (serveur ou cheval de Troie) un moyen de s'authentifier à sa place. La construction de mécanismes d'authentification nécessite les exigences suivantes : être *non-transmissible* et de ne *révéler aucune information sur l'identifiant de l'utilisateur* ("zero-knowledge") au-delà du fait que l'utilisateur est bien celui qu'il prétend être. Un protocole d'authentification est un exemple de protocole interactif entre deux parties : un prouveur cherche à prouver son identité à un vérifieur. Ils évaluent la fonction $f(x, \omega)$ qui vaut 1 si le prouveur connaît la clé secrète x et 0 sinon, où ω représente une chaîne de bits aléatoires.

2.1.1 Cryptographie interactive

La cryptographie *interactive* ou *calcul multiparties*, en anglais Multiparty Computation (MPC), concerne l'étude *générale* des calculs sécurisés entre plusieurs entités. Elle décrit des protocoles nécessitant des *interactions intensives entre les parties*. D'une part la cryptographie interactive a prouvé de manière constructive des théorèmes sur les calculs réalisables sous forme sécurisée en fonction du nombre des entités corrompues, du comportement de l'adversaire et des hypothèses sur les canaux reliant les entités. D'autre part, la cryptographie à seuil cherche à résoudre des problèmes spécifiques, comme le partage de certaines fonctions à trappe, plus *efficacement* que les protocoles génériques MPC qui permettent de calculer n'importe quelle fonction de manière sécurisée en temps polynomial. Bien que le temps d'exécution de ces protocoles MPC soit polynomial, leur mise en œuvre n'est pas efficace en pratique.

27. Les différentes personnes ayant une partie de la clé sont appelées *serveurs* dans le cas d'un protocole cryptographique puisqu'elles sont souvent implémentées sous forme de serveurs informatiques. Lorsque nous décrirons les protocoles, on parlera aussi de joueurs, participants ou acteurs.

Un cadre général pour présenter des problèmes cryptographiques consiste à spécifier un processus aléatoire qui envoie n entrées $(x_1, \ldots, x_n)$ vers n sorties $(y_1, \ldots, y_n)$. Les entrées de ce processus peuvent être vues comme les entrées locales de n parties, et les n sorties sont leurs sorties locales correspondantes. Le processus aléatoire décrit la fonction désirée. C'est-à-dire, que si les n parties se font mutuellement confiance, ou peuvent faire confiance à une troisième partie externe, alors elles peuvent toutes envoyer leur entrée locale à une tierce partie, qui pourrait calculer la sortie du processus et retourner à chacune d'entre elles la sortie correspondante. Une question essentielle dans ce cas est comment *émuler la tierce partie (imaginaire)* par des parties qui mutuellement ne se font pas confiance.

Que signifie "émuler une tierce partie"? Cette notion est importante pour définir la sécurité des protocoles permettant de calculer entre plusieurs parties et remonte à la définition du "zero-knowledge" (cf. [100]) et du "chiffrement sécurisé" (cf. [124]). Le modèle sous-jacent est qu'un schéma est *sûr* si tout ce qu'un adversaire peut obtenir après l'avoir attaqué peut aussi être obtenu à partir de rien. Comme on l'a dit dans la sous-section précédente, dans le cas d'un protocole d'identification "zero-knowledge", ceci signifie que l'information qu'un vérifieur peut obtenir sur la clé secrète après avoir interagi avec le prouveur est statistiquement ou calculatoirement équivalent à ce qui peut être calculé à partir de la clé publique. Dans le cas du calcul multiparties, on compare l'action d'un adversaire qui participe à l'exécution du protocole réel avec l'action des parties qui participent à l'exécution imaginaire d'un protocole trivial (idéal) pour calculer la fonction désirée avec l'aide d'une tierce partie. Si l'information qu'un adversaire peut obtenir dans le modèle réel peut être obtenue dans le modèle idéalisé, alors le protocole qui "émule une tierce partie" est dit sûr.

2.1.2 Classification des protocoles distribués

On classe les protocoles distribués en fonction de différents critères comme les hypothèses faites sur les canaux de communication entre les participants, les propriétés de sécurité que tente d'attaquer un adversaire ou encore la puissance des adversaires.

Les canaux

Les modèles de communication diffèrent en fonction de trois critères : l'existence de canaux secrets de communication reliant chaque paire de joueurs, l'existence d'un canal de *broadcast* qui permet de transmettre le même message à plusieurs destinataires en même temps, et suivant des contraintes temporelles sur les canaux de communication *synchrones* (un message émis arrive immédiatement au(x) destinataire(s)) ou *asynchrones* (existence d'un délai de transit variable selon le moment d'émission ou la destination).

Propriétés de sécurité d'un schéma partagé

On rappelle que dans les protocoles partagés, l'adversaire a le droit d'obtenir t parts de la clé secrète en attaquant t serveurs. De plus, dans le cas des schémas de signature ou de déchiffrement, le modèle de communication est le suivant. Initialement, chaque serveur a une part de la clé secrète. Lorsqu'un message doit être signé ou déchiffré, un joueur particulier, appelé *combineur*, envoie le message à signer ou à déchiffrer à tous les serveurs. Les serveurs appliquent une fonction dépendant de leur part de la clé et du message, et retournent le résultat au combineur. Dans le cas de la signature, le message renvoyé est appelé *part de la signature* et dans le cas du déchiffrement, *part du déchiffré*. Enfin, le combineur exécute l'algorithme de combinaison pour générer la signature ou déchiffrer le message.

Les deux propriétés que tente de contredire un adversaire contre un schéma multiparties sont :

– la *sécurité du protocole* qui dépend du schéma cryptographique. Dans le cas d'un schéma de

signature, il s'agit de forger une signature contre une attaque à messages choisis adaptative. Ainsi, l'adversaire a accès à t parts de la clé secrète de signature, aux signatures sur les messages m_i de son choix distincts du message à forger et des parts de signatures des joueurs non corrompus sur les messages m_i. La figure 2.1 représente le modèle de sécurité d'un schéma de signature distribué. La valeur pk représente la clé publique, sk_j les parts de la clé de signature connue par l'adversaire, s_i la signature sur le message m_i, et $f(m_i, \mathsf{sk}_j)$ les parts de signature sur le message m_i en utilisant la part de la clé sk_j.

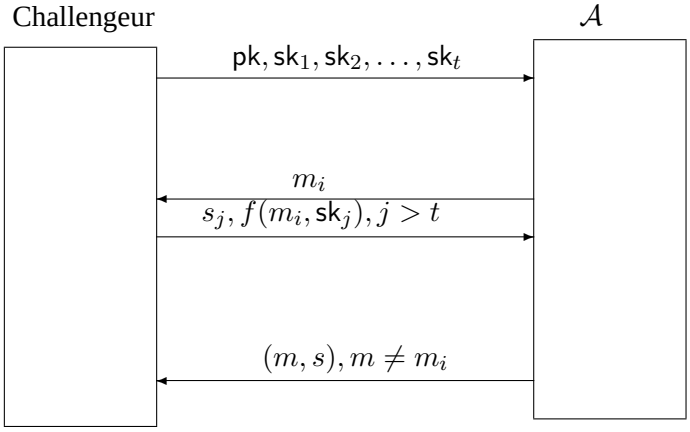


FIG. 2.1: *Modèle de sécurité d'un schéma de signature distribué.*

Dans le cas d'un système de chiffrement, il s'agit d'apprendre de l'information sur le message clair ou de contredire l'indistinguabilité du schéma de chiffrement. L'adversaire a accès à t parts de la clé de déchiffrement, $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_t$, aux parts de déchiffrement des joueurs non corrompus sur les messages σ_i de son choix, $m_i, f(\sigma_i, \mathsf{sk}_j)$, et aux déchiffrés sur les chiffrés de son choix suivant le mode d'attaque (cf. 2.2).

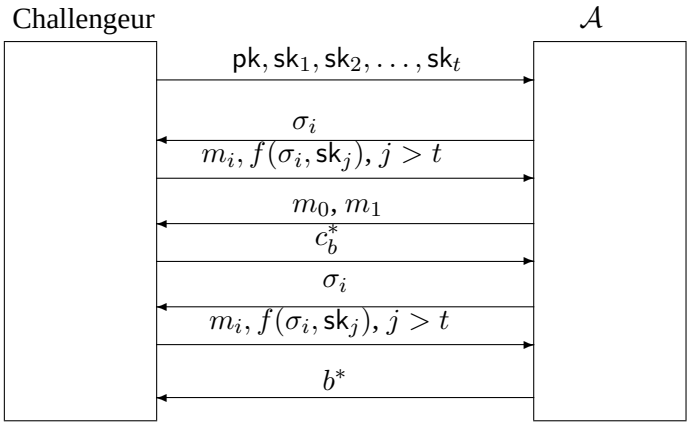


FIG. 2.2: *Modèle de sécurité sémantique d'un système de chiffrement distribué contre une attaque adaptative à chiffrés choisis.*

Dans le cas du partage de la clé, il s'agit d'apprendre de l'information sur une part de la clé détenue par un joueur non-corrompu.

– La *robustesse du protocole* permet de garantir que même en présence de joueurs capables de tricher en envoyant de mauvaises parts, le protocole permettra toujours de générer une signature valide, de déchiffrer correctement un message ou encore de générer une clé avec une distribution uniforme sur l'espace des clés.

Les adversaires

Il existe plusieurs catégories d'adversaires selon :

1. leur capacité d'attaque sur un serveur (droit de lecture du contenu de la mémoire, accès en écriture, envoi de mauvais messages),
2. leurs ressources de calcul,
3. leur méthode pour décider les machines à attaquer.

La première distinction que nous introduirons concerne les adversaires passifs et actifs :

– Un adversaire *passif* peut lire toutes les informations disponibles dans la mémoire des machines attaquées et tente uniquement de contredire la sécurité du protocole et non la robustesse du protocole.

– Un adversaire *actif* contrôle le comportement complet des joueurs corrompus. Il tente de contredire la sécurité du protocole et/ou d'attaquer la robustesse.

Les adversaires sont aussi classés suivant les ressources de calcul disponibles. Si l'adversaire est polynomialement borné, on parlera de sécurité cryptographique ou conditionnelle, et dans le cas où les ressources ne sont pas bornées, on parlera de sécurité au sens de la théorie de l'information ou inconditionnelle.

De plus, on peut distinguer adversaires *statiques* et adversaires *adaptatifs* ou *dynamiques*. Contrairement à un adversaire statique qui corrompt les joueurs au début de l'exécution du protocole, un adversaire adaptatif peut décider de corrompre n'importe quel joueur, au cours de l'exécution du protocole aussi longtemps que le nombre de joueurs corrompus ne dépasse pas une limite, dépendant par exemple des messages échangés. Il est clair qu'un adversaire adaptatif semble plus fort qu'un adversaire statique. Cependant dans certains modèles, il y a équivalence entre adversaires statique et adaptif [31]. Pour étudier ces équivalences, il est utile de mettre en correspondance les adversaires et les modèles de sécurité utilisés : sécurité au sens de la théorie de l'information et sécurité cryptographique. Il est assez intuitif de penser que dans le modèle de la théorie de l'information, il y a équivalence entre ces deux d'adversaires. En effet, l'attaquant n'obtient aucune information lorsqu'il voit passer les messages échangés entre les serveurs non corrompus. Il ne peut donc pas utiliser ces données pour décider quel serveur corrompre. En revanche dans le modèle cryptographique, les communications entre les serveurs non corrompus peuvent permettre à un adversaire puissant de déduire des informations. Ainsi, on suppose que de l'information fuit, mais en quelle quantité ? Si le nombre de serveurs est petit et si l'on ne considère que des adversaires passifs[28], l'adversaire ne peut pas réunir suffisamment d'information et il y a encore équivalence entre ces deux notions. Cependant, dès que $n = \omega(\log(k))$ où k est un paramètre de sécurité, ou si l'on ne considère que des adversaires actifs, il n'y a plus équivalence quel que soit le nombre de serveurs corrompus.

Il est enfin possible de considérer les adversaires *mobiles*. De même qu'un adversaire adaptatif, ils peuvent corrompre un serveur à n'importe quel moment du jeu. Ils peuvent aussi décider de perdre le

28. *Passifs* ne veut pas dire ici que l'adversaire ne fait qu'écouter le canal de communication. On rappelle qu'il a aussi accès en lecture à la mémoire de plusieurs machines.

contrôle sur certains et regagner la capacité de corrompre d'autres joueurs ultérieurement ou les mêmes un instant plus tard. Ces adversaires modélisent le comportement d'un attaquant qui craint d'être détecté et préfère perdre temporairement le contrôle sur une machine pour éviter de se faire remarquer. Dans le cas des protocoles proactifs, le temps est divisé en périodes. Ces adversaires peuvent donc attaquer n'importe quel serveur mais ne peuvent en attaquer que t durant la même période de temps.

2.1.3 Quelques résultats constructifs de cryptographie interactive

Le problème du calcul à plusieurs parties a été étudié pour la première fois par Yao [184]. La première solution générale pour ce problème a été présentée par Goldreich, Micali et Wigderson dans [95] où sont présentés :

- un protocole sûr contre des *attaques passives* permettant à n joueurs de calculer secrètement n'importe quelle fonction même si un adversaire passif peut corrompre $t < n$ joueurs, et
- un protocole sûr contre des *attaques actives* tolérant un adversaire actif qui peut corrompre $t < n/2$ joueurs.

La sécurité de ces deux protocoles est cryptographique, *i.e.* que l'adversaire est supposé polynomialement borné.

Ensuite, Ben-Or, Goldwasser, et Wigderson [13] ont prouvé que dans le modèle des canaux secrets et sans utiliser de canal de broadcast, la sécurité parfaite pour n joueurs peut être atteinte même si l'adversaire peut corrompre n'importe quel sous-ensemble de $n/2$ joueurs (dans le cas passif) ou n'importe quel ensemble de $n/3$ joueurs (dans le cas actif). Ces bornes sont optimales. Les mêmes résultats ont été montrés par Chaum, Crépeau et Damgård dans [43] dans le modèle inconditionnel avec une probabilité d'erreur exponentiellement petite.

Une caractéristique commune des deux articles est l'utilisation du schéma de partage de secret de Shamir [170] et du modèle général de compilateur permettant de transformer un protocole sûr contre des joueurs *semi-honnêtes*[29] en un protocole sûr contre des joueurs malicieux, actifs, en forçant les joueurs à prouver qu'ils se sont comportés comme des joueurs honnêtes. Cependant, les auteurs de [13] utilisent des techniques de la théorie des codes correcteurs d'erreurs, alors que les auteurs de [43] utilisent des schémas de mise en gage (commitment) *distribuée* et de zero-knowledge.

La borne dans le cas actif a été améliorée par Rabin et Ben-Or dans [14] en faisant comme hypothèse supplémentaire l'existence d'un canal de broadcast et en tolérant une probabilité d'erreur négligeable. Ils ont proposé des protocoles qui fournissent une sécurité inconditionnelle contre des adversaires actifs qui peuvent corrompre $t < n/2$ joueurs. Il est intéressant de constater que l'existence d'un canal de broadcast permet d'augmenter le nombre de serveurs que l'attaquant peut corrompre.

Dans le cas $n = 2$, les techniques employées sont différentes et s'appuient sur des schémas dit *oblivious transfer* (OT) [158] qui permettent d'échanger des bits entre deux parties. Ces schémas peuvent être utilisés pour concevoir des protocoles de vente sur Internet. Un marchand souhaite par exemple commercialiser des images (suites de bits) à des acheteurs qui ne veulent pas révéler les images qu'ils achetent.

Plus précisément, un protocole OT est un protocole entre un émetteur S et un receveur R vérifiant les propriétés suivantes : S a deux bits secrets, b_0 et b_1, et R choisit un bit secret s. A la fin du protocole, qui consiste en un nombre d'échanges entre S et R, R obtient le bit b_s mais sans avoir d'information sur b_{1-s} (sécurité de l'émetteur). De plus, S n'obtient aucune information sur le bit s (sécurité du receveur). A partir de cette primitive, on peut concevoir des systèmes sûrs à deux parties.

29. On dit aussi dans le modèle *honnête-mais-curieux*. Les adversaires sont *passifs* et non *actifs*.

2.2 Outils de cryptographie partagée

Dans cette partie, nous introduisons les outils couramment utilisés en cryptographie partagée en définissant tout d'abord différents types de partage. Diverses structures d'accès[30] peuvent être utilisées mais nous ne nous intéressons qu'aux structures (partages) *additives* et *polynomiales*. Dans un partage additif, la structure d'accès est réduite à l'ensemble $\{1, \ldots, n\}$ alors que dans un partage polynomial, elle est composée de tous les ensembles d'au moins $t+1$ joueurs (partage à seuil). Ensuite, nous décrirons le schéma de partage de secret de Shamir, l'algorithme de décodage de Berlekamp-Welch, les protocoles zero-knowledge de preuve de connaissance et d'appartenance à un langage et le partage de secret vérifiable. Ces deux derniers outils permettent de vérifier que les joueurs ont correctement effectué leur travail.

2.2.1 Partage additif

Partage avec des $\oplus$

Pour partager un secret $s \in \{0,1\}^\ell$ parmi n joueurs, le distributeur $\mathcal{D}$ choisit aléatoirement $s_i \in_R \{0,1\}^\ell$, pour $i = 1, \ldots, n-1$, puis définit

$$s_n = s \oplus s_1 \oplus \ldots \oplus s_{n-1}$$

Enfin, $\mathcal{D}$ envoie s_i au joueur i. Ce partage additif est aussi parfois appelé *partage n-parmi-n* : tout sous-groupe d'au plus $n-1$ personnes n'a aucune information sur le secret, mais les n morceaux permettent de reconstruire aisément le secret s :

$$s = s_1 \oplus \ldots \oplus s_n$$

Pour une application de ce partage au recouvrement de clé, voir [125].

Partage modulaire

Pour partager un secret $s \in \mathbb{Z}_N$ entre n joueurs, $\mathcal{D}$ choisit aléatoirement $s_i \in_R \mathbb{Z}_N$, pour $i = 1, \ldots, n-1$, puis définit

$$s_n = s - (s_1 + \ldots + s_{n-1}) \bmod N$$

Enfin, $\mathcal{D}$ envoie s_i au joueur i. Comme dans le cas précédent, il s'agit d'un *partage n-parmi-n* : tout sous-groupe d'au plus $n-1$ personnes n'a aucune information sur le secret, mais les n morceaux permettent de reconstruire aisément le secret s :

$$s = s_1 + \ldots + s_n \bmod N$$

Dans la suite, lorsque nous parlerons de partage additif, il s'agira toujours d'un partage modulaire.

2.2.2 Partage polynomial

Soient t et n deux entiers tels que $0 < t \leq n$. Un partage de secret à seuil t-parmi-n est une méthode de distribution d'un secret $s \in \{0,1\}^\ell$ en n morceaux $s_1, \ldots, s_n$ telle que pour tout sous-ensemble $S \subseteq \{1, \ldots, n\}$:

– si $\#S \leq t$, pour toute machine polynomiale A, en ℓ et n, et pour tout ε non-négligeable,

$$\Pr_{b \in \{0,1\},\, c \in \{0,1\}^\ell} [A((s_i)_{i \in S}, x_0, x_1) = b | x_b = s, x_{1-b} = c] < \frac{1}{2} + \varepsilon$$

30. Soit un ensemble de n joueurs. On appelle *structure d'accès* l'ensemble des sous-ensembles de joueurs (parties de $\{1, \ldots, n\}$) qui réunis peuvent recontruire le secret. Une propriété de ces structures est la monotonie : c'est-à-dire, soit $\mathcal{B}$ est une structure d'accès, si $B \in \mathcal{B}$ et $B \subseteq B'$ alors $B' \in \mathcal{B}$.

– si $\#S > t$, il existe une machine polynomiale A telle que

$$s = A((s_i)_{i \in S})$$

La formule d'interpolation de Lagrange

La formule d'interpolation de Lagrange permet de calculer un polynôme passant par un ensemble de points donnés. Le problème peut être formulé de la façon suivante.

Soit F un corps fini[31] et $F[X]$ l'anneau des polynômes à coefficients dans le corps F et à valeurs dans F. On représente par $F_{n-1}[X]$ l'ensemble des polynômes de $F[X]$ de degré $n-1$. Rechercher le polynôme $f \in F_{n-1}[X]$ prenant les valeurs données dans F: $b_1, \ldots, b_n$ aux points deux à deux distincts $a_1, \ldots, a_n$ de F.

Théorème 2. Il existe un et un seul $f \in F_{n-1}[X]$ tel que $f(a_i) = b_i$ pour $i = 1, \ldots, n$.

Preuve: Pour prouver l'unicité, on considère deux polynômes f_1 et f_2 de degré au plus $n-1$ qui décrivent la suite de points $\{(a_i, b_i) \ : i \in \{1, \ldots, n\}\}$. On dit qu'un polynôme p décrit une suite $\{(a_i, b_i) \ : i \in \{1, \ldots, n\}\}$ si $b_i = p(a_i)$ pour tout $i \in \{1, \ldots, n\}$. Dans ce cas, le polynôme $f_1 - f_2$ est de degré $n-1$ et s'annule en n valeurs. Ce polynôme est donc le polynôme nul et par conséquent $f_1 = f_2$.

Pour prouver l'existence, construisons un polynôme f de degré $n-1$ vérifiant $f(a_i) = b_i$ pour tout $i \in \{1, \ldots, n\}$. Considérons pour tout i, le polynôme $f_i(X) = \prod_{j \neq i} \frac{(X - a_j)}{(a_i - a_j)}$ de $F_{n-1}[X]$. Il prend la valeur 1 en a_i et 0 en a_j pour tout $j \neq i$. Alors, $f(X) = \sum_{i=1}^{n} b_i f_i(X)$ appartient à $F_{n-1}[X]$ et pour tout $i = 1, \ldots, n$, $f(a_i) = b_i$. □

La formule

$$f(X) = \sum_{i=1}^{n} b_i \times \prod_{j \neq i} \frac{(X - a_j)}{(a_i - a_j)}$$

s'appelle la *formule d'interpolation de Lagrange*. Ainsi, avec n valeurs, on peut reconstruire un polynôme de degré $n-1$ et déterminer la valeur de ce polynôme en n'importe quel point.

Interpolation de fraction rationnelle

Soit F un corps fini.

Définition 15 ***Fonction rationnelle.*** *Une fonction $r : F \to F$ est une fonction rationnelle si elle peut s'écrire sous la forme $r(x) = \frac{f(x)}{g(x)}$ où f et g sont deux polynômes quelconques de $F[X]$ et $g \neq 0$. Le degrée de r est donné par la paire (d_1, d_2), où d_1 est le degré de f et d_2 est celui de g.*

Une fonction rationnelle $r(x)$ décrit une suite $\{(x_i, y_i) | i = 1, \ldots, n\}$ si pour tout i, $y_i = r(x_i)$ ou x_i est un zero de f et g.

Théorème 3. (Unicité.) Si les fonctions rationnelles $\frac{f_1(x)}{g_1(x)}$ et $\frac{f_2(x)}{g_2(x)}$ de même degré (d_1, d_2) où $d_1 + d_2 < n$ décrivent une suite $\{(x_i, y_i) | i = 1, \ldots, n\}$, alors $\frac{f_1}{g_1} \equiv \frac{f_2}{g_2}$.

31. Nous verrons que la formule marche aussi dans certains anneaux.

Preuve: Les polynômes $f_1.g_2$ et $f_2.g_1$ sont égaux en x_i pour tout i. De plus, ces deux polynômes sont de degré $d_1 + d_2$ et comme ils sont égaux en $n > d_1 + d_2$ points, alors ils sont égaux. □

Théorème 4. (Interpolation.) Étant donnée une suite $\{(x_i, y_i)|i = 1, \ldots, n\}$, une fonction rationnelle de degré (d_1, d_2) qui décrit cette suite peut être trouvée en temps polynomial en n, si elle existe.

Preuve: Si les coefficients de f et g sont inconnus, alors les contraintes $f(x_i) = y_i g(x_i)$ sont des relations linéaires en les inconnues. Alors si le système linéaire a une solution, elle peut être trouvée par inversion de matrice. □

2.2.3 Partage de secret

Le schéma de partage de secret *à la Shamir*

Le partage de secret *à la Shamir* [170] permet de partager un secret s avec un polynôme de telle façon qu'une coalition de t parties parmi n ne puisse rien apprendre sur le secret, alors que toute réunion S de $t + 1$ peut retrouver le secret.

Phase d'initialisation Le distributeur $\mathcal{D}$ détermine un nombre premier p tel que $p > n + 1$ et code le secret comme un élément s de $\mathbb{Z}_p$. $\mathcal{D}$ choisit n éléments distincts non nuls de $\mathbb{Z}_p$ notés x_i pour $1 \leq i \leq n$. On peut supposer que $x_i = i$ par exemple. Les x_i sont rendus publics.

Phase de distribution $\mathcal{D}$ veut partager le secret $s \in \mathbb{Z}_p$ entre n serveurs. Il choisit secrètement au hasard t valeurs $f_1, f_2, \ldots, f_t \in \mathbb{Z}_p$ indépendantes. Pour chaque serveur i, $\mathcal{D}$ calcule $f(i)$ où le polynôme f est défini par :

$$f(x) = s + \sum_{j=1}^{t} f_j x^j \bmod p$$

puis transmet secrètement la valeur $f(i)$ au serveur i noté P_i.

Phase de reconstruction L'interpolation d'un polynôme de degré au plus t passant par $(t + 1)$ points distincts est obtenue de manière unique. La formule d'interpolation de Lagrange donne en un point $x \notin S$ où S est un sous-ensemble de $(t + 1)$ points parmi n :

$$f(x) = \sum_{j \in S} \lambda_{x,j}^S f(j) \bmod p, \text{ où } \lambda_{x,j}^S = \prod_{j' \in S \setminus \{j\}} \frac{(x - j')}{(j - j')}$$

Pour obtenir la valeur de f en 0, la formule devient

$$f(0) = s = \sum_{j \in S} \lambda_{0,j}^S f(j) \bmod p, \text{ où } \lambda_{0,j}^S = \prod_{j' \in S \setminus \{j\}} \frac{(0 - j')}{(j - j')}$$

Supposons que les serveurs $P_{i_1}, \ldots, P_{i_{t+1}}$ souhaitent reconstruire s. Ils savent que $y_{i_j} = f(x_{i_j})$ pour $1 \leq j \leq t + 1$ où $f \in F[X]$ est le polynôme secret choisi par $\mathcal{D}$. Le polynôme f est de degré au plus t, $f(X) = f_0 + f_1 X + \ldots + f_t X^t$ où $f_1, \ldots, f_t \in F$ sont inconnus et $f_0 = s$. Comme $y_{i_j} = f(x_{i_j})$ pour $1 \leq j \leq t + 1$, on obtient $t + 1$ équations linéaires aux $t + 1$ inconnues $f_0, \ldots, f_t$. Si les équations sont indépendantes, il y a une solution unique et on obtient la clé s. Il est important que le système des $t + 1$

équations linéaires admette une unique solution. On peut écrire le système d'équations linéaires sous la forme

$$\begin{cases} f_0 + f_1 x_{i_1} + \ldots + f_t x_{i_1}^t = y_{i_1} \\ f_0 + f_1 x_{i_2} + \ldots + f_t x_{i_2}^t = y_{i_2} \\ \ldots \\ f_0 + f_1 x_{i_{t+1}} + \ldots + f_t x_{i_{t+1}}^t = y_{i_{t+1}} \end{cases}$$

ou sous forme matricielle $A.F = Y$

$$\begin{pmatrix} 1 & x_{i_1} & x_{i_1}^2 & \ldots & x_{i_1}^t \\ 1 & x_{i_2} & x_{i_2}^2 & \ldots & x_{i_2}^t \\ \ldots & & \ddots & & \ldots \\ 1 & x_{i_{t+1}} & x_{i_{t+1}}^2 & \ldots & x_{i_{t+1}}^t \end{pmatrix} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_t \end{pmatrix} = \begin{pmatrix} y_{i_1} \\ y_{i_2} \\ \vdots \\ y_{i_{t+1}} \end{pmatrix}$$

La matrice A est une matrice de Vandermonde. Le déterminant d'une telle matrice est $\det(A) = \prod_{1 \leq j < k \leq t+1} (x_{i_k} - x_{i_j}) \bmod p$. Comme tous les x_{i_j} sont distincts dans $\mathbb{Z}_p$, $x_{i_k} - x_{i_j} \neq 0 \bmod p$ et comme $\mathbb{Z}_p$ est un corps, $\det(A) \neq 0$.

On peut maintenant essayer de voir quelle information obtient l'adversaire en ayant accès à t parts de la clé. Dans le système précédent, l'adversaire a t équations au lieu d'en avoir $t+1$. On rajoute l'équation $f_0 = r$ où r est un nombre aléatoire dans $\mathbb{Z}_p$. On obtient alors un système de $t+1$ équations qui a encore une fois une unique solution. Ainsi, toutes les valeurs de r sont possibles. Ainsi, un groupe de t utilisateurs quelconque n'obtient pas d'information sur s.

Le décodeur de Berlekamp Welch

Ce décodeur est issu de la théorie des codes correcteurs d'erreurs. Il utilise une fonction rationnelle décrivant une suite de points dont la plupart sont sur un polynôme à une seule variable. Les lemmes suivants proviennent de la thèse de Sudan.

Le problème que nous voulons résoudre est le suivant:
Étant donnés n points $\{(x_i, y_i) | i = 1, \ldots, n\}$, retrouver un polynôme p de degré au plus d tel que pour toutes sauf au plus e valeurs i (e erreurs), $y_i = p(x_i)$ (où $2e + d < n$).

Lemme 1 *Il existe une fonction rationnelle r de degré au plus $(e+d, e)$ qui décrit cette suite.*

Preuve: Supposons qu'un tel polynôme p existe. Considérons un polynôme g qui vaut 0 en x_i si $y_i \neq p(x_i)$. Il est clair grâce à la formule de Lagrange qu'un tel polynôme de degré au plus k existe. On considère maintenant la fonction rationnelle $\frac{p.g}{g}$. Elle décrit tous les points. □

Nous pouvons maintenant prouver le lemme suivant:

Lemme 2 *Étant donné n points $(x_i, y_i) \in F^2$, il existe un algorithme qui trouve un polynôme p de degré d tel que $p(x_i) = y_i$ pour toutes sauf e valeurs de i, où $2e + d < n$, si un tel polynôme p existe. Le temps d'exécution est polynomial en d et n.*

Preuve: Le lemme précédent dit qu'il existe une fraction rationnelle de la forme $\frac{p.g}{g}$ qui décrit les points. Une fonction rationnelle qui décrit cet ensemble peut être trouvée par interpolation et les fonctions rationnelles sont uniques à multiplication par une constante près et sont de la forme $\frac{p.g'}{g'}$. Ainsi, le quotient de la fonction rationnelle ainsi obtenue donne p. □

Algorithme de Berlekamp-Welch. Décrivons maintenant le fonctionnement de l'algorithme de Berlekamp-Welch. On change de notations pour les polynômes utilisés dans les paragraphes précédents. Cet algorithme permet de retrouver un polynôme p de degré $d < n - 2e - 1$ dans $\mathbb{Z}_q$ à partir d'un vecteur $\bar{y} \in {\mathbb{Z}_q}^n$: $\bar{y} = (y_1, \ldots, y_n)$ avec $y_i \in \mathbb{Z}_q$. Soit g un polynôme de degré $\leq e$ tel que $g(x_i) = 0$ pour tout $x_i = 1, \ldots, n$ quand $f(x_i) \neq y_i$. Par conséquent, pour tout $x_i = 1, \ldots, n$, $p(x_i)g(x_i) = g(x_i)y_i$. Le polynôme (pg) a un degré $\leq n - e - 1$. On a donc un système de n équations à n inconnues (les coefficients du polynôme (pg)) où le coefficient maximum de g vaur 1. Soient h et g, les solutions où g est un polynôme non nul de degré $\leq e$ et h un polynôme de degré $\leq n - e - 1$. On voit que quand $p(x_i) = y_i$ (*i.e.* en $n - e$ points), on a $h(x_i) = g(x_i)y_i = g(x_i)p(x_i)$, et donc $p = h/g$. L'algorithme consiste donc à résoudre un système de n équations à n inconnues.

Une implémentation de cet algorithme s'exécute en $\tilde{O}(n^2)$ où la notation $\tilde{O}(n)$ cache des termes polylogs en tirant parti de la structure particulière des solutions. L'algorithme le plus rapide est dû à Pan [141] qui a une complexité en temps en $O(n \log n \log \log n)$. Ces algorithmes sont utiles pour décoder rapidement les codes de Reed-Solomon et font partie de la théorie des codes correcteurs d'erreurs.

Cet algorithme permet donc de retrouver un polynôme de degré au plus d tel que $y_i = p(x_i)$ pour tout $i \in [1, n]$ sauf en au plus e points. Il est donc facile de montrer que cette solution est unique quand $e < (n - d)/2$. De manière surprenante, on peut aller au-delà de cette borne, mais la solution n'est alors plus unique. Dans un article récent, Guruswani et Sudan [103] ont montré que tant que $e < n - \sqrt{nd}$, il est possible de retrouver la liste de *tous* les polynômes p qui satisfont $y_i = p(x_i)$ pour toutes sauf e valeurs.

Ainsi, si nous avons $n \geq 3t + 1$ tel que au moins $t = n - 2e$, valeurs sont sur un polynôme de degré t, $(d = t)$, et au plus t erreurs, $(e = t)$, alors l'algorithme de Berlekamp-Welch permet de reconstruire ce polynôme. L'algorithme de Berlekamp-Welch permet donc de faire face à des adversaires actifs qui tentent d'envoyer des mauvaises parts pour empêcher la reconstruction du polynôme ou de la valeur en 0. Le nombre de tels attaquants doit être $< n/3$. Si l'on souhaite tolérer plus de machines corrompues, il va falloir utiliser une autre technique. On peut montrer que l'on peut accepter que $n/2$ machines soient corrompues en utilisant des canaux de broadcast comme le montrent les résultats de cryptographie interactive théorique.

2.2.4 Preuves interactives et zero-knowledge

Dans cette sous-section, nous rappellons la théorie développée pour prouver que les protocoles multiparties sont sûrs. L'idée principale est de réduire le problème général des protocoles à deux parties à un problème plus simple : Comment P(rouveur) peut convaincre V(érifieur) que x est dans le langage L de telle façon qu'aucune information au-delà du fait que $x \in L$ ne soit révélée ? Si cela peut être fait pour n'importe quel langage $L \in \mathcal{NP}$, P pourrait prouver à V qu'il a suivi les étapes du protocole dans un protocole distribué comme nous le verrons dans la suite de ce chapitre. Nous commençons par définir les termes de "preuve interactive" (ou "preuve par protocole") et de "zero-knowledge".

Système de preuve interactive

Une *machine de Turing interactive* (ITM) est une machine de Turing avec un ruban d'entrée en lecture seule, un ruban aléatoire en lecture seule, un ruban de travail en lecture/écriture, un ruban de communication en lecture seule, un ruban de communication en écriture seule, et un ruban de sortie en écriture seule. Quand on dit que l'on tire un bit aléatoirement, il s'agit de prendre le bit sur le curseur du ruban aléatoire et d'avancer le curseur d'une case vers la droite. Le contenu du ruban de communication

en écriture seule peut être vu comme les messages *envoyés* par la machine, alors que le contenu du ruban de communication en lecture seule peut être vu comme les messages *reçus* par la machine.

Un *protocole interactif* est une paire de ITMs (P, V) qui partagent le même ruban en entrée. De plus, le ruban de communication en écriture seule de V est le ruban en lecture seule de P et vice-versa. Les machines sont à tour de rôle actives, V commence. Pendant une étape active, la machine effectue quelques calculs internes basés sur le contenu de ses rubans et ensuite écrit une chaîne sur le ruban de communication en écriture seule. Le i-ème message de P (respectivement V) est la chaîne que P (respectivement V) écrit sur le ruban de communication à la i-ème étape. A cet instant, la machine est désactivée et l'autre machine devient active, sauf si le protocole est terminé. N'importe quelle machine peut décider de terminer le protocole, mais alors, elle n'envoie pas de messages. La machine P est une machine de Turing non bornée calculatoirement. Le temps de calcul de la machine V est défini comme la somme des temps de calculs pendant tous les états actifs et est borné par un polynôme en la taille de la chaîne d'entrée. Soit (P, V), une paire de machines de Turing interactives, on dénote par $[M_2(x_2), M_1(x_1)]$ la sortie de M_1 sur x_1 quand M_2 a x_2 comme entrée.

Définition 16 ***Système de preuve interactive.*** *Soit un langage $L \in \{0,1\}^*$. On dit que le langage L admet un* système de preuve interactive *s'il existe une paire de machines de Turing interactives (P, V) telles que :*

1. *sur l'entrée x, la machine V tourne en au plus $p(|x|)$ étapes, où $p(.)$ est un polynôme fixé,*
2. *Consistance : Pour toute constante $c > 0$, et tout mot $x \in L$ de taille suffisante,*

$$Pr\left[[P(x), V(x)] = 1\right] \geq 1 - |x|^{-c}$$

   (où les probabilités sont prises sur les rubans aléatoires de V et P),

3. *Significatif : Pour toute constante $c > 0$, toute machine $\tilde{P}$, et tout $x \notin L$ de taille suffisante, et tout $y \in \{0,1\}^*$,*

$$Pr\left[[\tilde{P}(y), V(x)] = 0\right] \geq 1 - |x|^{-c}$$

   (où les probabilités sont prises sur les rubans aléatoires de V et $\tilde{P}$).

On dit que (P, V) est un *système de preuve interactive* pour le langage L. On note $\mathcal{IP}$ l'ensemble des langages L qui possèdent un système de preuve interactive. On peut voir que la classe de langages $\mathcal{NP}$ est une classe spéciale de preuves interactives, où l'interaction est simple et le vérifieur V n'a pas de ruban aléatoire. En effet, comme P a une puissance de calcul infinie, il peut calculer un certificat (ou témoin) de taille polynomiale que V peut vérifier en temps polynomial. On a donc $\mathcal{NP} \subset \mathcal{IP}$.

Exemple : Le langage des non-résidus quadratiques

Soit $\mathbb{Z}_N^*$ l'ensemble des entiers positifs $x < N$ tel que x et N sont premiers entre eux $(\text{pgcd}(x, N) = 1)$. Soit QR, l'ensemble des éléments x de $\mathbb{Z}_N^*$ tels qu'il existe y tq $x = y^2 \bmod N$. Ces éléments, qui sont des carrés modulo N, sont appelés des *résidus quadratiques*, les autres des *non-résidus quadratiques*. Les résidus quadratiques forment un sous-groupe de $\mathbb{Z}_N^*$ alors que les non-résidus quadratiques ne forment pas de groupe. On les notera QNR.

Il existe un système de preuve interactive pour QNR. Soit l'entrée (x, N) sur le ruban d'entrée commun de (P, V). Dans le protocole suivant, P cherche à prouver à V que x est un élément de QNR.

– V choisit au hasard $z_i \in \mathbb{Z}_N^*$ et les $b_i \in \{0,1\}$. Puis il envoie à P une liste d'entiers $w_1, \ldots, w_k$, où $k = |N|$ et $w_i = {z_i}^2 \bmod N$ si $b_i = 0$ et $w_i = x.{z_i}^2 \bmod N$ si $b_i = 1$.

– P renvoie à V la liste $c_1, \ldots, c_k$ où $c_i = 0$ si w_i est un résidu quadratique modulo N et $c_i = 1$ sinon.

V accepte si et seulement si pour tout $i \in \{1, \ldots, k\}$, $c_i = b_i$. V interprète $b_i = c_i$ comme une preuve que $(x, N) \in QNR$, alors que $b_i \neq c_i$ l'amène à rejeter.

On dit que (P, V) est un système de preuve interactive pour QNR. Si $(x, N) \in QNR$, alors w_i est un résidu quadratique modulo N ssi $b_i = 0$. Ainsi, la machine P toute puissante peut facilement calculer si w_i est un résidu quadratique modulo N ou non, et calculer c_i correctement puis faire que V accepte avec probabilité 1. Si $(x, N) \notin QNR$, c'est-à-dire $(x, N) \in QR$, alors w_i est un résidu quadratique aléatoire modulo N indépendamment du fait que $b_i = 0$ ou $b_i = 1$. Alors, la probabilité que P (indépendamment du fait que P soit puissante ou non) puisse répondre c_i tel que $c_i = b_i$ vaut $\frac{1}{2}$ et pour chaque i la probabilité que V accepte est au plus $(\frac{1}{2})^k$.

Preuve d'appartenance à un langage et preuve de connaissance

Les preuves interactives ont été originellement développées pour les preuves d'appartenance à un langage par Goldwasser, Micali et Rackoff [100]. Dans une preuve, une partie P est engagée dans un protocole interactif avec une autre partie, V. Le but de l'interaction est de convaincre le vérifieur qu'une entrée x appartient à un langage. Le prouveur peut avoir une puissance calculatoire infinie. Les preuves de connaissance ont été introduites par Feige, Fiat et Shamir dans [67]. Supposons que $\mathcal{P}(x, w)$ soit un prédicat en temps polynomial[32]. Pour tout x, si w satisfait $\mathcal{P}(x, w) = 1$, alors w est appelé un *témoin* de x pour $\mathcal{P}(x, w)$. L'ensemble des témoins de x est noté $w(x)$. La preuve consiste pour un prouveur P polynomial de prouver à V qu'il connaît un témoin w pour l'entrée x. Par exemple, dans le cas du langage QR, un témoin w d'un élément x du langage est tel que $x = w^2 \bmod N$. De même, dans le cas du logarithme discret, soit le langage G_q composé des éléments de $\mathbb{Z}_p^*$ d'ordre q, où q est un nombre premier divisant $p - 1$. Un témoin w pour un élément x vérifie la relation $x = g^w \bmod p$ où g est un générateur du sous-groupe d'ordre q.

Les composants de base pour de tels protocoles sont les *challenges* envoyés au prouveur par le vérifieur et les *réponses* calculées par le prouveur et retournées au vérifieur. Afin de convaincre le vérifieur, le prouveur doit être capable de donner des réponses valides à la plupart des challenges. Dans le cas des preuves de connaissance, les capacités du prouveur à répondre au challenge dépend de la possession du témoin.

De manière intuitive, une preuve interactive de connaissance pour un prédicat $\mathcal{P}(x, w)$ doit satisfaire les propriétés suivantes, si la probabilité est une fonction de $|x|$:

– Si P connaît un témoin w pour x, alors il doit être capable de convaincre V avec probabilité écrasante (consistance).

– Si P ne connaît pas de témoin w tel que $\mathcal{P}(x, w) = 1$, alors V peut seulement être convaincu avec probabilité négligeable (significatif).

Comme P est une machine de Turing, que signifie connaître w ? Une hypothèse possible est que P a w sur un de ses rubans. Mais ceci est trop restrictif et P pourrait connaître w d'une manière plus complexe comme par un calcul. Une définition informelle possible de ce concept est donnée dans [100] : *P connaît w s'il existe une machine de Turing M avec un contrôle complet sur P qui peut afficher w comme résultat de ses interactions avec P.* Dire que la machine M a un contrôle complet de P signifie que M a le pouvoir de remettre à zéro et de réexécuter P polynomialement autant de fois qu'elle le désire.

32. Un prédicat en temps polynomial $\mathcal{P}(x, w)$ est un prédicat où $|w|$ est polynomialement relié à $|x|$ et la valeur "vraie" du prédicat peut être vérifiée en temps polynomial.

Si le protocole est de la forme challenges/réponses, le vérifieur peut envoyer un nombre polynomial de challenges.

Ainsi, dans une preuve de connaissance, la propriété de *significativité* donnée à la définition 16 peut être formulée de la manière suivante : il existe une machine de Turing M probabiliste en temps polynomial avec un contrôle complet sur P telle que pour tout P, si un vérifieur V accepte avec probabilité non-négligeable, à la fin de l'exécution de M, M peut obtenir w' tel que $P(x, w') = 1$ avec probabilité écrasante. La machine M est appelée "extracteur du secret".

$$\forall a, \exists M, \forall b, \forall P, \exists c, \forall |x| > c, \Pr\left[V_P(x) = \text{accepte}\right] > \frac{1}{|x|^a}$$

$$\Rightarrow \Pr\left[\text{sortie de } M_P \text{ sur } x \text{ satisfait } \mathcal{P}\right] > 1 - \frac{1}{|x|^b}$$

Zero-knowledge

Maintenant que nous savons ce qu'est un système de preuve de connaissance, on souhaite savoir quelle quantité d'"information" a été transférée durant le protocole pour convaincre un vérifieur polynomial de la véracité de la proposition. Un protocole dans lequel le prouveur d'un prédicat $\mathcal{P}(x, w)$ enverrait w au vérifieur serait encore un protocole de connaissance, mais dans ce cas, le vérifieur pourrait ensuite usurper l'identité du prouveur.

Goldwasser, Micali et Rackoff [99] ont précisé cette notion. Ils disent qu'un système de preuve interactive pour un langage L est zero-knowledge si pour tout $x \in L$ ($\exists w\ \mathcal{P}(x, w) = 1$ si $x \in L$), ce que le vérifieur peut calculer après avoir participé au protocole avec le prouveur, aurait pu être calculé par une machine probabiliste en temps polynomial avec seulement un accès à x. Il s'agit de la robustesse du prouveur contre des tentatives du vérifieur pour extraire de l'information via les interactions. On peut remarquer que le vérifieur peut dévier de manière arbitraire, mais en temps polynomial, du programme pré-déterminé.

Définitions

Soit (P, V) un protocole interactif. On définit la variable aléatoire $vue^{(V)}_{P(x,w)}(x, h)$ comme étant la vue de V pendant l'exécution du protocole avec P, sur l'entrée commune x, et l'entrée auxiliaire h (historique qui contient les bits aléatoires de V et les messages précédents échangés avec P). La chaîne h représente l'entrée privée que possède le vérifieur avec comme restriction que sa longueur soit polynomialement bornée. La variable aléatoire vue dépend des rubans aléatoires de P et V.

Le protocole (P, V) est dit *parfaitement zero-knowledge* pour L si, pour toute machine de Turing $\tilde{V}$ probabiliste en temps polynomial, il existe une machine de Turing probabiliste et polynomiale, $M_{\tilde{V}}$, telle que pour tout $x \in L$ et pour tout polynôme p, et pour toute chaîne h telle que $|h| < p(|x|)$, qui produise une sortie $M_{\tilde{V}}(x, h)$ distribuée de la même façon que la variable aléatoire vue. (La sortie de la machine M, $M_{\tilde{V}}(x, h)$, est distribuée suivant les bits aléatoires de $M_{\tilde{V}}$ avec en entrée x et h.)

On dit que le protocole (P, V) est un *protocole statistiquement zero-knowledge* pour L si pour toute machine $\tilde{V}$ probabiliste en temps polynomial, il existe une machine de Turing probabiliste et polynomiale $M_{\tilde{V}}$ telle que pour tout $x \in L$ et pour tout polynôme p, et pour toute chaîne h telle que $|h| < p(|x|)$,

$$\sum_{\alpha} \left| \Pr\left[M_{\tilde{V}}(x, h) = \alpha\right] - \Pr\left[vue^{(V)}_{P(x,w)}(x, h) = \alpha\right] \right| < \frac{1}{Q(|x|)}$$

pour tout polynôme $Q(.)$ et pour tout x suffisamment grand.

Intuitivement, on peut voir le vérifieur dans un protocole statistiquement zero-knowledge comme un juge possédant une puissance infinie et observant des échantillons $\{M_{\tilde{V}}(x,h)|$ les bits aléatoires de $M_{\tilde{V}}\}$ et $\{vue^{(V)}_{P(x,w)}(x,h)|$ les bits aléatoires de P et $V\}$ de taille polynomiale et pouvant dire de quelle distribution proviennent les échantillons.

Enfin, on dit qu'un protocole interactif (P,V) est un *protocole calculatoirement zero-knowledge* pour L si un juge borné polynomialement en temps et recevant un nombre polynomial d'échantillons pouvait séparer les deux échantillons selon leur provenance.

Deux distributions sont dites *calculatoirement indistinguables* si pour toute machine de Turing probabiliste et en temps polynomial D, où D peut retourner deux valeurs (0 signifie que la distribution provient de $M_{\tilde{V}}$ et 1 que la distribution provient de $vue^{(V)}_{P(x,w)}(x,h)$), pour tous polynômes $P(.)$ et $Q(.)$, pour tout $x \in L$ suffisamment grand, et toute chaîne $h < P(|x|)$,

$$\left|\Pr_{\alpha}\left[D(\alpha)=1|\alpha \overset{R}{\leftarrow} M_{\tilde{V}}(x,h)\right] - \Pr_{\alpha}\left[D(\alpha)=1|\alpha \overset{R}{\leftarrow} vue^{(V)}_{P(x,w)}(x,h)\right]\right| < \frac{1}{Q(|x|)}$$

Si on note $E_{\delta}(X)$ l'espérance d'une variable aléatoire X à valeurs dans $\{0,1\}$ sur un ensemble δ, alors $E_{\delta}(X) = \Pr_{\alpha\in\delta}[X(\alpha)=1]$, et on en déduit que

$$|E_{\delta_0}(D) - E_{\delta_1}(D)| < \mathsf{negl}(|\mathsf{x}|)$$

Deux distributions sont donc calculatoirement indistinguables, si la distance maximale entre les espérances de tout distingueur, D, sur chaque distribution est négligeable.

On dit que L a un *système de preuves (parfaitement/statistiquement/calculatoirement) zero-knowledge* si

1. il existe un système de preuve interactif (P,V) pour L,
2. pour toute ITM $\tilde{V}$, le protocole interactif $(P,\tilde{V})$ est (parfaitement/statistiquement/ calculatoirement) zero-knowledge pour L.

On note $\mathcal{ZKIP}$ l'ensemble des langages qui ont une preuve calculatoirement zero-knowledge. On peut alors montrer [96] que si les fonctions à sens unique existent alors $\mathcal{NP} \subset \mathcal{ZKIP}$.

La propriété "zero-knowledge" est bien souvent trop forte et exige au moins 4 passes entre le vérifieur et le prouveur [107]. Il existe cependant des schémas d'identification ou des preuves de connaissance sûrs en 3 passes contre des attaques actives du vérifieur. Dans ce cas, on prouve que l'on peut transformer une attaque active du vérifieur en une machine capable de résoudre un problème difficile, comme la factorisation dans le cas du protocole d'identification de Fiat-Shamir [133, 146]. Pour ce faire, on utilise des protocoles dits à témoins cachés ou indistinguables ("witness-hiding" ou "witness-indistinguishability"). Au lieu de prouver que durant le protocole aucune information n'est révélée, dans le cas des témoins cachés, on prouve qu'aucune information *sur le secret* n'est révélée. L'avantage est que la propriété de "witness indistinguishability" est conservée par composition parallèle du protocole ce qui permet de concevoir des protocoles plus efficaces. Les preuves zero-knowledge sont séquentiellement sûres. C'est-à-dire que l'on peut répéter plusieurs protocoles à quatre passes plusieurs fois pour atteindre un niveau de sécurité élevé. Ceci provient du ruban d'histoire permettant de transmettre de l'information entre les différents tours. Dans le cas des preuves witness-indistinguishable, on peut les répéter en parallèle, c'est-à-dire en faire plusieurs en même temps, ce qui permet de faire des preuves plus efficaces d'un même niveau de sécurité.

En cryptographie partagée, on utilise les preuves d'appartenance à un langage afin de prouver aux autres joueurs qu'un joueur a effectué correctement son travail. Dans ce cas, comme dans le cas des protocoles d'identification sûrs contre des attaques par "reset" ("resettable Zero-Knowledge", rZK) [6], un

prouveur peut convaincre de son identité en étant "capable" de prouver l'appartenance à un langage *difficile* L, plutôt que de prouver qu'il connaît un témoin (la clé secrète) pour un langage L (preuve de connaissance)[33].

Preuves non-interactives

On peut transformer n'importe quel schéma zero-knowledge sûr contre un vérifieur honnête[34] en un schéma de signature sûr dans le modèle de l'oracle aléatoire en utilisant l'heuristique de Fiat-Shamir. Elle consiste à prendre pour challenge le résultat d'une fonction de hachage sur le premier message (dans une preuve ZK classique en trois tours comme le schéma Fiat-Shamir [69]) émis par le prouveur et sur certaines constantes du système. La théorie des preuves non-interactives de connaissance utilise généralement un ruban d'aléas partagé entre le prouveur et le vérifieur [24]. Pour émuler ce ruban, on utilise une fonction de hachage dont on suppose que la sortie ne peut pas être prédite à l'avance par le prouveur. Les preuves d'appartenance à un langage difficile peuvent aussi être rendues non-interactives grâce à cette heuristique.

Exemples de preuve d'identification sûre contre un adversaire passif

Le protocole d'identification de Schnorr [166] est un protocole de preuve de connaissance d'un logarithme discret. Soit G_q un sous-groupe de $\mathbb{Z}_p^*$ d'ordre premier q tel que $q|p-1$ et g un générateur de G_q. Le prouveur a une clé publique y pour laquelle il connaît un témoin x tel que $y = g^x \bmod p$.

Prouveur		Vérifieur
$x \in \mathbb{Z}_q$	$y = g^x \bmod p$	
$k \in_R \mathbb{Z}_q$		
$t = g^k \bmod p$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \in_R [0, B[$
$z = k + ex \bmod q$	$\xrightarrow{\quad z \quad}$	vérifie $g^z y^{-e} \stackrel{?}{=} t \bmod p$

Preuve:

Consistante. Il est clair que cette preuve est consistante car un prouveur honnête, connaissant le secret x, arrive toujours à calculer t et z tels que $t = g^z y^{-e} \bmod p$. Le prouveur est capable de répondre aux B questions du vérifieur.

Significative. Supposons un prouveur qui réussit l'authentification avec probabilité supérieure à $\varepsilon + 1/B$, c'est-à-dire qu'il réussit à calculer z tel que $t = g^z y^{-e} \bmod p$ pour au moins une fraction

33. "Reseter" une machine signifie "remettre à zéro" la mémoire de la carte. Les preuves zero-knowledge resettable sont utiles dans le cas d'une carte à puce utilisée pour faire de l'authentification. En effet, si l'on n'utilise pas ce genre de preuve, mais un processus d'identification bâti sur une preuve de connaissance, alors l'attaque suivante permet de retrouver le secret de la carte. Supposons qu'un adversaire obtienne la carte à puce d'un utilisateur. Il fait tourner la carte et enregistre les réponses de la carte en face d'un vérifieur quelconque. Puis, l'adversaire remet à zéro la carte. Ceci va réinitialiser la mémoire de la carte dans le même état que celui d'origine. Le générateur aléatoire de la carte permettant de générer le premier message de la preuve d'identification sera remis lui aussi dans son état d'origine. Par conséquent, il générera les mêmes randoms et les mêmes premiers messages. Le vérifieur quant à lui fournira des challenges différents. Ceci permettra alors à l'attaquant d'obtenir le secret de la carte car il obtiendra dans le troisième message deux équations pour le même premier message. Pour être protégé contre ce type d'attaque, il faut donc utiliser des protocoles d'identification bâtis sur des protocoles zero-knowledge resettable.

34. c'est-à-dire que V ne choisit pas les challenges suivant une stratégie, mais de manière honnête (aléatoire), en accord avec le protocole.

$\varepsilon + 1/B$ des B questions possibles e. Pour prouver que cette *preuve de connaissance* est significative, nous fabriquons un extracteur du secret x. Pour ce faire, nous remplaçons le vérifieur. Quand le prouveur cherche à s'authentifier, le vérifieur voit les messages (t, e, z). Si nous *rembobinons* la machine du prouveur au moment où il a envoyé t et que nous envoyons $e' \neq e$, alors le prouveur va renvoyer z' tel que $t = g^z y^{-e} = g^{z'} y^{-e'}$. Si nous posons $x = (z - z')/(e - e') \bmod q$, alors x est le logarithme discret de y en base g. La complexité de l'extracteur est en $O(1/\varepsilon)$. Ainsi, pour avoir un extracteur fonctionnant en temps polynomial, il faut avoir ε non-négligeable.

Si le prouveur ne connaît pas la clé secrète x, il ne pourra pas répondre à plus d'une question. Sa probabilité de fraude est donc inférieure à $1/B$. Ainsi, quiconque a une probabilité de passer avec succès le protocole d'identification doit être capable d'obtenir le secret du prouveur[35].

Zero-knowledge. Nous allons décrire un simulateur pour lequel les messages échangés avec le vérifieur seront indistinguables pour ce dernier à ceux d'une exécution réelle (avec le vrai prouveur). En effet, on peut remplacer le prouveur par un simulateur qui ne connaît pas la clé secrète x. Le simulateur commence par choisir aléatoirement $z \in \mathbb{Z}_q$ et devine la question e que va lui poser le vérifieur. Il peut donc calculer $t = g^z y^{-e} \bmod p$.

Il commence donc par envoyer t ainsi calculé. Si le vérifieur envoie la question e, le simulateur renvoie z, sinon, le simulateur arrête et recommence avec un nouveau z'. Si $B = 2$, alors avec probabilité 1/2, le simulateur réussira l'authentification. Ainsi, le temps d'exécution de cet algorithme sera polynomial. Si B n'est pas de taille petite, ce protocole n'est pas zero-knowledge car la complexité du simulateur est en $O(B)$.

□

Ce protocole est donc sûr face aux *attaques passives* du vérifieur car le protocole est zero-knowledge contre un vérifieur honnête. En général, on doit envisager le cas où le prouveur s'identifie plusieurs fois au vérifieur. Ce dernier peut ne pas poser ses questions de manière aléatoire afin d'obtenir des informations sur le secret x. Dans une attaque passive, le vérifieur pose ses questions aléatoirement pour augmenter la probabilité d'attraper un prouveur malhonnête. En effet, pour prouver que le protocole est zero-knowledge, nous avons tiré les questions du vérifieur au hasard. Cependant, si ce dernier ne tire pas ses questions au hasard mais en fonction d'une stratégie qui lui permet d'extraire de l'information sur x, nous ne pouvons plus faire la simulation.

Exemple de preuve d'identification sûre contre un adversaire actif

Contre une attaque active de la part du vérifieur, le schéma de Schnorr n'a été prouvé sûr que pour B de taille petite. Pour contrer ce type d'attaque quand B est grand, on peut utiliser un protocole "witness-indistinguable" comme l'a fait Okamoto à Crypto '92 [133]. Ce protocole est le suivant. Dans ce schéma, il y a deux bases g et $h \in \langle g \rangle$ telles que le logarithme de h en base g est tenu secret. La preuve de witness-indistinguishability prouve que si le vérifieur mène une attaque active, alors il peut retrouver le logarithme de h en base g, ce qui constitue un problème difficile. Le prouveur a une clé publique y pour laquelle il connaît un témoin r et s dans $\mathbb{Z}_q$ tels que $y = g^r h^s \bmod p$. Il s'agit d'une preuve de représentation.

35. Cependant, cela ne suffit pas à prouver que le protocole est *sûr*. En effet, si le prouveur révélait simplement x pour prouver son identité, le protocole serait toujours consistant et significatif, mais le vérifieur pourrait réutiliser x pour se faire passer pour le prouveur. Ceci amène naturellement à la notion d'information sur x transmise par le protocole. On espère qu'aucune information sur x ne soit révélée au cours du protocole qui permettrait au vérifieur de se faire passer pour le prouveur dans la suite.

Prouveur		Vérifieur
$r, s \in \mathbb{Z}_q$	$y = g^r h^s$	
$k, u \in_R \mathbb{Z}_q$		
$t = g^k h^u \bmod p$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \in_R [0, B[$
$z = k + er \bmod q$		
$w = u + es \bmod q$	$\xrightarrow{\quad z, w \quad}$	vérifie $g^z h^w y^{-e} \stackrel{?}{=} t \bmod p$

Supposons que le prouveur s'identifie au vérifieur un nombre polynomial de fois en suivant le protocole. En supposant que le vérifieur est capable d'obtenir de l'information sur les exposants r et s, on montre qu'une association entre le prouveur et le vérifieur peut calculer le logarithme de h en base g, ce qui contredit l'hypothèse de sécurité. On peut prouver les deux théorèmes suivants :

Théorème 5. On suppose que le vérifieur connaît une valeur t telle qu'il puisse satisfaire le procédé d'identification avec succès avec probabilité supérieure à $\varepsilon + 1/B$. Alors, il peut calculer r' et s' tels que $y = g^{r'} h^{s'} \bmod p$ en temps polynomial.

Théorème 6. On suppose que le vérifieur connaît une valeur t telle qu'il puisse satisfaire le procédé d'identification avec succès avec probabilité supérieure à $\varepsilon + 1/B$. Alors, le prouveur et le vérifieur peuvent calculer $\log_g h$ en temps polynomial avec probabilité $1 - 1/q$.

En effet, la connaissance de deux témoins (r, s) et (r', s') associés à une même clé publique y, $y = g^r h^s = g^{r'} h^{s'} \bmod p$, donne le logarithme de h en base g, $\log_g h = (s' - s)/(r' - r) \bmod q$. Pour une preuve de ces deux théorèmes, le lecteur lira [181].

Le lecteur lira les thèses de David Pointcheval et Guillaume Poupard pour plus de renseignements sur ces preuves [146, 154].

Exemple de preuve non-interactive et signature

La preuve de Schnorr peut être transformée en une preuve non-interactive de connaissance du logarithme discret et en un schéma de signature. Ce schéma a été prouvé sûr [166] *dans le modèle de l'oracle aléatoire* [9].

- la clé publique : un quadruplet (y, p, g, q), où $y = g^x \bmod p$, p et q sont des nombres premiers tels que $q|p - 1$, et g est un générateur de G_q, sous-groupe d'ordre q de $\mathbb{Z}_p^*$.
- la clé secrète : x tel que $y = g^x \bmod p$.
- Signer : Le signataire génère un random k dans $\mathbb{Z}_q$. La signature d'un message m est une paire (e, z) telle que $e = H(g, p, g^k, m)$, $z = k + ex \bmod q$.
- Vérifier : Vérifier si $e = H(g, p, g^z / y^e, m)$.

Il est clair que cette preuve est consistante.

Ce protocole est aussi significatif car un signataire qui ne connaît pas x n'arrivera pas à générer une signature valide avec probabilité supérieure à $\varepsilon + 1/B$, où B est la taille de la fonction de hachage. De la même manière que précédemment, on montrerait que l'on arrive à extraire le secret x au signataire.

Zero-knowledge Regardons maintenant la simulation zero-knowledge. Nous pouvons construire un simulateur qui simule la vue de l'adversaire sans connaître la valeur x. Cette vue comprend les valeurs de l'oracle aléatoire aux points où l'adversaire a questionné l'oracle de telle sorte que le simulateur soit complètement responsable de l'oracle aléatoire. Quand l'adversaire fait une requête à l'oracle, si l'oracle n'a pas encore défini la valeur en ce point, le simulateur la définit avec une valeur aléatoire et dans tous les cas retourne la valeur à l'adversaire. Pour simuler la vue de l'adversaire, le simulateur choisit $e \in [0, B[$ (où B est la taille de la fonction de hachage $H(.)$) et $z \in [0, q[$ au hasard, et pour des valeurs données m et y, il définit la valeur de l'oracle aléatoire au point $(g, p, g^z/y^e, m)$ comme étant e. Avec probabilité écrasante, le simulateur n'a pas encore défini l'oracle en ce point auparavant, et il est donc libre de le faire. La preuve de validité est constituée de la paire (e, y). Il est simple de vérifier que la distribution produite par ce simulateur est parfaite.

Remarque 9 *Le schéma de signature DSS ou DSA [130] du NIST, norme américaine, est fortement inspiré de ce schéma. La sécurité du schéma DSA n'a cependant jamais pu être prouvée. En revanche, des variantes de ce schéma, comme la norme coréenne de signature, peuvent l'être.*

2.2.5 Partage de secret publiquement vérifiable

Quand un distributeur partage un secret s et donne les parts s_i aux parties P_i, aucun ensemble de t parties n'est supposé connaître le secret et personne ne peut vérifier si la part obtenue est valide, c'est-à-dire si le distributeur a correctement effectué son travail. Chaque partie veut être sûre que la combinaison des parts redonne bien le secret, ou tout du moins un secret non ambigu. Une solution consiste à faire une preuve d'appartenance à un langage. Une autre solution est d'utiliser des fonctions mises en gage. C'est le cas du partage de Feldman [68]. Le premier schéma de partage publiquement vérifiable a été conçu par Chor, Goldwasser, Micali et Awerbuch dans [47].

Un scéma de partage de secret publiquement vérifiable est composé des trois algorithmes suivants :

1. *Partager* : Le distributeur utilise les fonctions de chiffrement publiques pour distribuer les parts en calculant $S_i = \mathcal{E}_{\mathsf{pk}_i}(s_i)$ pour $1 \leq i \leq n$. Le distributeur publie ensuite chaque part S_i.

2. *Recouvrer* : Si un groupe de joueurs veut reconstruire le secret, ils exécutent *Recouvrer* qui satisfait la propriété suivante : pour tout sous-ensemble B appartenant à une structure d'accès $\mathcal{B}$, *Recouvrer*$(\{S_i | i \in B\}) = s$, le secret partagé initialement par le distributeur, et pour tout $B \notin \mathcal{B}$, il est calculatoirement impossible de calculer s à partir des $\{S_i | i \in B\}$.

3. *Pub_Verification* : Pour vérifier la validité des parts chiffrées, *Pub_Verification* peut être exécuté par n'importe quelle partie. Cet algorithme a la propriété suivante : il existe u, tel que pour tout sous-ensemble B,

$$(Pub\_Verification(\{S_i | i \in B\}) = 1) \Rightarrow Recouvrer(\{\mathcal{D}_i(S_i) | i \in B\}) = u$$

   et $u = s$ si le distributeur est honnête.

Décrivons le schéma de Feldman que nous réutiliserons dans la suite. Feldman a conçu un système de partage publiquement vérifiable de clé Diffie-Hellman. Soit une clé Diffie-Hellman $y = g^x \bmod p$ dans le sous-groupe de $\mathbb{Z}_p^*$ engendré par g d'ordre q, $(q|p-1)$.

Le distributeur effectue un partage *à la Shamir* de la clé secrète $\mathsf{sk} = x$ en choisissant un polynôme aléatoire f tel que $f(0) = x$ et t coefficients a_k au hasard dans $\mathbb{Z}_q$ et pose

$$f(X) = \sum_{k=0}^{t} a_k X^k \in \mathbb{Z}_q$$

La part du joueur i est alors définie comme la valeur du polynôme f en i, soit $f(i) \bmod q$. Puis, il calcule pour chaque part, $y_i = g^{f(i)} \bmod p$, et pour chaque coefficient, $A_k = g^{a_k} \bmod p$, et broadcaste ces valeurs sur un canal de broadcast public. Il transmet aussi $f(i)$ au i-ème serveur en utilisant un canal privé et ce dernier vérifie sa part en calculant

$$y_i \stackrel{?}{=} g^{f(i)} \bmod p$$

Les autres serveurs vérifient la part du joueur i en calculant

$$y_i \stackrel{?}{=} \prod_{k=0}^{t} A_k^{i^k} = g^{\sum_{k=0}^{t} a_k i^k} = g^{f(i)} \bmod p$$

Si la part du i-ème serveur n'est pas correcte, ce dernier broadcaste $f(i)$ et le distributeur est déclaré fautif.

Au moment de la reconstruction, les serveurs envoient $f(i)$ et chaque serveur peut vérifier si $y_i \stackrel{?}{=} g^{f(i)} \bmod p$. Si oui, $t+1$ parts correctes permettent de reconstruire la valeur de f en 0, c'est-à-dire x.

La sécurité de ce schéma est basée sur la difficulté de calculer le logarithme discret. Stadler dans [178] a construit un système PVSS basé sur le logarithme discret à deux étages (g^{h^x}). Schoenmakers dans [169] a simplifiée l'hypothèse en utilisant le problème DDH et récemment Young et Yung dans [186] ont utilisé le problème CDH dans le modèle de l'oracle aléatoire.

2.3 Partage de fonction

Dans cette section, nous allons voir les propriétés des schémas de partage de fonction, la notion de sécurité d'un système de chiffrement, et enfin nous prouverons la sécurité d'une version distribuée du cryptosystème El Gamal.

2.3.1 Propriétés des schémas cryptographiques de partage de fonction

Les protocoles de partage de fonction (signature et déchiffrement) doivent vérifier les deux propriétés suivantes : la *sécurité du schéma cryptographique* et la *robustesse*.

Sécurité du schéma cryptographique

La sécurité du schéma signifie que le protocole partagé doit être aussi sûr que le protocole non partagé vu que l'on souhaite résister à des adversaires plus forts que dans le cas "centralisé".

Dans le cas d'un schéma de signature, on souhaite éviter les *forges existentielles* face à une attaque adaptative à messages choisis.

Dans le cas d'un système de déchiffrement, on souhaite prouver la *sécurité sémantique* contre une attaque à clairs choisis, inévitable en cryptographie à clé publique et contre une attaque à chiffrés choisis dans le cas le plus fort.

Dans le cas d'un partage de génération d'une clé, on souhaite prouver que l'attaquant ne peut pas obtenir d'information sur la part de clé connue par les serveurs non corrompus.

Robustesse

Certains schémas de partage à seuil sont vulnérables aux attaques d'un adversaire actif essayant de corrompre les parties pour empêcher la génération d'une signature valide, le déchiffrement d'un message ou la génération d'une clé. Dans le cas où tous les joueurs jouent honnêtement (pas d'adversaire actif), on dit que l'on est dans le modèle *honnêtes-mais-curieux* car des joueurs "corrompus" qui jouent correctement peuvent se coaliser pour tenter d'obtenir de l'information sur la part des autres.

Considérons un protocole de signature ou de déchiffrement dans lequel un joueur particulier appelé le *combineur* va envoyer à chaque joueur un message x à signer ou à déchiffrer. Chaque serveur possède une partie de la clé secrète sk_i et va calculer $y_i = f(x, \mathsf{sk}_i)$. Un attaquant actif, au lieu de renvoyer y_i comme tous les serveurs, va renvoyer $\tilde{y_i}$ pour empêcher le déchiffrement ou la génération d'une signature. Le combineur va recevoir la liste des parts de signature ou de déchiffrement $\{y_1, \ldots, y_n\}$ et va tenter de reconstuire la signature ou le déchiffrement.

Dans le cas d'une signature, le combineur peut prendre $t+1$ parts au hasard et tenter de reconstuire la signature. Une signature putative est ensuite testée pour vérifier sa validité. Pour ce faire, le combineur peut essayer de vérifier la signature. La probabilité de tomber sur un groupe de k serveurs honnêtes nécessaires parmi les n serveurs alors qu'il y a au plus t serveurs corrompus est le nombre de sous-ensembles contenant k bons serveurs parmi les $n-t$ serveurs honnêtes, sur le nombre de groupe de k serveurs possibles parmi n : soit $P = \frac{C_{n-t}^k}{C_n^k}$. Pour tomber sur un bon groupe, le combineur doit essayer $1/P$ groupes en moyenne. Ce nombre est exponentiel en le nombre de serveurs attaqués. Dans le cas où le nombre de serveurs corrompus est petit, $t < n/3$, l'algorithme de Berlekamp-Welch permet de reconstruire le polynôme. Mais que faire si $\frac{n}{3} \leq t < \frac{n}{2}$?

Une solution consiste pour chaque joueur à prouver que son calcul est correct. Les techniques de preuves non-interactives d'appartenance à un langage difficile ou de preuves de connaissance zero-knowledge vont permettre de résoudre efficacement ce problème.

2.3.2 Sécurité d'un système de chiffrement partagé

Pour prouver la sécurité des schémas partagés, on peut soit effectuer une réduction entre le protocole dans sa version "centralisée" vers le protocole dans sa version "distribuée", soit montrer une réduction entre un problème difficile et le protocole "distribué". Dans le premier cas, on prouve que s'il existe un adversaire contre le schéma distribué, alors on peut construire un attaquant contre le schéma centralisé. Or si ce schéma est sûr, de tels attaquants n'existent pas et il n'existe donc pas non plus d'adversaires contre le schéma distribué.

Modèle formel d'un système de chiffrement partagé

Dans cette sous-section, nous définissons un modèle formel pour un cryptosystème à seuil t parmi n. On suppose l'existence d'un distributeur de confiance et d'un ensemble de serveurs de déchiffrement $P_1, \ldots, P_n$.

Dans une *phase d'initialisation*, le distributeur crée une clé publique pk, une clé de vérification vk, et des clés privées $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$. Pour $1 \leq i \leq n$, la clé privée sk_i est donnée au serveur P_i.

Un utilisateur qui veut envoyer un message avec un label[36] donné peut exécuter l'algorithme de chiffrement en utilisant la clé publique.

36. Le service de déchiffrement ne doit pas déchiffrer tout ce qui lui arrive et le donner à n'importe qui, mais doit implémenter une politique de déchiffrement. Pour implémenter de telles politiques, on peut incorporer un *label* au chiffré durant la phase de chiffrement. Un tel label est une chaîne de bits contenant des informations qui peuvent être utilisées par un tiers pour déterminer si la demande de déchiffrement est autorisée selon la politique. Un label pourrait aussi contenir l'identité du receveur de sorte que les serveurs de déchiffrement lui envoie les parts de déchiffrement.

Un utilisateur qui veut déchiffrer un chiffré, le donne aux serveurs en demandant une part de déchiffrement. Le label est placé dans le chiffré de telle sorte que les serveurs peuvent le vérifier. L'utilisateur peut vérifier la validité des parts en utilisant la clé de vérification. Quand un utilisateur collecte des parts valides d'au moins $t+1$ serveurs, il peut exécuter l'algorithme de combinaison pour obtenir le déchiffré.

De manière plus formelle, un système de chiffrement à seuil comprend les algorithmes suivants.

- Un *algorithme probabiliste de génération de clés* $\mathcal{K}$ qui prend en entrée un paramètre de sécurité k, un nombre $n \geq 1$ de serveurs de déchiffrement, et le paramètre du seuil t $(1 \leq t < n)$; et retourne
$$(\mathsf{pk}, \mathsf{vk}, \mathsf{sk}) = \mathcal{K}(k, n, t)$$
où pk est *la clé publique de chiffrement*, vk est *la clé de vérification*, et $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ est la liste de *clés secrètes*.

- Un *algorithme probabiliste de chiffrement* $\mathcal{E}$ qui prend en entrée la clé publique pk, le clair m et un label L, et retourne le chiffré $c = \mathcal{E}_{\mathsf{pk}}(m, L)$.

- Un *algorithme d'extraction du label* $\mathcal{L}$ qui prend en entrée un chiffré c et retourne un label $L = \mathcal{L}(c)$.

- Un *algorithme probabiliste de génération d'une part de déchiffré* $\mathcal{D}$ qui prend en entrée une clé privée sk_i et un chiffré c et retourne la *part de déchiffrement* $\sigma_i = \mathcal{D}_{\mathsf{sk}_i}(c)$.

- Un *algorithme de vérification de part* $\mathcal{V}$ qui prend en entrée une clé publique de vérification vk_i, un chiffré c, une part de déchiffrement σ_i et retourne $\mathcal{V}_{\mathsf{vk}_i}(c, \sigma_i) \in \{0, 1\}$.

- Un *algorithme de combinaison* $\mathcal{C}$ qui prend en entrée une clé publique de vérification vk, un chiffré c, un ensemble S de parts de déchiffrement, et retourne un déchiffré $m' = \mathcal{C}_{\mathsf{vk}}(c, S)$.

Tous ces algorithmes s'exécutent en temps polynomial en la longueur de leurs entrées avec la convention que les entrées de l'algorithme $\mathcal{K}$ sont encodées avec la notation unaire.

Modèle de sécurité d'un schéma sémantiquement sûr CPA dans un environnement distribué

On peut définir le jeu de la sécurité sémantique d'un adversaire montant une attaque CPA contre un système partagé comme le montre la figure 2.3.

On peut remarquer que l'attaquant peut obtenir les parts de déchiffré de certains messages. En effet, l'attaquant peut être un serveur. En conséquence, si le combineur envoie des chiffrés σ_i à déchiffrer, l'attaquant peut voir passer m_i, σ_i, $f(\sigma_i, \mathsf{sk}_i)$, pour $i > t$ pour des σ_i qu'il n'a pas choisis. On note par $f(.)$ la fonction de calcul d'une part de déchiffrement. Il est facile de simuler les parts des joueurs non corrompus car le simulateur connaît t parts et le clair donne la part en 0 qui est la $t+1$-ième part. Ainsi, la formule de Lagrange permet de reconstruire les parts des joueurs honnêtes dans le cas des schémas qui ont des propriétés homomorphiques comme l'exponentiation modulaire.

2.3.3 Exemple : Partage de déchiffrement El Gamal

Dans cette section, on va montrer que le schéma El Gamal distribué est sémantiquement sûr contre les attaques à clairs choisis. On dit aussi que ce système est IND-CPA.

La preuve va se faire par réduction. Dans un premier temps, supposons qu'il existe un adversaire $\mathcal{A}$ contre le schéma El Gamal à seuil et construisons un attaquant $\mathcal{B}$ contre le cryptosystème El Gamal non distribué (centralisé). On conclura par la même technique que précédemment, en disant que comme

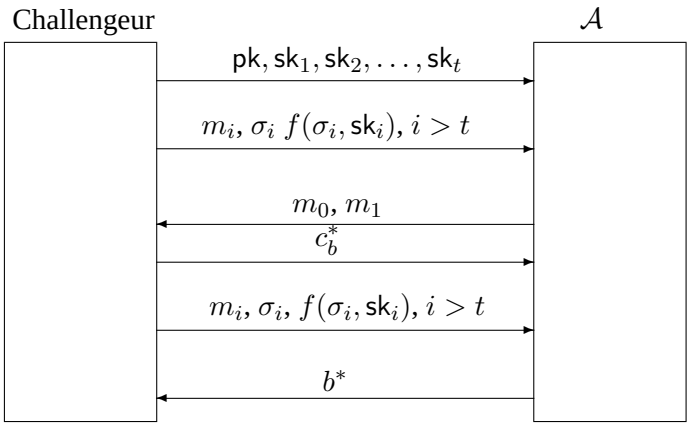


FIG. 2.3: *Modèle de sécurité sémantique face à une attaque à clairs choisis dans un environnement distribué.*

aucun attaquant $\mathcal{B}$ contre le schéma El Gamal centralisé n'est connu (cf. la section précédente), un tel adversaire $\mathcal{A}$ n'existe pas. La figure 2.4 présente ce type de preuve. L'attaquant $\mathcal{B}$ que l'on construit va permettre d'attaquer le schéma non distribué, alors que l'adversaire $\mathcal{A}$ est un attaquant contre le schéma distribué.

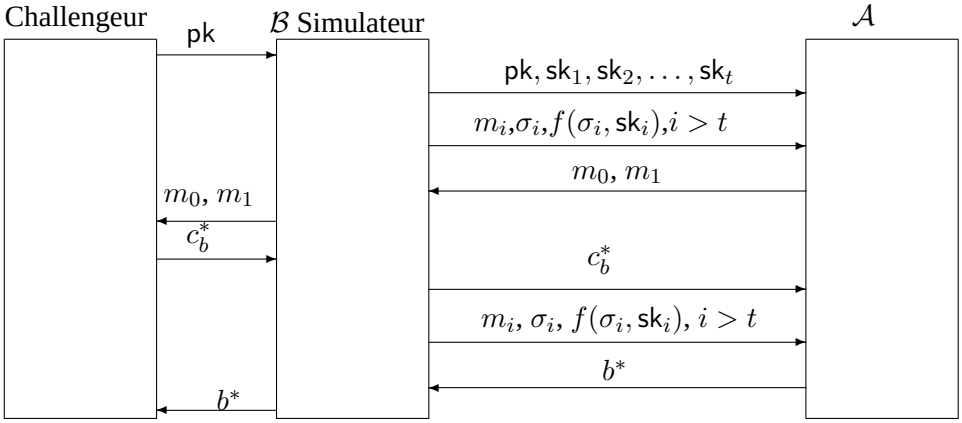


FIG. 2.4: *Modèle de preuve pour montrer la sécurité d'un système de chiffrement partagé sémantiquement sûr à partir d'un système de chiffrement sémantiquement sûr mais centralisé*

Soit le système de chiffrement El Gamal rappelé dans la section 1.4.4. Il est clair que l'avantage de l'attaquant $\mathcal{B}$ en devinant le bit b^* correctement est le même que celui de l'adversaire $\mathcal{A}$, donc $\varepsilon' = \varepsilon$ si l'on note $\varepsilon = Avantage = 2pr - 1$ où pr est la probabilité de succès de $\mathcal{A}$. Enfin, nous devons montrer que l'adversaire $\mathcal{A}$ ne peut pas distinguer les interactions avec un challengeur normal des interactions avec le simulateur qui ne connaît pas les parts des joueurs honnêtes. Pour ce faire, nous devons montrer qu'avec t parts $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_t$ de la clé secrète et m, nous pouvons calculer les parts des serveurs honnêtes à l'aide de la formule de Lagrange. En effet, soit un chiffré $(A, B) = (g^r, m \times y^r)$. Une part du déchiffré est de la forme A^{sk_j} car l'algorithme de combinaison, à partir des parts $p_i = A^{sk_i}$, calcule $\prod_i p_i^{\lambda_{0,i}^S} = g^{xr}$ où S est un ensemble de $t+1$ valeurs. Le déchiffrement se termine de la même façon que dans le cas non partagé. Pour simuler la part du serveur honnête j, le simulateur qui connaît m_i et

$\sigma_i = (A_i, B_i)$ calcule B_i/m_i qui vaut g^{xr} en théorie. Ainsi, pour calculer la part du serveur $j > t$, le simulateur calcule la part de déchiffré de j comme étant $(B_i/m_i)^{\lambda^S_{j,0}} \times \prod_{k=1}^{t} A_i^{\mathsf{sk}_k.\lambda^S_{j,k}} \bmod p$.

Remarque 10 ***El Gamal sémantiquement sûr contre les attaques à chiffrés choisis adaptatives.*** *Shoup et Gennaro ont construit et prouvé un schéma de chiffrement basé sur le cryptosystème El Gamal sémantiquement sûr dans le modèle de l'oracle aléatoire contre des attaques à chiffrés choisis adaptatives [175].*

2.3.4 Sécurité d'un schéma de signature

Dans cette sous-section, nous définissons un modèle formel pour un schéma de signature à seuil t parmi n. On suppose l'existence d'un trusted dealer et d'un ensemble de serveurs de signature $P_1, \ldots, P_n$.

Dans une *phase d'initialisation*, le distributeur crée une clé publique pk, une clé de vérification des parts vk, et des clés privées $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$. Pour $1 \leq i \leq n$, la clé privée sk_i est donnée au serveur P_i.

Un utilisateur, qui veut signer un message m, l'envoie aux serveurs en demandant une part de signature. Lorsqu'un utilisateur collecte les parts valides d'au moins $t + 1$ serveurs, il peut exécuter l'algorithme de combinaison pour obtenir la signature.

Une signature résultant d'un protocole de signature à seuil est la même que si elle avait été produite par un serveur unique. La validité de la signature peut être vérifiée par n'importe qui ayant la clé publique correspondante. En d'autres termes, le fait que la signature ait été produite par un protocole distribué est transparent au receveur.

De manière plus formelle, un schéma de signature à seuil comprend les algorithmes suivants.

– Un *algorithme probabiliste de génération de clés* $\mathcal{K}$ qui prend en entrée un paramètre de sécurité k, un nombre $n \geq 1$ de serveurs de signature, et le paramètre du seuil t $(1 \leq t < n)$; et retourne

$$(\mathsf{pk}, \mathsf{vk}, \mathsf{sk}) = \mathcal{K}(k, n, t)$$

où pk est *la clé publique de vérification de signature*, $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ est *la clé de vérification des parts*, et $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ est la liste de *clés secrètes*.

– Un *algorithme probabiliste de génération de part de signature* $\mathcal{S}$ qui prend en entrée la clé secrète sk_i, le message m, et retourne la part $\sigma_i = \mathcal{S}_{\mathsf{sk}_i}(m)$.

– Un *algorithme de vérification de part* $\mathcal{V}$ qui prend en entrée une clé publique de vérification vk_i, une part de signature σ_i, et retourne $\mathcal{V}_{\mathsf{vk}_i}(m, \sigma_i) \in \{0, 1\}$.

– Un *algorithme de combinaison* $\mathcal{C}$ qui prend en entrée une clé publique de vérification vk, un message m, un ensemble S de $t+1$ parts valides de signature, et retourne une signature $s = \mathcal{C}_{\mathsf{vk}}(m, S)$.

– Un *algorithme de vérification de signature* $\mathcal{V}'$ qui prend en entrée une clé publique pk, un message m, une signature s et retourne $\mathcal{V}'_{\mathsf{pk}}(m, s) \in \{0, 1\}$.

Tous ces algorithmes s'exécutent en temps polynomial en la longueur de leurs entrées avec la convention que les entrées de l'algorithme $\mathcal{K}$ sont encodées avec la notation unaire.

Les notions de sécurité pour les schémas de signature partagée sont les mêmes que celles de signature non partagée définies dans le chapitre 1 en prenant en compte des adversaires qui peuvent corrompre jusqu'à t serveurs de manière passive ou active.

La propriété de *robustesse* est implicitement définie dans la notion de signature et ressemble dans à la *consistance* dans un système de preuve interactive. On souhaite qu'avec forte probabilité, l'algorithme de génération de signature retournera une signature valide sur un message.

2.4 Partage proactif

2.4.1 Motivation

Dans un protocole partagé, une donnée sensible, clé de signature ou clé de déchiffrement, est partagée entre n serveurs de telle sorte qu'il est nécessaire que $t + 1$ serveurs soient présents pour signer ou déchiffrer alors que même si t serveurs sont corrompus par un attaquant, celui-ci ne pourra pas obtenir d'information sur la clé ou calculer une signature ou un déchiffrement. La sécurité proactive améliore la sécurité des protocoles partagés. Cette notion de sécurité découpe le temps en période de longueur T. Elle permet de garantir la sécurité du système contre des attaquants qui ont la capacité d'attaquer *tous* les acteurs et même plusieurs fois, mais ne peuvent en attaquer que t durant une période de temps T. Les protocoles proactifs sont plus résistants que les protocoles partagés dans lesquels l'attaquant peut corrompre au plus t serveurs durant toute la durée de validitié de la clé. Par exemple, ces adversaires peuvent être des virus informatiques ou un ensemble d'anciens employés mécontents qui ont eu accès à des parts d'information.

Ostrovsky et Yung ont montré comment une grande classe de problèmes de protocoles multiparties peut être résolue de manière proactive, dans un cadre où des canaux de communications sûrs sont disponibles [137]. Leur solution, basée sur le modèle général de calcul multiparties, a un intérêt théorique significatif, mais elle laisse la porte ouverte à des solutions pratiques plus efficaces pour des problèmes spécifiques.

Dans [39], l'approche proactive comme amélioration de la sécurité pour des systèmes centralisés a été considérée, et un générateur pseudo-aléatoire pratique proactif avec des applications pour des problèmes d'authentification sécurisée est présenté. Une autre fonction qui a été "proactivisée" est le partage de secret, et en particulier le partage de secret "vérifiable" (c'est-à-dire, le partage de secret résistant aux fautes intentionnelles) [105]. Cet algorithme joue un rôle clé dans les solutions proactives pour les cryptosystèmes à clé publique, et en particulier, dans les systèmes de signature proactive [104] (en étendant la signature à seuil de [60]). Des solutions proactives ont aussi été trouvées pour l'algorithme de signature DSS [88, 104] et pour RSA [78, 77].

Les signatures proactives sont un outil très puissants. Elles sont utilisées dans [38] pour fournir des solutions automatiques et proactives au rafraîchissement de clés. En particulier, [38] montre comment utiliser la cryptographie pour assurer des communications authentifiées et secrètes entre les serveurs, avec recouvrement contre les corruptions d'un certain nombre de serveurs. Ceci fournit une alternative au rafraîchissement manuel de clés.

2.4.2 Exemples de schémas proactifs

Nous décrivons maintenons rapidement deux techniques proactives : le partage de secret proactif et les signatures proactives.

Le partage de secret proactif

Pour maintenir la sécurité des schémas de partage de secret même en présence d'attaquants qui peuvent attaquer tous les serveurs, mais uniquement un nombre limité durant chaque période de temps, nous pouvons rafraîchir périodiquement (disons, chaque jour) la part du secret de chaque serveur. Le protocole de rafraîchissement doit garantir que les nouvelles parts sont *indépendantes des anciennes parts*, exceptées qu'elles *définissent le même secret*.

Soit par exemple le schéma de partage de Shamir déjà présenté précédemment. Si le secret est une valeur s dans l'ensemble des entiers $\{0, \ldots, p-1\}$ où p est un nombre premier, alors ce processus

peut être exécuté de la manière suivante [170]. Le distributeur (qui partage le secret) génère t nombres aléatoires $a_1, \ldots, a_t$ modulo p. Etant donné le polynôme $f(X) = s + a_1 X + \ldots + a_t X^t$, le distributeur donne au serveur i la *part* $s_i = f(i) \bmod p$. Il est clair que n'importe quel ensemble de t serveurs n'obtient aucune information sur s alors que tout ensemble de $t+1$ serveurs peut reconstruire la valeur par interpolation polynomiale.

Les rafraîchissements périodiques des parts peuvent être exécutés de la manière suivante. Chaque serveur i choisit un polynôme aléatoire $f_i(X)$ de degré t tel que $f_i(0) = 0$. Le serveur i envoie alors au serveur j la valeur $s_{ij} = f_i(j) \bmod p$. Le serveur j calcule alors sa nouvelle part rafraîchie $\hat{s}_j$ de la manière suivante :

$$\hat{s}_j = s_j + s_{1,j} + \ldots + s_{n,j} \bmod p$$

et efface son ancienne part. Il est facile de vérifier que les nouvelles parts $\hat{s}_i$ sont les valeurs du polynôme $\hat{f}(X) = f(X) + f_1(X) + \ldots + f_n(X)$ qui est toujours un polynôme de degrée au plus t et dont le terme constant est toujours s.

La procédure précédente fonctionne uniquement dans le cas où un adversaire passif peut lire le contexte de la mémoire mais ne peut pas la modifier ni changer le comportement d'un serveur. Dans le cas d'un attaquant actif, les techniques précédentes sont étendues en utilisant des protocoles de Partage de Secret Vérifiable (*Verifiable Secret Sharing*) [47]. En particulier, le protocole VSS de Feldman [68] est particulièrement adapté, et fournit en plus la capacité de retrouver les parts de clés corrompues et de les réinstaller (cf. [105] pour plus de détails).

Signatures proactives

La sécurité des cryptosystèmes à clé publique repose sur la sécurité et l'intégrité de la clé privée. Ainsi on doit ajouter à de tels schémas la protection de la clé privée tout en conservant la disponibilité du système comme la possibilité de signer ou de déchiffrer.

Une solution simple à ce problème peut être de partager la clé privée en utilisant un schéma de partage de secret proactif. Cette solution fournit la protection nécessaire aussi longtemps que la clé est utilisée. Cependant, afin de générer une signature, la clé privée doit être reconstruite dans un seul site et ainsi perdre l'avantage de la distribution : une attaque de ce site compromettra la sécurité. En revanche, un schéma de signature à seuil proactif permet aux serveurs de générer conjointement des signatures valides de manière à éviter un attaquant de générer de mauvaises signatures. En particulier, le schéma assure que la clé n'est jamais reconstruite dans un seul site.

Un schéma de signature proactif met en jeu trois phases : la phase de *génération de la clé* (effectuée de préférence sans distributeur de confiance), la phase de *génération conjointe de la signature* et finalement une phase spéciale de *rafraîchissement proactif de la part de clé* detenue par chaque serveur qui est effectuée périodiquement. La signature est générée de manière distribuée à partir des parts de la clé. Le schéma résiste à un attaquant qui peut corrompre tous les serveurs mais seulement un nombre limité (disons, la moitié) entre deux invocations du protocole de rafraîchissement. Des solutions proactives pour différents schémas de signature ont été élaborées et parmi elles une solution pour les signatures RSA et les signatures DSS. Le lecteur lira les articles suivants pour plus de précisions [104, 88, 78, 77].

Deuxième partie

Cryptographie à seuil

3

Partage du cryptosystème RSA

Dans ce chapitre, nous décrivons le partage du cryptosystème RSA. Dans l'article [75] qui sera présenté à la conférence Asiacrypt '01 avec Jacques Stern, nous avons proposé plusieurs algorithmes afin de distribuer *complètement* ce système de la phase de génération des clés à la phase de signature. Le but visé est la protection d'applications particulièrement sensibles et exigeant un fort niveau de sécurité où un distributeur de confiance ne peut pas être utilisé. Pour ces applications, nous sommes prêts à payer le prix en terme d'efficacité mais sans compromis en terme de sécurité.

Récemment, Victor Shoup a proposé un schéma de signature RSA à seuil permettant de partager la capacité de signer entre un ensemble de joueurs. Ce schéma permet aussi de distribuer le déchiffrement du système de chiffrement RSA. Cependant, le protocole de Shoup nécessite un distributeur de confiance pour produire et distribuer les clés. Ceci provient du fait que le schéma requiert des modules RSA spéciaux qui ne peuvent pas être générés de manière efficace entre plusieurs serveurs. Bien sûr, il est toujours possible de faire appel à des résultats théoriques de calcul multiparties puisque des compilateurs permettent de partager le calcul de n'importe quelle fonction de manière sûre, mais dans ce cas, nous ne pouvons pas espérer concevoir des protocoles efficaces en pratique. Le seul protocole efficace pour fabriquer des modules RSA entre plusieurs serveurs est le protocole de Boneh et Franklin [27] qui ne peut cependant pas être facilement modifié pour générer les modules RSA nécessaires dans le protocole de Shoup.

Nous expliquons dans un premier temps les algorithmes nécessaires à notre solution : le schéma de signature RSA à seuil de Shoup et l'algorithme partagé de génération de clé RSA de Boneh-Franklin. Puis dans un deuxième temps, nous décrivons notre solution du partage *complet* de RSA. Enfin, nous mentionnons la récente solution de Damgård et Koprowski.

Sommaire

3.1 Introduction

Les schémas de chiffrement et de signature RSA [162] sont très largement utilisés dans les systèmes actuels. Par exemple, de nombreux produits de sécurité bâtis sur une infrastructure à clés publiques (ICP) implémentent ce cryptosystème. Dans de tels produits, la protection de la clé racine de l'ICP nécessite un fort niveau de sécurité. Par conséquent, les protocoles à seuil peuvent être utilisés pour partager les capacités de signature parmi un sous-ensemble de personnes plutôt que de donner le pouvoir de signer à une seule d'entre elles. Dans un protocole à seuil, la clé secrète est produite puis partagée et chaque part est donnée à un serveur du groupe. Cependant, afin d'être sûr qu'à aucun moment la clé ne soit entièrement détenue par une machine et donc vulnérable aux attaques d'intrus internes ou externes (cf. chapitre 2), on peut aussi vouloir distribuer la phase de génération des clés. En conséquence, on dit qu'un schéma de signature ou un système de chiffrement est *complètement distribué* s'il est distribué de la génération de la clé à la phase de signature ou de déchiffrement.

Dans le cas des cryptosystèmes basés sur le logarithme discret, des solutions existent pour partager DSA [88, 118], El Gamal [60, 175] et Cramer-Shoup [37]. De plus, un protocole pour distribuer la clé a été initialement proposé par Pedersen dans [143]. Ce protocole a été ensuite modifié pour réparer une faille de sécurité. Nous verrons ce protocole et ces améliorations dans le chapitre 6. Par conséquent, les cryptosystèmes basés sur le logarithme discret sont complètement distribués. Cependant, une version de RSA *complètement distribuée* serait utile en pratique.

Nous proposons ici de nouvelles techniques pour distribuer complètement RSA. Ceci résout un problème ouvert. En effet, dans le cas du logarithme discret, les clés produites par l'algorithme de génération distribuée peuvent directement être utilisées dans l'algorithme de signature ou de déchiffrement à seuil. Dans le cas RSA, les clés produites par les algorithmes de génération partagée de clés RSA ne sont pas nécessairement de la forme spéciale exigée par le schéma de signature ou de déchiffrement à seuil.

D'une part, à Eurocrypt '00 [174], Shoup a décrit un schéma de signature RSA à seuil pratique nécessitant l'utilisation de modules RSA *sûrs*[37]. Il présente les caractéristiques intéressantes suivantes : il est sûr et robuste dans le modèle de l'oracle aléatoire sous l'hypothèse que le problème RSA est difficile. Puis, les phases de génération et de vérification des parts de signature sont complètement non-interactives et enfin, la taille d'une part de signature est bornée par une constante fois la taille du module RSA.

37. On dira qu'un module RSA, $N = pq$, est *sûr* si p et q sont des nombres premiers *sûrs*, c'est-à-dire de la forme $p = 2p' + 1$ et $q = 2q' + 1$ où p, q, p' et q' sont tous des nombres premiers. Attention, ceci ne veut pas dire que seuls les modules RSA de cette forme sont sûrs d'un point de vue sécurité. Pendant longtemps, on a cru qu'il fallait imposer des critères supplémentaires sur les nombres premiers des modules RSA, par exemple, que $p-1$ et $p+1$ aient tout deux au moins un gros facteur premier de 160 bits. L'analyse de Silverman [176] prouve que la seule exigence sur p et q est qu'ils soient de taille suffisamment grande.

D'autre part, Boneh et Franklin à Crypto '97 [27] ont décrit un protocole pour partager un module RSA. Cependant, il semble difficile d'adapter le protocole de Boneh-Franklin pour générer des modules RSA sûrs.

Pour résoudre ce problème, nous décrivons dans ce chapitre une méthode différente qui d'une part génère des clés RSA ayant des propriétés spécifiques en modifiant l'algorithme de Boneh-Franklin et qui d'autre part revisite l'algorithme de signature de Shoup afin de l'adapter aux modules ainsi produits.

Ces deux modifications diminuent certes les performances des protocoles de base, mais nous pensons que dans les applications sensibles, elles ne sont pas trop pénalisantes et fournissent des solutions efficaces en pratique. En effet, le protocole de génération de clé est normalement utilisé une seule fois et le nombre de serveurs nécessaires pour signer ou déchiffrer n'est pas très grand. De plus, ces serveurs ont du temps pour exécuter leur tâche. Nous veillerons cependant à ce que les complexités en communication [38] et en temps ne soient pas très grandes.

Indépendamment de notre travail, Damgård et Koprowski ont récemment considéré le même problème dans [57]. Ils ont revisité l'article de Shoup pour montrer, sous des hypothèses non standard, que la preuve de validité fonctionne sans exigence supplémentaire sur le module RSA. Comme algorithme de génération de clé, ils ont utilisé celui de Frankel, Mac Kenzie et Yung [80] qui est une version robuste de l'algorithme de Boneh-Franklin.

Dans notre travail, nous considérons des environnements où un niveau de sécurité élevé est demandé, comme dans le cas des schémas de vote électronique. Par conséquent, nous préférons utiliser des protocoles dont la preuve de sécurité est basée sur des hypothèses standards ; condition nécessaire pour construire des protocoles sûrs. Plusieurs schémas de vote électronique [72, 56, 1] ont été construits en utilisant le cryptosystème de Paillier afin de chiffrer les votes. Ce système de chiffrement est relié au cryptosystème RSA. Les techniques développées dans ce chapitre peuvent aussi être utilisées pour *distribuer complètement* ce système de chiffrement.

3.2 Signature RSA partagée

Dans cette section, nous expliquons tout d'abord l'évolution des différents schémas de partage de l'algorithme RSA. Puis, nous décrivons le schéma de signature RSA à seuil. Ensuite, nous montrons la sécurité de ce schéma contre des adversaires passifs et enfin, nous montrons la sécurité du schéma de Shoup contre des adversaires actifs.

3.2.1 Historique des schémas de partage RSA

Rappelons le modèle de communication. Quand un message m doit être signé par un quorum d'au moins $t+1$ serveurs, où $2t+1 \leq n$, un serveur spécial, appelé le *combineur*, transmet à l'ensemble des serveurs le message m (ou $x = H(m)$ avec $H(.)$ une fonction de hachage suivie d'une fonction de padding). Puis, chaque serveur calcule sa part de signature et génère une preuve de validité. Le combineur récupère ensuite les parts de signature et les preuves de validité, et vérifie ces dernières pour sélectionner un sous-ensemble de $t+1$ serveurs. Il recombine enfin les $t+1$ parts de signature correctes pour générer la signature s. Le combineur peut être un serveur spécial ou n'importe quel utilisateur.

Le premier problème à résoudre pour distribuer RSA en utilisant un partage polynomial est le calcul d'inverse modulo l'ordre des éléments [60]. Dans le cas du logarithme discret, q est l'ordre du groupe G généré par g. Comme q est public, il est facile de calculer des inverses $\bmod q$. Avec RSA, nous ne

38. La complexité en communication représente la quantité de bits transmis ainsi que le nombre d'échanges entre les serveurs.

pouvons pas dévoiler les inverses modulo $\mod \varphi(N)$ sans révéler aussi la factorisation de N [39], à moins d'utiliser une structure algébrique spéciale, appelée un module, comme dans [165, 89]. Les calculs dans une telle structure peuvent être effectués efficacement [115]. Si nous ne voulons pas utiliser de module, nous devons résoudre le problème du calcul d'inverse lorsqu'un partage polynomial est utilisé afin de calculer avec les coefficients de Lagrange.

Partage additif

Frankel, Mac Kenzie et Yung [78] et Rabin [159] ont proposé des partages additifs pour éviter ces calculs. Dans un partage additif, $d = \sum_{i=1}^{n} d_i \mod \varphi(N)$, le combineur calcule facilement la signature s à partir de n parts de signature correctes $\sigma_i = x^{d_i} \mod N$, où $x = H(m)$ est le message à signer, en utilisant la formule [40] :

$$s = \prod_{i=1}^{n} \sigma_i \left(= \prod_{i=1}^{n} x^{d_i} = x^{\sum_{i=1}^{n} d_i} = x^d \right) \mod N \tag{3.1}$$

La conséquence de la formule précédente est que les protocoles qui utilisent un partage additif ont alors besoin de générer *toutes* les parts de signature, les n parts, pour calculer s. Pour ce faire, les articles [78] et [159] présentent différentes stratégies permettant de reconstituer les parts des serveurs corrompus.

Partage de Rabin. Pour reconstruire ces parts, il est possible d'utiliser un double partage : un partage additif pour d et un partage polynomial de chaque part d_i comme le propose Rabin. Ainsi, si une part doit être reconstruite, $t+1$ serveurs honnêtes peuvent calculer la part de signature à l'aide du partage polynomial de la part de la clé du serveur corrompu qu'ils détiennent. Dans ce cas, pour reconstruire σ_i, les serveurs commencent par reconstruire d_i puis calculent $\sigma_i = x^{d_i} \mod N$. Chaque serveur doit pouvoir reconstruire d_i et vérifier les parts $d_{i,j}$ des autres serveurs. Rabin utilise un schéma de partage de secret publiquement vérifiable à la Feldman [68] dans $\mathbb{Z}_N^*$. Ainsi, les serveurs transmettent leurs parts $d_{i,j} = f_i(j)$, où f_i représente le polynôme qui a permis de partager d_i, ce qui permet de reconstruire d_i. En effet, tout les serveurs peuvent vérifier que

$$g^{f_i(j)} = \prod_{k=0}^{t} A_{i,k}^{j^k} \mod N$$

où $A_{i,k}$ correspond à $g^{a_{i,k}}$ avec $a_{i,k}$ le k-ième coefficient du polynôme f_i utilisé dans le partage à la Shamir de d_i. Pour plus de détails sur le schéma de partage publiquement vérifiable de Feldman, le lecteur pourra se référer au chapitre 2.

Partage additif de Frankel *et al.* Une autre possibilité est d'utiliser plusieurs partages additifs comme le proposent Frankel, Mac Kenzie et Yung dans [78] de telle sorte qu'il n'y ait pas de défaillances au moment de signer, ni d'adversaire qui puisse reconstituer toutes les parts d_i de d. Dans ce type de partage, chaque serveur appartient à plusieurs comités et détient plusieurs parts additives d'une même clé mais issues de partages additifs différents. Il y a autant de partages additifs qu'il y a de comités. Une famille

39. En effet, à partir de $y = x^{-1} \mod \varphi(N)$ et de x, on peut calculer xy tel que $xy = 1 \mod \varphi(N)$. L'entier $xy - 1$ est donc un multiple de $\varphi(N)$. Enfin, un algorithme dû à Miller [126, 154] permet de factoriser tout nombre N à partir d'un multiple de $\varphi(N)$.

40. Dans les formules suivantes, on note entre parenthèses certains calculs pour vérifier l'exactitude des calculs effectués avec des valeurs correctes. Cependant, nous ne savons pas, lorsque l'on calcule la partie qui n'est pas entre parenthèses, si les valeurs utilisées sont correctes.

contient plusieurs comités et chaque comité contient plusieurs serveurs. La répartition des serveurs dans les comités est probabiliste. Les familles et comités sont tels que :

- pour chaque famille, il y a toujours au moins un comité sans aucun serveur corrompu, appelé comité *excellent*,
- dans 90% des familles, tous les comités ont au moins un serveur non corrompu. Une famille F ayant cette propriété est dite *bonne*.

Avantages et inconvénients. Les principaux inconvénients de ces techniques sont la taille des parts de clé : en effet, plusieurs partages sont nécessaires et les protocoles permettant de reconstruire les parts de signature modifiées par l'adversaire actif ne sont pas efficaces. L'avantage du protocole de Rabin [159] est d'être très efficace dans le cas où il y a pas d'attaquant actif. En présence de tels attaquants, le surcoût n'est pas trop important.

En revanche, le partage additif a besoin impérativement de la part de signature de chaque personne : si une personne ne veut pas signer le message, la signature ne peut pas être produite avec le protocole [78], alors qu'elle peut l'être avec le protocole [159]. Cependant, dans ce cas, la génération de la signature n'est pas efficace. De plus, dans le cas où une personne ne veut pas signer le message, sa part de la clé secrète est dévoilée, ce qui l'élimine des futures signatures. Il faut alors proactiver le système, c'est-à-dire générer un nouveau partage de la clé, comme le propose Rabin. Ainsi, les protocoles à seuil sont plus résistants en présence d'attaquants actifs et permettent de générer des signatures même si des personnes ne veulent pas signer le document transmis par le combineur ou par un utilisateur du système. Dans le cas des protocoles de signature à seuil, Shoup a introduit un second paramètre indiquant le nombre de serveurs minimal ou qui doivent être d'accord pour signer un message. Par défaut, dans un schéma à seuil, ce nombre est fixé à $t+1$, mais il peut être arbitrairement choisi $\geq t+1$.

Partage polynomial

Soit S un sous-ensemble de $t+1$ serveurs. Les coefficients de Lagrange sont calculés par la formule : $\lambda'^S_{i,j} = \prod_{j' \in S \setminus j} \frac{i-j'}{j-j'} \bmod \varphi(N)$. Ainsi, on peut reconstruire la clé secrète d en utilisant la formule de Lagrange : $d = \sum_{i \in S} \lambda'^S_{0,i} d_i \bmod \varphi(N)$. Il y a plusieurs difficultés ici. D'une part, 2 divise $\varphi(N)$ et donc $(j-j')$ n'est pas toujours inversible modulo $\varphi(N)$. Si on note $m = \varphi(N)/4$ et que m ne contienne pas de petits facteurs, alors on peut effectuer le calcul des coefficients de Lagrange modulo m (cf. solution de Shoup). La deuxième solution est de considérer formellement la définition des $\lambda'_{i,j}$ dans $\mathbb{Z}$ (cf. solution de Frankel *et al.*). D'autre part, le combineur doit calculer

$$s = \prod_{i \in S} \sigma_i^{\lambda'^S_{0,i}} \left(= \prod_{i \in S} s^{\lambda'^S_{0,i} d_i} = s^{\sum_{i \in S} \lambda'^S_{0,i} d_i} = s^d \right) \bmod N \tag{3.2}$$

Si on ne peut pas dévoiler les $\lambda'^S_{i,j}$, car ils contiennent des inverses (cf. la 39-ième note de bas de page), le combineur doit calculer $\sigma_i^{\lambda'^S_{0,i}}$. Autant, il peut aisément calculer $\sigma_i^{\prod_{j' \in S \setminus j}(-j')} \bmod N$, autant le calcul de $\sigma_i^{\prod_{j' \in S \setminus j} \frac{1}{(i-j')}} \bmod N$ ne peut pas être effectué sauf si le combineur sait calculer des racines $\prod_{j' \in S \setminus \{j\}}(i-j')$ modulo une quantité composée, ce qui correspond à résoudre le problème RSA. Par conséquent, le combineur ne sait pas calculer l'équation (3.2).

Frankel, Gemmel, Mc Kenzie, et Yung dans [77], ont proposé le premier schéma prouvé sûr basé sur un partage polynomial et qui utilise le schéma de Desmedt et Frankel [61]. Cependant, dans le cas d'adversaires actifs, on peut avoir besoin de recommencer t fois pour réussir afin d'éliminer les mauvais serveurs car les parts de signature dépendent du sous-groupe de $t+1$ serveurs non corrompus.

L'idée principale de Frankel, Gemmel, Mac Kenzie et Yung, qui a été utilisée auparavant par Beaver et So [3], pour distribuer le générateur de bits pseudo-aléatoires Blum-Blum-Shub [21] est de remarquer que les $\Delta \times {\lambda'_{i,j}}^S$, avec $\Delta = n!$, sont des entiers. En effet, $\Delta \times {\lambda'_{i,j}}^S = n! \prod_{j' \in S \setminus \{j\}} \frac{(i-j')!}{(j-j')!}$ où $S \subset \{0, \ldots, n\}$. Or $(j - j') \in \{1, \ldots, n\}$ pour $j' \in S \setminus \{j\}$ et ne prend jamais deux fois la même valeur. De plus, il y a $t < n$ valeurs $(j - j')$ et donc $\prod_{j' \in S \setminus \{j\}} (j - j')$ divise $n!$.

Les parts d_i de d sont telles que Δ divise d_i et $d_i = f(i)$ où f est un polynôme de degré t et de terme constant égal à Δd. Pour ce faire, $\Delta | a_i$ où les a_i sont les coefficients du polynôme. Par conséquent, si nous écrivons $d_i = \Delta \times d'_i$, alors, pour un sous-ensemble S de $t + 1$ serveurs, chacun d'entre eux peut calculer $\ell^S_{0,i} = \Delta \times {\lambda'}^S_{0,i} \in \mathbb{Z}$ et ${\sigma'}_i = x^{{\lambda'}^S_{i,0} d_i} = x^{\ell^S_{0,i} \times d'_i} \bmod N$. Enfin, le combineur utilise $t+1$ parts pour calculer

$$s = \prod_{i \in S} {\sigma'}_i \left(= \prod_{i \in S} x^{\ell^S_{0,j} \times d'_i} = \prod_{i \in S} x^{\Delta \times {\lambda'}^S_{0,i} \times d'_i} = \prod_{i \in S} x^{{\lambda'}^S_{0,i} d_i} = x^d \right) \bmod N \tag{3.3}$$

Si la signature n'est pas valide, le combineur élimine les mauvais serveurs à l'aide de la preuve de validité, définit un autre groupe S et recommence le protocole. Il est donc évident qu'après t essais, tous les mauvais serveurs seront éliminés de S et la signature sera correcte. Cependant, cette redéfinition du sous-ensemble S ne paraît pas idéale. Shoup dans [174] en même temps que Miyazaki, Sakurai et Yung [127] ont alors proposé de nouvelles méthodes pour éviter ce problème.

Solution de Shoup

Shoup a résolu le problème précédent en utilisant un lemme [102] permettant d'extraire sans aucun secret une racine e-ième de w modulo un nombre composé à partir d'une racine e-ième d'une puissance connue de w. Cette solution consiste à multiplier les coefficients de Lagrange par Δ pour les rendre entiers : $\lambda^S_{i,j} = \Delta \times {\lambda'}^S_{i,j} \in \mathbb{Z}$ et $\Delta d = \sum_{i \in S} \lambda^S_{0,i} d_i$. Ainsi, si nous posons $\sigma_i = x^{d_i}$, le combineur peut calculer des signatures et changer de sous-ensemble (calculer les nouveaux coefficients de Lagrange suivant le nouveau sous-ensemble de $t + 1$ serveurs) sans demander de nouvelles parts de signature aux serveurs. Il calcule l'équation (3.2) en utilisant $\lambda^S_{i,j}$ et obtient $s^\Delta = \prod_{i \in S} \sigma_i^{\lambda^S_{0,i}} (= x^{\Delta d}) \bmod N$. Le combineur peut ainsi calculer s^Δ une racine e-ième de x^Δ avec la formule précédente ($s^{\Delta e} = x^\Delta$) et peut générer une racine e-ième de x en utilisant le lemme que nous verrons en section 3.2.2. Par conséquent, si on utilise le schéma de Shoup, il n'est plus nécessaire de générer d_i tel que $\Delta | d_i$ comme cela est fait dans [80, 57]. On peut remarquer que cette méthode fonctionne pour tout N.

Même si le protocole de Frankel *et al.* dans [77] propose une version de RSA complètement distribuée, il est moins élégant que le schéma de Shoup en présence d'adversaires actifs. De plus, ce dernier propose d'autres améliorations importantes pour la preuve de validité. Par conséquent, on doit maintenant résoudre le problème de la distribution de la preuve de validité non-interactive puisque Shoup a résolu celui de la distribution de la signature.

Preuve de robustesse et utilisation de nombres premiers sûrs

Comme on l'a vu précédemment, le second point important dans le schéma de signature de Shoup est la preuve de validité qui garantit la robustesse du schéma. La robustesse signifie que des serveurs corrompus ne peuvent pas empêcher les serveurs non corrompus de signer. Cette propriété n'a de sens qu'en présence d'adversaires actifs qui peuvent modifier le comportement des serveurs corrompus. La preuve de validité demande des modules RSA construits avec des nombres premiers sûrs, (tels que p

et $\frac{p-1}{2}$ sont tous les deux premiers) et permet à chaque serveur de prouver qu'il a élevé x à la bonne puissance d_i leur part de la clé secrète[41].

Pour cela, chaque serveur i détient une clé de vérification $\mathsf{vk}_i = \mathsf{vk}^{d_i} \bmod N$ et fait une preuve que $\log_{\mathsf{vk}} \mathsf{vk}_i = \log_x \sigma_i (= d_i \bmod \varphi(N))$. Des preuves d'égalité de logarithmes sont connues dans $\mathbb{Z}_p^*$ ou modulo un ordre connu. Cependant, il est plus difficile de prouver ces protocoles dans $\mathbb{Z}_N^*$. Les problèmes sont : $\mathbb{Z}_N^*$ n'est pas un groupe cyclique (et donc si x est pris au hasard dans $\mathbb{Z}_N^*$, on ne sait même pas si le logarithme de ce nombre existe), puis l'ordre de ce groupe $\varphi(N)$ est inconnu pour le prouveur, ensuite un générateur v n'existe pas, et enfin, un élément d'ordre maximal, $\lambda(N)$, ne peut pas être facilement trouvé.

Shoup a remarqué qu'en utilisant des modules RSA sûrs, le groupe des carrés de $\mathbb{Z}_N^*$, noté Q_N dorénavant, est cyclique, et il est alors facile de trouver des générateurs. De plus, la preuve d'égalité de logarithme discret dans $\mathbb{Z}_p^*$, preuve non-interactive de Chaum-Pedersen [44], peut être adaptée dans Q_N et prouvée sûre dans le modèle de l'oracle aléatoire. Enfin, ces modules RSA sûrs sont aussi utiles pour le protocole de génération de clé afin de garantir la sécurité du partage de secret de Shamir.

Ceci pose donc la question de la génération de modules RSA pour le schéma de signature à seuil de Shoup sans distributeur de confiance.

3.2.2 Schéma de signature RSA à seuil de Shoup

Dans cette section, nous rappelons le schéma de signature à seuil de Shoup [174].

Algorithme de génération de clés. Le distributeur choisit deux nombres premiers sûrs $p = 2p' + 1$ et $q = 2q' + 1$ tels que p' et q' soient également premiers; le module RSA est $N = pq$ et l'exposant public est e un nombre premier plus grand que le nombre de serveurs n: $\mathsf{pk} = (N, e)$. Soit $m = p'q'$. Le distributeur calcule alors la clé secrète $\mathsf{sk} = d \in \mathbb{Z}_m$ telle que $de = 1 \bmod m$. La clé secrète sk est partagée grâce à un partage de secret à la Shamir : soit $f_0 = d$ et, pour $i = 1, \dots t$, f_i est choisi aléatoirement dans $\mathbb{Z}_m$. Soit $f(X) = \sum_{i=0}^{t} f_i X^i$; la clé secrète sk_i est $d_i = f(i) \bmod m$. Finalement, le distributeur choisit aléatoirement v dans le sous-groupe cyclique des carrés dans $\mathbb{Z}_N^*$ et calcule les clés de vérification $\mathsf{vk} = v^{\Delta}$ et, pour $i = 1 \dots n$, $\mathsf{vk}_i = v^{\Delta d_i} \bmod N$.

Algorithme de signature. Pour signer un message M, le signataire calcule $x = H(M)$ et envoie x au combineur qui le transmet aux serveurs. Chaque serveur calcule $\sigma_i = x^{2\Delta d_i} \bmod N$ et génère une preuve de validité. Elle va permettre de convaincre le combineur, ou n'importe quel autre serveur, que le logarithme discret de σ_i^2 en base $\tilde{x} = x^{4\Delta}$ est le même que le logarithme discret de vk_i dans la base vk, soit la clé secrète $\mathsf{sk}_i = d_i$. Une telle preuve non-interactive est proposée dans la section 3.2.4. Le combineur vérifie les parts correctes et en choisit $t + 1$ pour générer la signature grâce à l'algorithme de combinaison.

Algorithme de combinaison. Si moins de $t + 1$ parts de signature ont des preuves de validité correctes, l'algorithme échoue. Sinon, soit S un ensemble de $t+1$ parts valides; pour n'importe quel $i \in \{0, \dots n\}$ et $j \in S$, nous définissons les coefficients de Lagrange :

$$\lambda_{i,j}^S = \Delta \times \frac{\prod_{j' \in S \setminus \{j\}} (i - j')}{\prod_{j' \in S \setminus \{j\}} (j - j')} \in \mathbb{Z}$$

41. Une première solution pour cette preuve de validité serait de dévoiler les clés publiques $e_i = d_i^{-1} \bmod \varphi(N)$ associées à chaque d_i et de vérifier si ${\sigma_i}^{e_i} \stackrel{?}{=} x \bmod N$. Le problème de cette solution est que la connaissance une paire de clé (e_i, d_i) permet de factoriser car e_i est un inverse modulo $\varphi(N)$ d'une quantité connue du serveur P_i. Par conséquent, chaque serveur pourrait factoriser le module RSA. Le lecteur pourra lire aussi l'annexe 10 qui présente une autre méthode permettant au serveur P_i de retrouver d_j.

La formule d'interpolation de Lagrange implique que :

$$\Delta f(i) = \sum_{j \in S} \lambda_{i,j}^{S} f(j) \bmod m \tag{3.4}$$

En utilisant le fait que par définition, $f(0)$ est égal à la clé secrète d, on peut obtenir une signature de x. Pour cela, remarquons que :

$$w = x^{4\Delta^2 d} = x^{4\Delta^2 f(0)} = \prod_{j \in S} x^{4\Delta \lambda_{0,j}^{S} f(j)} = \prod_{j \in S} \sigma_i^{2\lambda_{0,j}^{S}} \bmod N$$

Ainsi, $w^e = x^{4\Delta^2} \bmod N$. La signature s est telle que $s^e = x \bmod N$. Comme la quantité $4\Delta^2$ est publique et comme l'exposant public e est un nombre premier plus grand que n, il est relativement premier avec $4\Delta^2$ (car $\Delta = n!$). Ainsi, l'algorithme d'Euclide étendu trouve deux entiers a et b tels que $a \times 4\Delta^2 + b \times e = 1$. Ceci permet d'obtenir la signature $s = w^a \times x^b \bmod N$. En effet, on a : $s^e = w^{ae} \times x^{eb} = x^{a \times 4\Delta^2} \times x^{eb} = x^{a \times 4\Delta^2 + eb} = x \bmod N$.

Le premier Δ provient de la formule 3.4 et le deuxième Δ provient de la définition de σ_i. Le Δ dans la définition de σ_i est utile pour simuler les parts σ_j à partir du chiffré et de t valeurs σ_i.

Algorithme de vérification d'une signature. L'algorithme de vérification est le même que celui d'une signature RSA classique.

3.2.3 Preuve de sécurité du schéma de Shoup contre des adversaires passifs

Théorème 7. Le schéma de partage de secret est sûr.

Preuve: Pour n'importe quel sous-ensemble de t points dans $[0, n]$, la valeur de $f(X)$ modulo m en ces points détermine de manière unique les coefficients de $f(X)$ modulo m, et ainsi la valeur de $f(X)$ modulo m en n'importe quel autre point modulo m dans $[0, n]$. Ceci est une conséquence du fait que la matrice de Vandermonde (cf. section 2.2.3) est inversible modulo m, car son déterminant est relativement premier avec m. Par conséquent, pour tout sous-ensemble de t points dans $[0, n]$, les distributions de la valeur $f(X)$ modulo m en ces points sont uniformes et mutuellement indépendantes. □

Le schéma de signature à seuil de Shoup est sûr contre des adversaires passifs. Shoup a montré le théorème suivant dans [174].

Théorème 8. Dans le modèle de l'oracle aléatoire [9], le protocole précédent est un schéma de signature à seuil sûr (robuste et non-forgeable) sous l'hypothèse que le schéma de signature RSA est sûr (RSA-FDH ou RSA-PSS).

Preuve: Nous commençons par montrer comment simuler la vue de l'adversaire, étant donné un accès à l'oracle de signature RSA que l'on utilise seulement quand l'adversaire demande une part de signature pour un joueur non-corrompu.

Soit $i_1, \ldots, i_t$ l'ensemble des joueurs corrompus. On rappelle que $d_i \equiv f(i) \bmod m$ pour tout $1 \leq i \leq n$, et $d \equiv f(0) \bmod m$.

Pour simuler la vue de l'adversaire, on choisit les d_{i_j} appartenant à l'ensemble des joueurs corrompus au hasard dans l'ensemble $\{0, \ldots, \lfloor N/4 \rfloor - 1\}$. Nous avons déjà dit que les parts de la clé secrète détenues par les serveurs corrompus sont choisies dans l'ensemble $\{0, \ldots, m-1\}$. Nous avons :

$$N/4 - m = (p' + q')/2 + 1/4 = O(N^{1/2})$$

et à partir de cela, un calcul montre que la distance statistique entre la distribution uniforme sur $\{0,\ldots,\lfloor N/4\rfloor-1\}$ et la distribution uniforme sur $\{0,\ldots,m-1\}$ est $O(N^{-1/2})$. En effet,

$$\begin{aligned}\sum_{\alpha}\left|\Pr_{X\in[0,m[}[X=\alpha]-\Pr_{Y\in[0,\lfloor N/4\rfloor[}[Y=\alpha]\right| &= \sum_{\alpha=0}^{m-1}\left|\frac{1}{m}-\frac{1}{m+\delta}\right|+\sum_{\alpha=m}^{m+\delta}\left|0-\frac{1}{m+\delta}\right| \\ &= \frac{\delta}{m+\delta}+\frac{\delta}{m+\delta}=\frac{2\delta}{m+\delta}=O(N^{-1/2})\end{aligned}$$

où $\lfloor\frac{N}{4}\rfloor=m+\delta$ et donc $\delta=(p'+q')/2$.

Une fois que ces valeurs d_{i_j} sont choisies, les valeurs d_i pour les serveurs non-corrompus sont aussi complètement déterminées modulo m, mais elles ne peuvent pas être facilement calculées. Cependant, étant donnés $x, s\in\mathbb{Z}_N^*$ avec $s^e=x$, nous pouvons facilement calculer $\sigma_i=x^{2\Delta d_i}$ pour un serveur i non-corrompu de la manière suivante :

$$\sigma_i = s^{2(\lambda^S_{i,0}+e(\lambda^S_{i,i_1}d_{i_1}+\ldots+\lambda^S_{i,i_t}d_{i_t}))}$$

où $S=\{0,i_1,\ldots,i_t\}$. Ceci est une conséquence de l'équation (3.4).

En utilisant ces techniques, on peut générer les valeurs $\mathsf{vk},\mathsf{vk}_1,\ldots,\mathsf{vk}_n$, et donc générer n'importe quelle part de signature σ_i, étant donnée une signature standard RSA.

Cet argument justifie la définition de la part de signature σ_i à $x^{2\Delta d_i}$, au lieu de x^{2d_i} par exemple.

On a montré que tous les messages pouvaient être simulés de manière efficace et indistinguable d'un jeu réel, c'est-à-dire en présence des vraies serveurs qui connaissent leur part de la clé secrète. Ainsi, un adversaire qui obtient t parts de la clé secrète n'apprend pas d'information supplémentaire qui lui permettrait par exemple d'obtenir une $(t+1)$-ième part de la clé et de reconstruire toute la clé secrète. Le protocole RSA distribué est donc sûr face à un tel adversaire. □

3.2.4 Preuve de robustesse contre des adversaires actifs

Il s'agit de faire une preuve d'appartenance à un langage. En effet, on n'a pas besoin de prouver que les serveurs connaissent la clé secrète d_i, mais seulement qu'ils ont fait leur travail correctement, *i.e.* $\sigma_i=x^{d_i} \bmod N$.

Dans ce but, on peut faire des preuves d'appartenance à un langage difficile. Si le mot, (σ_i,vk_i), composé de la part de signature σ_i et d'informations connues par le vérifieur n'est pas dans le langage, alors la significativité de la preuve prouvera que le prouveur ne peut pas construire une telle preuve. Le vérifieur est sûr que les données vk_i sont correctes car elles sont garanties dans le protocole de génération de clés et donc $\mathsf{vk}_i=\mathsf{vk}^{d_i} \bmod N$.

Dans cette section, nous présentons la preuve de validité de Shoup [174], preuve d'égalité de logarithmes discrets dans un groupe cyclique d'ordre inconnu. Nous présentons en chapitre 9 deux autres preuves de validité. Chacune d'entre elles a ses avantages, mais nous nous intéressons ici à la preuve de Shoup qui est non-interactive et qui ne suppose aucune relation particulière entre le prouveur et le vérifieur.

Preuve non-interactive d'égalité de logarithmes discrets dans un groupe cyclique d'ordre inconnu

Soit Q_N un groupe cyclique d'ordre inconnu $m=p'q'$. Soit v un générateur de Q_N et σ_i^2 un élément de Q_N. Une preuve d'égalité du logarithme discret d'un élément σ_i^2 en base $\tilde{x}=x^{4\Delta}$ et d'un autre élément vk_i en base vk peut être conçue en utilisant le protocole de Chaum et Pedersen [44] sans que l'ordre de Q_N n'ait besoin d'être connu. Soit L_1 la taille de la fonction de hachage $H(.)$. Décrivons la version non-interactive de la preuve.

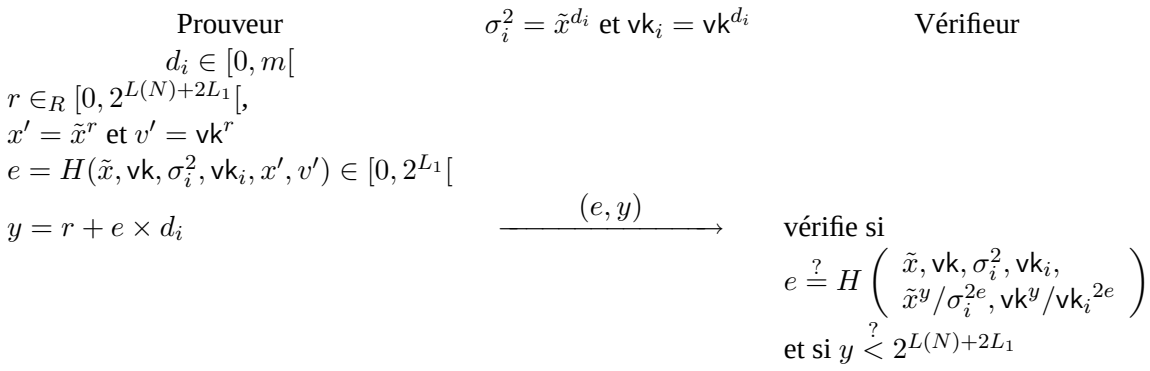


FIG. 3.1: *Preuve de validité non-interactive de Shoup.*

Description de la preuve. Soit d_i le logarithme discret commun et soit r un élément choisi au hasard dans $[0, 2^{L(N)+2L_1}[$. La preuve calcule $v' = \mathsf{vk}^r$ et $x' = \tilde{x}^r$. Soit e la valeur de la fonction de hachage $H(\tilde{x}, \mathsf{vk}, \sigma_i^2, \mathsf{vk}_i, x', v')$ où H est une fonction de hachage qui prend ses valeurs de sortie dans l'intervalle $[0, 2^{L_1}[$. On calcule ensuite $y = r + e \times d_i$. Une preuve d'égalité de logarithmes discrets est une paire $(e, y) \in [0, 2^{L_1}[\times[0, 2^{L(N)+2L_1}[$; elle est correcte si elle vérifie les équations $e = H(\tilde{x}, \mathsf{vk}, \sigma_i^2, \mathsf{vk}_i, \tilde{x}^y/\sigma_i^{2e}, \mathsf{vk}^y/{\mathsf{vk}_i}^e)$ et $y < 2^{L(N)+2L_1}$.

Remarque. Quand le vérifieur reçoit σ_i, il ne sait pas si σ_i est un carré ou non. Pour en être certain, il utilise σ_i^2 et pas simplement σ_i.

Preuve: Consistance. La consistance de ce schéma est claire pour la première équation de vérification. Pour la deuxième équation, on montre que la probabilité que la preuve échoue est $2^{-L_1} \geq \frac{2^{L_1} \times m}{2^{L(N)+2L_1}}$.

Significative. On peut montrer que la preuve est significative. On décrit la preuve de sécurité de Shoup [174] dans le modèle de l'oracle aléatoire. Si une preuve (e, y) est valide, alors soit $e = H(\tilde{x}, \mathsf{vk}, \sigma_i^2, \mathsf{vk}_i, \tilde{x}^y/\sigma_i^{2e}, \mathsf{vk}^y/{\mathsf{vk}_i}^e)$, soit $x' = \tilde{x}^y/\sigma_i^{2e}$ et $v' = \mathsf{vk}^y/\mathsf{vk}_i^e$. Comme Q_N est un groupe cyclique généré par vk, il existe a, b, c et d des entiers tels que $\tilde{x} = \mathsf{vk}^a$, $\sigma_i^2 = \mathsf{vk}^b$, $v' = \mathsf{vk}^c$ et $x' = \mathsf{vk}^d$. En utilisant les définitions de v' et x', on obtient les équations

$$c = y - ed_i \bmod m$$

et

$$d = ay - be \bmod m$$

donc, en multipliant la première par a et en soustrayant la seconde, on obtient

$$ca - d = e(b - ad_i) \bmod m$$

Dans le modèle de l'oracle aléatoire, e est une valeur aléatoire, indépendante des entrées de la fonction de hachage et par conséquent,

$$b - ad_i = 0 \bmod m$$

Ainsi,

$$\sigma_i^2 = \mathsf{vk}^b = \mathsf{vk}^{ad_i} = \tilde{x}^{d_i}$$

De plus, comme on est sûr que $\mathsf{vk}_i = \mathsf{vk}^{d_i} \bmod N$, si le mot $(\sigma_i, \mathsf{vk}_i)$ est dans le langage $L = \{(\alpha, \beta) \text{ tq } \log_{\tilde{x}}(\alpha^2) = \log_{\mathsf{vk}}(\beta)\}$, alors un prouveur ne peut pas tromper un vérifieur avec probabilité supérieure à $\frac{1}{2^{L_1}}$ et on sait alors que $\sigma_i^2 = \tilde{x}^{d_i} = x^{4\Delta d_i}$.

Zero-Knowledge. Regardons maintenant la simulation zero-knowledge. Nous pouvons construire un simulateur qui simule la vue de l'adversaire sans connaître la valeur d_i. Cette vue comprend les valeurs de l'oracle aléatoire aux points où l'adversaire a questionné l'oracle de telle sorte que le simulateur soit complètement responsable de l'oracle aléatoire. Quand l'adversaire fait une requête à l'oracle, si l'oracle n'a pas encore défini la valeur en ce point, le simulateur la définit avec une valeur aléatoire et dans tous les cas retourne la valeur à l'adversaire. Quand un serveur non-corrompu est supposé générer une preuve de validité pour un message x, et une part de signature σ_i, le simulateur choisit $e \in [0, 2^{L_1}[$ et $y \in [0, 2^{L(N)+2L_1}[$ au hasard, et pour des valeurs données x et σ_i, il définit la valeur de l'oracle aléatoire au point $(\tilde{x}, \mathsf{vk}, \sigma_i^2, \mathsf{vk}_i, \tilde{x}^y/\sigma_i^{2e}, \mathsf{vk}^y/{\mathsf{vk}_i}^e)$ comme étant e. Avec probabilité écrasante, le simulateur n'a pas encore défini l'oracle à ce point auparavant, et il est donc libre de le faire. La preuve de validité est constituée de la paire (e, y). Il est simple de vérifier que la distribution produite par ce simulateur est statistiquement proche d'une distribution uniforme.

Cette preuve est seulement *statistiquement* zero-knowledge car $y = r + ed_i$ est calculé dans $\mathbb{Z}$. Par conséquent, les valeurs sur les bords de l'intervalle $[0, 2^{L(N)+2L_1}[$ sont moins souvent atteintes que les autres (cf.page 99 de la thèse de Guillaume Poupard [154]). □

3.3 Algorithme de génération partagée de clés RSA de Boneh-Franklin

Il existe de nombreux protocoles pour fabriquer des clés RSA de manière distribuée [27, 80, 48, 49, 18, 155, 92]. Boneh et Franklin dans [27] ont conçu un tel protocole dans le modèle *honnête-mais-curieux*, c'est-à-dire sans adversaire actif. Plus tard, Frankel, MacKenzie et Yung dans [80] ont rendu cet algorithme robuste contre des adversaires actifs.

Dans [155], Poupard et Stern ont aussi proposé un protocole pour produire un module partagé pour deux joueurs seulement car le protocole de Boneh et Franklin est $t = \lfloor \frac{n-1}{2} \rfloor$ sûr, ce qui donne $t = 0$ pour $n = 2$. Gilboa dans [92] a ensuite étendu la méthode de Poupard et Stern. Cette méthode utilise des mécanismes d'Oblivious Transfer (cf. chapitre 2) et en étendant l'OT de Poupard et Stern à plusieurs serveurs en faisant des OT 1-parmi-n. Cependant ces protocoles OT ne sont pas aussi efficaces que le protocole BGW, décrit dans l'annexe 9, et utilisé par Boneh et Franklin. Ce protocole comme le protocole de Cocks [48, 49] permet de tolérer $n - 1$ serveurs passifs. Malheureusement, la sécurité du protocole de Cocks repose sur un argument heuristique. Il a été rendu robuste par Blackburn *et al.* dans [18].

En conséquence, le protocole de Boneh et Franklin est le plus efficace des algorithmes de génération RSA partagée et présente le plus de garantie en ce qui concerne la sécurité.

3.3.1 Description

Nous décrivons le protocole de Boneh-Franklin dans la figure 3.2.

Le caractère pratique du test de biprimalité est basé sur les résultats empiriques de Rivest [161] montrant que si un crible est préalablement effectué, le test de primalité de Miller-Rabin n'est pas nécessaire, les nombres pseudoprimes étant rares selon des conjectures de Pomerance [152, 153].

3.3.2 Preuve de sécurité

Dans l'étape 1 de ce protocole, chaque serveur choisit un random p_i uniformément dans un intervalle $\left[\sqrt{2}.2^{L(N)/2-1}, \lfloor \frac{2^{L(N)/2}-1}{n} \rfloor\right[$ comme sa part secrète. Le nombre premier p résultant est pris comme la somme de ces valeurs. Comme la somme de variables aléatoires uniformes et indépendantes *n'est pas* une variable uniformément distribuée, p est choisi dans une distribution ayant moins d'entropie qu'une

1. Dans la première étape, chaque serveur choisit au hasard deux valeurs p_i et q_i dans l'intervalle $\left[\sqrt{2}.2^{L(N)/2-1}, \lfloor\frac{2^{L(N)/2}-1}{n}\rfloor\right[$ selon [176], où $L(N)$ est la taille en bits du module N.
2. Puis les serveurs utilisent l'algorithme distribué BGW, décrit dans l'annexe 9 pour calculer le produit N de $p_1 + \ldots + p_n$ et $q_1 + \ldots + q_n$.
3. Ensuite, les parties exécutent un test de biprimalité similaire au test de Fermat modulo N. Fixons $p = p_1 + \ldots + p_n$ et $q = q_1 + \ldots + q_n$ si le module est de la bonne forme, *i.e.*, $N = pq$. Ce test de biprimalité est $n - 1$ sûr.
4. Enfin, les serveurs utilisent un protocole pour générer un partage de la clé secrète.

FIG. 3.2: *Algorithme de génération distribuée de modules RSA*

distribution uniforme[42] sur un même intervalle. Ceci n'est pas un problème car une somme de variables aléatoires tend vers une gaussienne selon le Théorème de la Limite Centrale. On peut montrer que cette gaussienne est très étalée. On peut estimer la moyenne de la variable aléatoire somme S_i à $\frac{2^{\frac{L(N)-3}{2}}/n+1}{2}$ et de variance $\frac{2^{\frac{L(N)-3}{2}}/n^2-1}{12}$. Ceci permet de caractériser complètement la variance. Il est facile de voir que cette variance est suffisamment grande et qu'elle contient plus de 2^{128} nombres premiers car elle est supérieure à $> 2^{128} \times \log(2^{128})$.

Une deuxième conséquence de ce protocole est que chaque serveur peut obtenir l'information suivante : $p_i < p$. Boneh et Franklin ont montré que cette information n'est pas utile à un adversaire.

Dans le lemme 2.1. de [27], Boneh et Franklin réduisent le protocole distribué au protocole centralisé et montrent en particulier que s'il existe un attaquant $\mathcal{A}$ qui réussit à casser la sécurité du protocole distribué de génération de clé (*i.e.*, avec t parts de la clé secrète), il existe un simulateur $\mathcal{S}$ qui factorise une fraction non-négligeable des modules RSA N de taille $L(N)$.

Théorème 9. Supposons qu'il existe un algorithme en temps polynomial $\mathcal{A}$ qui étant donné : (1) un aléa $N \in \mathbb{Z}^{(2)}_{L(N)}$ choisi dans la distribution de $\mathbb{Z}^{(2)}_{L(N)}$ induite par le protocole, et (2) les parts $\langle p_i, q_i\rangle$ de t parties, factorise N avec probabilité au moins $1/L(N)^{\delta}$ pour un certain δ. Alors il existe un algorithme $\mathcal{B}$ fonctionnant en temps polynomial en moyenne qui factorise $1/(4(t+1)^3 L(N)^{\delta})$ des entiers dans $\mathbb{Z}^{(2)}_{L(N)}$.

3.4 Schéma complètement distribué de signature RSA à seuil

Nous présentons ici la solution que nous avons conçue pour distribuer la preuve de validitié et l'ensemble du schéma de signature RSA à seuil de Shoup. Nous commencerons par présenter le problème, c'est-à-dire à identifier les cas où les modules RSA sûrs sont nécessaires dans le schéma de Shoup, puis nous définirons le modèle de sécurité. Ensuite, nous décrirons comment modifier l'algorithme de

42. Dans le chapitre 6, nous montrerons qu'un adversaire peut biaiser cette distribution de façon à obtenir de l'information sur la clé secrète. Ceci est une attaque sur le protocole distribué de génération de la clé.

Boneh et Franklin afin qu'il génère des modules RSA ayant des propriétés spéciales. Nous montrerons que le schéma de Shoup est toujours résistant aux attaques passives avec les modules ainsi générés et nous montrerons que ce schéma est aussi robuste, c'est-à-dire sûr contre les attaques actives. Enfin, nous donnerons les paramètres pratiques pour notre schéma.

Soit $N = pq$ tel que $p = 2p' + 1$ et $q = 2q' + 1$ où en général $p' = \prod_i {p_i}^{e_i}$ et $q' = \prod_j {q_j}^{e_j}$. Posons $M = p'q'$. On rappelle qu'un nombre premier p est dit *nombre premier sûr* si p et p' sont simultanément premiers. Un module RSA $N = pq$ composé de deux nombres premiers sûrs est appelé un *module RSA sûr*.

3.4.1 Problématique

Comme on va le voir dans la suite, les nombres premiers sûrs sont utilisés dans la génération de la clé (afin de prouver que le schéma de partage de secret de Shamir [170] est sûr dans l'anneau $\mathbb{Z}_M$ [43]), et dans la preuve de validité. Expliquons ici le second problème qui est moins évident.

Où est le problème ?

La propriété de robustesse garantit que même si t joueurs malicieux envoient de fausses parts de signature, le schéma générera quand même une signature correcte s. Cette propriété est nécessaire car sinon le combineur doit résoudre le problème de la sélection des parts correctes [44].

Par exemple, le combineur reçoit les parts de signature des serveurs et doit générer la signature. Un moyen pour lui consiste à choisir au hasard $t + 1$ parts de signature, à générer une signature possible s' et à tester si s' est une signature valide de m. Si le résultat est correct, la signature a été trouvée, sinon, le combineur doit tester un autre groupe de $t + 1$ parts de signature. Comme le combineur ne peut pas deviner où sont les mauvaises parts, il doit faire face à une explosion combinatoire exponentielle d'essais C_n^{t+1}. Par conséquent, il est nécessaire de construire un test efficace afin de vérifier si un serveur a correctement répondu à une question. Comme on l'a vu dans la sous-section 3.2.4, Shoup a proposé une preuve efficace non-interactive.

Nos résultats

Nous allons montrer que le protocole de Shoup peut être modifié pour fonctionner sous des hypothèses standards avec des modules RSA ayant des propriétés spéciales et que ces modules peuvent être générés de manière distribuée.

Les modules sûrs sont nécessaires dans la preuve de validité et dans le protocole de génération des clés. De plus, différentes caractéristiques de ces nombres sont utilisées dans la preuve de validité. En effet, le protocole de Shoup utilise deux propriétés importantes du sous-groupe des carrés Q_N de $\mathbb{Z}_N^*$ quand un N est module sûr. D'une part, ce sous-groupe est cyclique et d'autre part, son ordre M n'a pas de petits facteurs premiers ce qui permet de simuler le partage *à la Shamir*. Le groupe cyclique est utilisé pour montrer l'**existence du logarithme discret** dans la preuve de robustesse. L'utilisation des nombres premiers sûrs permet de garantir qu'avec probabilité écrasante, un élément pris au hasard dans Q_N est un **générateur**.

Notre première observation concerne le lien entre la structure de Q_N liée au $\mathrm{pgcd}(p-1, q-1)$ et la recherche de générateurs dans Q_N liée à la décomposition en facteurs premiers de $\frac{p-1}{2}$ et de $\frac{q-1}{2}$.

43. L'article d'origine de Shamir prouvait la sécurité dans un corps fini $\mathbb{Z}_p$.

44. On précise que l'algorithme de Berlekamp-Welch (cf. chapitre 2) ne peut pas être utilisé ici car on aurait besoin de résoudre un système d'équations linéaires dans les exposants et que par exemple dans $\mathbb{Z}_p^*$, cela reviendrait à résoudre le problème de Diffie-Hellman ou du logarithme discret.

En particulier, si $\frac{p-1}{2}$ et $\frac{q-1}{2}$ n'ont pas de petits facteurs premiers, alors avec forte probabilité un petit nombre d'éléments pris au hasard génère le groupe Q_N en entier. Si nous choisissons plusieurs éléments au hasard $(g_1, \ldots, g_k)$, on peut garantir que le groupe généré par $\langle g_1, \ldots, g_k \rangle$ est Q_N en entier avec forte probabilité. De telles techniques ont déjà été utilisées par Frankel *et al.* dans [80] et un traitement plus précis a été effectué par Poupard et Stern dans [157]. De plus, en utilisant une astuce de Gennaro *et al.* qui est apparue en premier dans [91] et le protocole récemment proposé par Catalano *et al.* dans [40], le calcul de $\mathrm{pgcd}(p-1, q-1)$ peut être effectué de manière distribuée. Ces méthodes permettent de conserver un algorithme de génération de clé efficace.

Dans cette section, nous montrons comment construire de manière conjointe des modules RSA de telle façon que le sous-groupe Q_N soit cyclique, ce qui garantit l'existence des logarithmes discrets[45] et des générateurs de Q_N. De plus, l'ordre M de ce groupe ne contient pas de petits facteurs premiers plus petits qu'une borne de crible B. Cette vérification n'augmente pas trop le temps d'exécution de l'algorithme de génération de clé.

3.4.2 Modèle de sécurité

Le réseau

Nous faisons comme hypothèse que le groupe de n serveurs est connecté à un canal ayant des capacités de broadcast et que les messages envoyés sur ce canal de communication atteignent instantanément chaque partie reliée à ce dernier.

Définition formelle

Un schéma de signature à seuil RSA est composé des quatre algorithmes suivants :

- Un *algorithme de génération de clés* prend comme entrée deux paramètres de sécurité $L(N), k$, le nombre n de serveurs de signature, le paramètre du seuil t et un ruban aléatoire ω; il retourne une clé publique (N, e) où $L(N)$ est la taille en bits de N, les clés privées $\mathsf{sk}_1 = d_1, \ldots \mathsf{sk}_n = d_n$ connues uniquement par les bons serveurs et pour chaque $u \in [1, k]$ une liste $\mathsf{vk}_u, \mathsf{vk}_{u,1} = {\mathsf{vk}_u}^{d_1}, \ldots \mathsf{vk}_{u,n} = {\mathsf{vk}_u}^{d_n} \bmod N$ de clés de vérification.

- Un *algorithme de part de signature* prend en entrée la clé publique (N, e), un index $1 \leq i \leq n$, la clé privée d_i et un message m; il retourne une part de signature $\sigma_i = x^{2d_i} \bmod N$, où $x = H(m)$ et $H(.)$ est une fonction de hachage et de formatage, et une preuve de sa validité preuve_i.

- Un *algorithme de combinaison* prend en entrée la clé publique (N, e), une liste $\sigma_1, \ldots \sigma_{t'}$ $(t' > t)$ de parts de signature, pour chaque $u \in [1, k]$ la liste $\mathsf{vk}_u, \mathsf{vk}_{u,1}, \ldots \mathsf{vk}_{u,t'}$ de clés de vérification, un message m, et une liste $\mathsf{preuve}_1, \ldots \mathsf{preuve}_{t'}$ de preuves de validité; il retourne une signature s ou échoue.

- Un *algorithme de vérification* prend en entrée une clé publique (N, e), un message m, une signature s; il retourne un bit b indiquant si la signature est correcte ou non.

Les joueurs et le scénario

Le protocole comprend les joueurs suivants : un combineur, un ensemble de n serveurs P_i, un adversaire et les utilisateurs. Tous sont considérés comme des machines de Turing probabilistes et fonctionnant

45. Cependant, même si Q_N est cyclique, on ne sait pas pour autant trouver un générateur

en temps polynomial. Considérons le scénario suivant :

– Durant la phase d'initialisation, les serveurs utilisent l'algorithme de génération distribuée pour créer les clés publique, privées et de vérification. La clé publique (N, e) et toutes les clés de vérification vk_u, $\mathsf{vk}_{u,i}$ sont publiées et chaque serveur obtient sa part d_i de la clé secrète d.

– Pour signer un message m, le combineur commence par envoyer m aux serveurs. En utilisant leur clé secrète d_i et ses clés de vérification $\mathsf{vk}_u, \mathsf{vk}_{u,i}$ pour $u \in [1, k]$, chaque serveur exécute l'algorithme de part de signature et retourne une part de signature σ_i en même temps qu'une preuve de validité de la part de signature preuve_i. Finalement, le combineur utilise l'algorithme de combinaison pour générer la signature, si suffisamment de parts de signature sont valides et disponibles.

Les adversaires

Nous considérons un adversaire capable de corrompre jusqu'à t parmi les n serveurs. Une telle corruption peut être passive, *i.e.* l'attaquant peut seulement intercepter les messages des serveurs et lire la mémoire d'au plus t serveurs. Il peut aussi tenter de faire échouer les serveurs ou de les arrêter. Finalement, il peut être actif; dans ce cas, l'adversaire contrôle complètement le comportement des serveurs corrompus. Dans la suite, nous considérons uniquement des adversaires non-adaptatifs qui choisissent les serveurs qu'ils souhaitent corrompre avant la phase de génération des clés.

Propriétés des schémas de signature à seuil

Les deux propriétés des schémas de signature à seuil t parmi n avec $t < n/2$ qui nous intéressent sont la *robustesse* et la *non-forgeabilité*. La robustesse garantit que même si t serveurs corrompus envoient des mauvaises parts de signature, le schéma retournera toujours une signature correcte.

La non-forgeabilité garantit que tout sous-ensemble de $t + 1$ joueurs peut générer une signature s, mais interdit la génération par tout ensemble de moins de t joueurs.

Les jeux de l'adversaire

Dans cette sous-section, nous décrivons les notions de sécurité pour l'algorithme de génération de clés à seuil et le protocole de signature à seuil sous forme de jeux que tente de gagner l'adversaire. Nous devons montrer que les informations révélées pendant ces protocoles n'aident pas l'adversaire.

Le jeu pour la version distribuée de la génération de clé. La validité de la génération de clé exige que les distributions de probabilité des clés secrètes d, p, q, et de la clé publique (N, e) paraissent uniformes à l'adversaire.
La sécurité de la génération des clés exige que s'il existe un adversaire $\mathcal{A}$ qui corrompt au plus t serveurs au début du jeu, alors il ne peut pas obtenir plus d'information sur les clés secrètes détenues par les joueurs non corrompus.

Le jeu pour le protocole de signature à seuil. La sécurité du protocole de signature signifie que s'il existe un adversaire $\mathcal{A}$ qui corrompt au plus t serveurs au début du protocole et qui peut obtenir des signatures de messages de son choix, alors il ne peut pas forger une signature sur un nouveau message.

Remarques. On peut remarquer que nous ne montrons pas que l'information apprise durant le protocole de génération de clé n'aide pas l'adversaire à déchiffrer ou à signer le message. Nous faisons ici une hypothèse plus forte sur un adversaire qui peut factoriser le module N comme le font Boneh et Franklin dans [27].

3.4.3 Nouvel algorithme de génération de clé RSA

Le but de cette section est de générer des modules RSA tels que le groupe des carrés est un groupe cyclique dont l'ordre ne contient pas de petits facteurs premiers et des clés pour le schéma de signature de Shoup.

Pour ce faire, nous revisitons l'algorithme de génération partagée de clés RSA de Boneh-Franklin pour produire des modules RSA tels que $N = pq$, $\mathrm{pgcd}(\frac{p-1}{2}, \frac{q-1}{2}) = 1$ et ni $\frac{p-1}{2}$ ni $\frac{q-1}{2}$ n'ont de petits facteurs premiers. Nous utilisons un algorithme de crible pour améliorer simultanément le protocole de génération de clé, la probabilité de trouver un ensemble de générateurs de Q_N, et pour rendre sûr le schéma de partage de secret de Shamir. Nous prouvons enfin la robustesse et la sécurité de ce nouveau protocole distribué de génération de clés.

Génération distribuée de modules RSA de forme spéciale

Nous décrivons dans la figure 3.3 une adaptation du protocole de Boneh et Franklin pour générer un module RSA partagé, puis nous montrons comment générer les clés de vérification dont nous avons besoin.

1. Dans la première étape, chaque serveur choisit au hasard deux valeurs p_i et q_i dans l'intervalle $\left[\sqrt{2}.2^{L(N)/2-1}, \lfloor\frac{2^{L(N)/2}-1}{n}\rfloor\right[$ selon [176], où $L(N)$ est la taille en bits du module N.
2. Les serveurs effectuent un crible sur les entiers partagés $(p_1 + \ldots + p_n) - 1$ et $(q_1 + \ldots + q_n) - 1$. Puis, ils vérifient si $\mathrm{pgcd}(4P, p-1) \stackrel{?}{=} 2$ et si $\mathrm{pgcd}(4P, q-1) \stackrel{?}{=} 2$ où $P = \prod_{2<p_i<B} p_i$ et B la borne du crible en utilisant l'algorithme GCD.
3. Puis le protocole BGW, décrit dans le chapitre 9 est exécuté pour calculer le produit N de $p_1+\ldots+p_n$ et $q_1+\ldots+q_n$ et pour calculer le produit $\varphi(N)$ de $(p_1+\ldots+p_n)-1$ et de $(q_1 + \ldots + q_n) - 1$. Les serveurs vérifient ensuite de manière distribuée si $\mathrm{pgcd}(\varphi(N), N-1) \stackrel{?}{=} 1$ en utilisant l'algorithme GCD.
4. Ensuite, les parties exécutent un test de biprimalité similaire au test de Fermat modulo N. Nous fixons $p = p_1 + \ldots + p_n$ et $q = q_1 + \ldots + q_n$.

FIG. 3.3: *Algorithme de génération distribuée de modules RSA de la forme spéciale*

Génération distribuée des clés dans le protocole de Shoup

Une fois que le modulo N est généré, soit e le premier nombre premier supérieur à n. On exécute ensuite le protocole de Catalano *et al.* décrit dans le chapitre 9 pour générer la clé secrète partagée d de manière partagée. À la fin du protocole, chaque serveur peut calculer ses clés de vérification comme $v_{u,i} = v_u^{\Delta d_i}$. Les clés de vérification v_u sont des carrés aléatoires et on suppose qu'avec k tels nombres on génère Q_N. On peut donc les calculer de la façon suivante : $y_u^2 \bmod N$ où y_u est la concaténation

de $H(N\|i)$ pour suffisamment de valeurs de i pour obtenir un paramètre de sécurité suffisant. La valeur aléatoire des valeurs y_u est obtenue dans le modèle de l'oracle aléatoire.

Ce protocole est robuste face à des adversaires actifs car le protocole GCD l'est. Nous étudierons ultérieurement la sécurité de ce protocole.

Algorithme efficace de crible pour améliorer la génération de nombres aléatoires n'ayant pas de petits facteurs

Dans cette section, nous montrons que le module N généré avec l'algorithme précédent est tel que $p' = \frac{p-1}{2}$ et $q' = \frac{q-1}{2}$ n'ont pas de petits facteurs premiers. Notre méthode utilise un nouveau protocole distribué de crible conçu par Boneh, Malkin et Wu dans [28] que nous modifions afin de créer p tel que ni p, ni p' n'ait dans sa décomposition en facteurs premiers de petits facteurs inférieurs à B. De plus, nous montrons comment résister aux adversaires actifs. Notons P le produit de tous les nombres premiers impairs jusqu'à B.

1. Chaque serveur choisit au hasard un entier a_i dans l'intervalle $[1, \ldots, P]$ de telle façon que a_i soit relativement premier avec P. Alors, le produit $a = a_1 \times \ldots \times a_n \bmod P$ est aussi relativement premier avec P.
2. Les serveurs exécutent un protocole pour convertir le partage multiplicatif de a en un partage additif de $a = b_1 + \ldots + b_n$. Ce protocole est décrit dans la suite et utilise le protocole de BGW, décrit en annexe 9.
3. Chaque serveur choisit un aléa $r_i \in [0, \frac{2^{L(N)}}{P}]$ et fixe $p_i = r_i P + b_i$.

FIG. 3.4: *Algorithme de crible distribué*

Clairement, $p = \sum p_i \equiv a \bmod P$ (où p_i est dans la figure 3.4) et ainsi p n'est divisible par aucun nombre premier inférieur à B. On peut remarquer que $p = RP + a$ où $R = \sum_i r_i$. Ce crible ne fonctionne que jusqu'à une borne B telle que $p > P = \prod_{3 \leq p < B} p$, soit 379 pour p de 512 bits. Ce crible fonctionne conformément au théorème suivant: si $\mathrm{pgcd}(a, P) = 1$, alors si $p = rP + a$, alors $\mathrm{pgcd}(p, P) = \mathrm{pgcd}(rP+a, P) = \mathrm{pgcd}(a, P) = 1$. Ainsi, si on veut étendre la taille du crible à $B_1 > B$, alors peut générer a tel que $\mathrm{pgcd}(a, P) = 1$ et ensuite tester si $\mathrm{pgcd}(P', p) \stackrel{?}{=} 1$ avec $P' = \prod_{B \leq p \leq B_1} p$ avec l'algorithme GCD (cf. annexe 9).

Preuve de validité et de robustesse de l'étape 2 de l'algorithme de génération du module.
Preuve:

Preuve de validité. Afin de prouver que $p' = \frac{p-1}{2}$ n'a pas de facteur premier inférieur à B dans sa décomposition, nous devons vérifier les égalités $\mathrm{pgcd}(p-1, P) = \mathrm{pgcd}(2p', P) = 1$ et $\mathrm{pgcd}(p - 1, 4P) = 2$ (pour tester la puissance de 2). Si nous notons $P' = 4P$, nous pouvons exécuter un unique test simple $\mathrm{pgcd}(p-1, P') = 2$.

Preuve de sécurité de l'étape 2 contre des adversaires passifs. Pour distribuer le test de l'étape 2 dans le modèle *honnête-mais-curieux*, le premier serveur modifie sa part p_1 en $p_1 - 1$ et nous utilisons le protocole GCD distribué décrit dans l'annexe 9 qui est sûr contre ce type d'adversaire. De plus, le crible est sûr contre ce type d'adversaire.

Preuve de robustesse de l'étape 2. Il est aussi possible d'effectuer le test de manière robuste en présence d'un adversaire actif. Pour faire face à des adversaires actifs, nous utilisons l'algorithme de Frankel, MacKenzie et Yung [80] qui génère directement un partage polynomial de p et q. Pour ce faire, chaque serveur partage p_i et q_i en utilisant un partage *à la Shamir*. Remarquons que $\sum_{j\in S\setminus\{0\}} \lambda_{0,j}^S = 1$, où $\lambda_{0,j}^S$ dénote le coefficient de Lagrange du j-ième serveur[46]. Par conséquent, si on note f, le polynôme de partage de l'entier p, la part de p du i-ième serveur est $f(i)$. La part de l'entier $p-1$ du serveur i peut alors être fixée à $(f(i)-1)$. En effet, si $p = f(0) = \sum_{j\in S\setminus\{0\}} \lambda_{0,j}^S f(j)$, alors

$$p - 1 = f(0) - 1 = \sum_{j\in S\setminus\{0\}} \lambda_{0,j}^S f(j) - \sum_{j\in S\setminus\{0\}} \lambda_{0,j}^S = \sum_{j\in S\setminus\{0\}} \lambda_{0,j}^S (f(j)-1)$$

Ensuite, le protocole GCD peut s'appliquer avec un partage polynomial de la valeur secrète $p-1$.

Finalement, nous pouvons aussi montrer que le crible peut être effectué de manière robuste. En effet, le protocole qui transforme le partage multiplicatif de $a = \prod_{i=1}^n a_i$ en un partage additif $b = \sum_{i=1}^n b_i$ peut aussi être effectué de manière robuste car il utilise le protocole BGW. Cette transformation appelle n fois le protocole BGW. La valeur $b_{i,j}$ est la part détenue par le i-ième serveur au j-ième tour, et est telle que $\sum_{i=1}^n b_{i,j} = \prod_{k=1}^j a_k$. On fixe $b_{i,0} = 0$ pour tout $i \in \{0,\ldots,n\}$. Alors, pour $i = 1$ à n, $u_i = a_i$ et $u_j = 0$ pour tout $j \neq i$, le protocole BGW calcule le produit partagé de ab à partir des partages de a et de b : $(b_{1,i-1} + \ldots + b_{n,i-1}) \times a_i = (b_{1,i-1} + \ldots + b_{n,i-1})(u_1 + \ldots + u_n) = b_{1,i} + \ldots + b_{n,i}$.

Si, on ne veut pas utiliser l'algorithme de partage multiplicatif en partage additif, on peut utiliser l'algorithme GCD pour tester si $\mathrm{pgcd}(P,p) \stackrel{?}{=} 1$ ou $\mathrm{pgcd}(PP',p) \stackrel{?}{=} 1$ directement. Le test sur $p-1$ se déduit aisément $\mathrm{pgcd}(4PP', p-1) \stackrel{?}{=} 2$. □

Ainsi, nous avons vu que la deuxième phase de l'algorithme de génération de clés, qui élimine les mauvais choix des p_i et q_i, peut être rendu robuste pour générer des modules RSA tels que ni $\frac{p-1}{2}$, ni $\frac{q-1}{2}$ n'aient de petits facteurs premiers inférieurs à B, et $\mathrm{pgcd}(\frac{p-1}{2}, \frac{q-1}{2}) = 1$. De plus, comme la suite du protocole suit l'algorithme de génération de clés de Boneh-Franklin et que ce protocole a été rendu robuste par Frankel, Mac Kenzie et Yung [80], on peut conclure que le nouveau protocole de génération de modules RSA est sûr et peut être rendu robuste contre des adversaires actifs en utilisant le protocole de Frankel *et al.*.

Théorème 10. Le protocole de génération de clés de Boneh-Franklin et le protocole de crible permettent de générer des modules RSA tels que l'ordre M du groupe Q_N ne contient pas de petits facteurs premiers inférieurs à B.

Remarque. Le crible a aussi comme effet de rendre l'algorithme de génération plus efficace. En effet, il évite le test de biprimalité distribué, ce qui permet d'accroître la vitesse de l'algorithme de Boneh-Franklin d'un facteur 10 comme le prouvent les tests réalisés dans [28]. Il faut environ une minute et demie à ce protocole pour générer de manière distribuée une clé RSA 1024 bits dans le modèle *honnête-mais-curieux*.

46. Ceci provient du fait que si on partage, avec le schéma de partage de secret de Shamir, le polynôme constant égal à 1, la valeur de ce polynôme en 0 est aussi 1.

Génération de clés RSA telles que Q_N soit cyclique

Nous montrons que le module N généré par l'algorithme de génération est tel que le groupe Q_N est cyclique. Ceci est une conséquence de l'étape 3 de l'algorithme décrit dans la figure 3.3. Nous allons utiliser le fait que le produit de deux groupes cycliques d'ordre premier entre eux est un groupe cyclique. Le lemme suivant et le protocole GCD permettent de vérifier de manière distribuée que p' et q' sont premiers entre eux. Montrons d'abord le lemme suivant, utilisé sous une autre forme dans [91].

Lemme 3 *Soit $N = pq$ un module RSA, alors $\mathrm{pgcd}(p-1, q-1)|\mathrm{pgcd}(N-1, \varphi(N))$ et la part sans carré de $\mathrm{pgcd}(N-1, \varphi(N))$ divise $\mathrm{pgcd}(p-1, q-1)$.*

Preuve: Nous pouvons remarquer que $\varphi(N) = (p-1)(q-1) = N-p-q+1 = (N-1)-(p-1)-(q-1)$. Donc,

$$(N-1) - \varphi(N) = (p-1) + (q-1)$$

Par conséquent, $\mathrm{pgcd}(N-1, \varphi(N)) = \mathrm{pgcd}((N-1)-\varphi(N), \varphi(N)) = \mathrm{pgcd}((p-1)+(q-1), \varphi(N))$. Si nous notons $a = p-1$ et $b = q-1$, nous devons montrer que $\mathrm{pgcd}(a, b)|\mathrm{pgcd}(a+b, ab)$. Si $f|\mathrm{pgcd}(a, b)$, alors $f|a$ et $f|b$. Donc $f|(a+b)$ et $f|ab$. D'où $f|\mathrm{pgcd}(a+b, ab)$.

Montrons maintenant que la part sans carré de $\mathrm{pgcd}(a+b, ab)$ divise $\mathrm{pgcd}(a, b)$.

$$\mathrm{pgcd}(a+b, ab) = \mathrm{pgcd}(a+b, ab - a(a+b)) = \mathrm{pgcd}(a+b, -a^2) = \mathrm{pgcd}(a+b, a^2)$$

Supposons que $f|(a+b)$ et $f|a^2$. Si f est un nombre premier, $f|a$ et comme $f|(a+b)$, $f|\mathrm{pgcd}(a, b)$. Si f n'est pas un nombre premier mais une puissance d'un nombre permier, disons $f = f'^{\alpha}$, nous avons $f'^{\alpha}|a^2$ et $\alpha = 2\beta$. Ainsi, $f'^{\beta}|(a+b)$ et $f'^{\beta}|a$, donc $f'^{\beta}|\mathrm{pgcd}(a, b)$. □

Corollaire 1 *Si $\mathrm{pgcd}(N-1, \varphi(N)) = 2$, alors $\mathrm{pgcd}(p-1, q-1) = 2$.*

Preuve: Si $\mathrm{pgcd}(N-1, \varphi(N)) = 2$, alors comme $\mathrm{pgcd}(p-1, q-1)|\mathrm{pgcd}(N-1, \varphi(N))$, $\mathrm{pgcd}(p-1, q-1) = 2$ car $\mathrm{pgcd}(p-1, q-1)$ ne peut pas valoir 1. □

La vérification $\mathrm{pgcd}(N-1, \varphi(N)) \stackrel{?}{=} 2$ peut être faite en utilisant le protocole GCD.

Théorème 11. Le protocole de génération de clés permet de générer des modules RSA tels que le groupe Q_N soit cyclique d'ordre $M = p'q'$, où $N = pq$, $p = 2p'+1$, $q = 2q'+1$ et ni p' ni q' n'ont de petits facteurs premiers inférieurs à B. Le nombre de fois qu'il faut itérer ce protocole par rapport à celui de Boneh-Franklin est en moyenne de $4 \times e^{\gamma} \ln(B)$.

Preuve: En suivant la section 3.4.3 et le corrolaire 1, nous pouvons supposer que l'on a obtenu un module RSA tel que tous les facteurs premiers de $p-1$ et de $q-1$ sont supérieurs à B, et qu'ils n'ont pas de facteurs communs, *i.e.*, $\mathrm{pgcd}(p-1, q-1) = 2$ et $\frac{p-1}{2}$ et $\frac{q-1}{2}$ n'ont pas de petits facteurs premiers. Comme le produit de groupes cycliques dont les ordres sont premiers entre eux est encore un groupe cyclique, et comme le groupe des carrés de $\mathbb{Z}_p^*$ et le groupe des carrés de $\mathbb{Z}_q^*$ sont cycliques, le groupe Q_N est aussi cyclique. Ceci permet de garantir que Q_N est cyclique d'ordre $M = p'q'$.

Nous pouvons estimer le nombre d'itérations de l'algorithme 3.3 par rapport au protocole de Boneh-Franklin. Tout d'abord, il est bien connu que

$$\lim_{n\to\infty} \Pr_{(p',q')\in[1,n[^2} \left[\mathrm{pgcd}(p', q') = 1\right] = \frac{6}{\pi^2} > 1/2$$

si l'on suppose que les nombres premiers sont uniformément distribués dans $[2^{\frac{L(N)-1}{2}}, 2^{\frac{L(N)}{2}}[$. De plus, le seul facteur augmentant le temps d'exécution de la première phase du protocole de génération de clé correspond à la phase de vérification $\mathrm{pgcd}(P', p-1) \stackrel{?}{=} 2$, où $P' = 4P$. On peut remarquer que :

$$\begin{aligned}
\Pr_{p'}\left[\mathrm{pgcd}(2p', P') = 2\right] &= \Pr_{p'}\left[2 \nmid p' \wedge 3 \nmid p' \wedge \ldots \wedge B \nmid p'\right] \\
&= (1-\frac{1}{2})(1-\frac{1}{3})\ldots(1-\frac{1}{B}) \\
&= \prod_{p_i \leq B}(1-\frac{1}{p_i}) \approx \frac{1}{e^{\gamma}\ln(B)}
\end{aligned}$$

selon le deuxième théorème de Mertens, où γ représente la constante d'Euler. Par conséquent, nous devons exécuter cet algorithme $2 \times (e^{\gamma}\ln(B) + e^{\gamma}\ln(B))$ en moyenne afin d'obtenir un module RSA de forme spéciale. □

Ainsi, l'étape 3 est sûre contre des adversaires passifs ou actifs car elle utilise le protocole BGW et le protocole GCD qui peuvent être rendu robuste.

Preuves de securité de l'algorithme de génération de clé

Dans cette partie, nous prouvons la sécurité complète de l'algorithme de génération du module RSA de forme spéciale et la sécurité de l'algorithme de génération de clés.

Théorème 12. La génération distribuée de la clé est sûre et robuste contre un adversaire statique et actif qui contrôle jusqu'à t serveurs.

Preuve: La robustesse de chaque étape des algorithmes de génération distribuée de modules RSA de forme spéciale et de génération de clés a été montrée dans les sections précédentes. Montrons maintenant la sécurité de l'algorithme complet de génération du module et de l'algorithme de génération de clés.

Dans la preuve de sécurité, nous devons montrer que s'il existe un adversaire $\mathcal{A}$ qui corrompt au plus t serveurs au début du jeu, alors nous pouvons l'utiliser pour construire un attaquant contre la version centralisée du protocole de génération de clés afin de factoriser le modulo N. Cet attaquant est un simulateur dont le rôle est de simuler de "fausses" informations à l'adversaire : il lui fournir des données similaires à celles qu'il recevrait dans un jeu normal (mais le simulateur ne possède pas les données privées d'un vrai jeu) de façon à ce que l'adversaire ne puisse pas distinguer si ces "fausses" informations proviennent d'un jeu normal ou d'un jeu simulé. Par conséquent, l'attaquant sera quelquefois appelé simulateur $\mathcal{S}$.

Considérons le jeu A suivant :

A1 L'adversaire choisit de corrompre t serveurs. Il apprend toutes leurs informations secrètes et contrôle activement leur comportement.

A2 L'algorithme de génération des clés est exécuté; les clés publiques et privées et les clés de vérification sont calculées.

A3 L'attaquant essaie soit de factoriser le module RSA à partir des informations qu'il a apprises durant le processus de génération de clés soit d'obtenir des informations sur des parties de la clé secrète détenues par des joueurs non corrompus.

Alors que le schéma de Boneh et Franklin exige de simuler seulement le *module public* N durant la génération de la clé, notre schéma nécessite aussi de simuler les *clés de vérification* qui font aussi partie de la clé publique. Nous commençons par prouver la **sécurité du module RSA**.

Sécurité de l'algorithme de génération du module.
D'après le théorème 9 de Boneh et Franklin, la génération du module RSA est sûre contre des adversaires passifs. Nous affirmons que la modification que nous avons introduite dans le protocole de génération des clés ne modifie pas le résultat de Boneh et Franklin sur N. En effet, nous ajoutons quelques restrictions sur le choix de p et q en imposant que ces valeurs vérifient $\mathrm{pgcd}(p-1, q-1) = 2$ et $\frac{p-1}{2}$ et $\frac{q-1}{2}$ ne contiennent pas de petits facteurs premiers. Nous avons vu au théorème 11 que nous affaiblissons la probabilité de réussite d'un facteur $1/4(e^{\gamma}\ln(B))$. Ainsi, le résultat est toujours valide en remplaçant $1/4(t+1)^3 L(N)^{\delta}$ par $1/16(e^{\gamma}\ln(B))(t+1)^3 L(N)^{\delta}$.

Sécurité de l'algorithme de génération de clés.
La sécurité de l'algorithme de génération de la clé secrète est une conséquence de la sécurité de l'algorithme GCD prouvée dans [40]. Maintenant, nous devons prouver que les informations révélées par **les clés de vérification** n'aident pas l'adversaire.

A partir des informations connues par l'adversaire, en particulier les t parts de la clé privée obtenues à partir des parts possédées par les serveurs corrompus et des clés de vérification $\mathsf{vk}_u \in Q_N$ pour $u \in [1, k]$, nous voulons montrer que les clés de vérification de chaque serveur peuvent être calculées par l'adversaire. Ainsi, cette information révélée ne lui donnera pas d'informations sur les parts de la clé privée détenues par les serveurs non corrompus.

Sans perte de généralité, nous pouvons faire comme hypothèse que $\mathcal{A}$ corrompt les t premiers serveurs. Donc, l'attaquant choisit d'apprendre les clés secrètes $d_1, \ldots, d_t$ des joueurs corrompus dans la phase A1 du jeu A; d_i doit être dans l'intervalle $\{0, \ldots, M\}$, mais comme M est inconnu, $\mathcal{S}$ choisit d_i dans $\{0, \ldots, \lfloor N/4 \rfloor\}$. Cependant, comme la distance statistique entre la distribution uniforme sur $\{0, \ldots \lfloor N/4 \rfloor - 1\}$ et la distribution uniforme sur $\{0, \ldots M-1\}$ est $O(N^{-1/2})$, l'adversaire $\mathcal{A}$ ne peut pas distinguer entre les clés réelles et les clés simulées.

Dans la simulation, $\mathcal{S}$ choisit les vk_u en tirant k éléments aléatoires w_u dans $\mathbb{Z}_N^*$ tels que $\mathsf{vk}_u = {w_u}^{2e} \bmod N$. Donc nous savons que ${\mathsf{vk}_u}^d = {w_u}^2 \bmod N$ et les vk_u sont dans Q_N. Les clés de vérification des serveurs corrompus sont calculées en utilisant les clés secrètes connues d_i et les $\mathsf{vk}_{u,t+1}, \ldots, \mathsf{vk}_{u,n}$ manquants sont obtenus avec la formule d'interpolation de Lagrange. Bien sûr, nous ne sommes pas capables de trouver les clés secrètes manquantes mais en fait nous n'en avons pas besoin.

En effet, nous notons S, l'ensemble constitué de l'entier 0 et des index des joueurs corrompus. Pour chaque clé de vérification vk_u, les parts des serveurs corrompus i sont $\mathsf{vk}_{u,i} = {\mathsf{vk}_u}^{\Delta d_i} \bmod N$ où $\Delta = n!$. L'adversaire $\mathcal{A}$ peut calculer ces valeurs car il connaît d_i pour l'ensemble des serveurs corrompus. Pour les autres serveurs, le simulateur utilise l'interpolation polynomiale.

$$\mathsf{vk}_{u,i} = {w_u}^{2\lambda_{i,0}^S} \times \prod_{j \in S \setminus \{0\}} {\mathsf{vk}_u}^{d_j \lambda_{i,j}^S} = {\mathsf{vk}_u}^{d\lambda_{i,0}^S + \sum_{j \in S\setminus\{0\}} d_j \lambda_{i,j}^S} \bmod N$$

□

3.4.4 Sécurité du schéma de signature contre un adversaire passif

Dans cette sous-section, nous montrons que le protocole de Shoup utilisé avec des modules RSA générés comme dans la section précédente est sûr contre un adversaire statique et passif.

Information révélée par les parts de la clé

A la fin du protocole de génération des clés, nous voulons savoir si les informations qu'un adversaire peut collecter peuvent l'aider à obtenir des informations utiles sur le secret d. Soient $d_{i_1}, \ldots, d_{i_t}$, t parts de la clé secrète obtenues par l'adversaire à partir des t serveurs corrompus.

Information révélée par les parts d_{i_j}.
Pour chaque d_{i_j}, remarquons que

$$d_{i_j} = f(i_j) = a_0 + a_1 i_j + \ldots + a_t i_j^t \bmod M$$

Si on ajoute une $(t+1)$-ième équation, $a_0 = d$, on obtient le système linéaire suivant :

$$\begin{cases} a_0 + a_1 i_1 + \ldots + a_t {i_1}^t = d_{i_1} \bmod M \\ a_0 + a_1 i_2 + \ldots + a_t {i_2}^t = d_{i_2} \bmod M \\ \qquad\qquad \ldots \\ a_0 + a_1 i_t + \ldots + a_t {i_t}^t = d_{i_t} \bmod M \\ a_0 + 0 + \ldots + 0 = d \end{cases}$$

ou, sous forme matricielle, $I.A = D \bmod M$

$$\begin{pmatrix} 1 & i_1 & i_1^2 & \ldots & i_1^t \\ 1 & i_2 & i_2^2 & \ldots & i_2^t \\ \ldots & & \ddots & & \ldots \\ 1 & i_t & i_t^2 & \ldots & i_t^t \\ 1 & 0 & 0 & \ldots & 0 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_t \end{pmatrix} = \begin{pmatrix} d_{i_1} \\ d_{i_2} \\ \vdots \\ d_{i_t} \\ d \end{pmatrix} \bmod M$$

La matrice I est une matrice de Vandermonde. Le déterminant d'une telle matrice est $\det(I) = \prod_{1 \le j < k \le t+1} (i_k - i_j) \bmod M$, où $i_{t+1} = 0$. Les i_j's sont distincts dans $\mathbb{Z}_M$, $i_k - i_j \neq 0 \bmod M$, car $n < B$. Ainsi, chaque facteur $(i_k - i_j)$ est inversible modulo M car donc le produit $\det(I)$ est lui aussi inversible dans $\mathbb{Z}_M$. Par conséquent, toutes les valeurs de d sont possibles. Ainsi un groupe de t serveurs ne peut pas obtenir d'information sur d à partir des parts de d. On voit ici que le crible exécuté est important pour éviter la perte d'information sur d. En conséquence, on doit avoir $n < B$.

Le protocole de signature

Pour générer une signature sur un message m, chaque serveur i calcule $x = H(m)$, $\sigma_i = x^{2\Delta d_i} \bmod N$ et envoie σ_i au combineur sans aucune preuve car nous sommes dans le modèle honnête-mais-curieux. Le combineur choisit un ensemble S de $t+1$ valeurs et calcule $w = \prod_{j=1}^{t+1} s_{i_j}^{2\lambda_{0,i_j}^S} \bmod N$. Il s'ensuit que $w^e = x^{4\Delta^2}$ car $s_{i_j}^2 = x^{4\Delta d_{i_j}}$. En appliquant un lemme bien connu [102], on peut extraire une racine e-ième de x à partir de w car $4\Delta^2$ est une valeur connue (en utilisant l'algorithme d'Euclide étendu sur e et $4\Delta^2$). Comme nous sommes dans le modèle *honnête-mais-curieux*, la signature s est toujours correcte.

On peut prouver le théorème suivant :

Théorème 13. Dans le modèle de l'oracle aléatoire, le protocole de signature décrit ci-dessus est un schéma de signature à seuil sûr (non-forgeable existentiellement) sous l'hypothèse que la signature RSA (RSA-FDH ou RSA-PSS) est sûre.

Preuve: Similaire à la section 3.2.3. □

3.4.5 Sécurité du schéma de signature contre un adversaire actif

Le but de cette section est de revisiter la preuve de robustesse d'origine conçue par Shoup pour couvrir le cas des modules RSA générés dans la section 3.4.3. La méthode utilisée permet de générer avec forte probabilité le groupe des carrés en prenant peu d'éléments au hasard.

Preuve de validité des parts de signature

Soit N un module tel que $N = pq$ et $p = 2p' + 1$ et $q = 2q' + 1$ où p' et q' n'ont pas de petits facteurs premiers et $\mathrm{pgcd}(p-1, q-1) = 2$. Comme Q_N est cyclique, il existe un générateur g dans Q_N. Alors, le logarithme discret en base g de n'importe quel élément ${\sigma_i}^2$ existe, où $\sigma_i = x^{2\Delta d_i}$, $\Delta = n!$, et $x = H(m)$. Comme dans la section 15, notons $v_1, \ldots, v_k$ un k-uplet d'éléments choisis au hasard dans $(Q_N)^k$ tel qu'avec forte probabilité, ce k-uplet génère le groupe Q_N en entier, d'ordre $M = p'q'$, *i.e.* pour chaque $x \in Q_N$, il existe $(a_1, \ldots, a_k) \in [0, M[^k$ tels que $x = \prod_{i=1}^{k} {v_i}^{a_i} \bmod N$. Posons $\mathsf{vk}_i = v_i$ pour tout $i \in [1, k]$.

Chaque serveur i possède un k-uplet de clés de vérification $\mathsf{vk}_{1,i} = {\mathsf{vk}_1}^{d_i} \bmod N, \ldots, \mathsf{vk}_{k,i} = {\mathsf{vk}_k}^{d_i} \bmod N$. Il calcule une part de signature, $\sigma_i = x^{2d_i\Delta} \bmod N$, où d_i est la i-ième de la clé secrète d et prouve que

$$\log_{\mathsf{vk}_1}(\mathsf{vk}_{1,i}) = \ldots = \log_{\mathsf{vk}_k}(\mathsf{vk}_{k,i}) = \log_{x^{4\Delta}}({\sigma_i}^2)$$

Décrivons maintenant la preuve de "robustesse" et comme d'habitude, soit $d_i \in [0, M[$ les parts secrètes de la clé de chaque serveur, et A et B' deux entiers tels que $\log(A) \geq \log(B'Mh) + k_2$ où B' et k_2 sont des paramètres de sécurité et h est le nombre de tours d'une version interactive de la preuve. On fixe $A = 2^{L(N)+hL'_1+k_2}$ et $B' = 2^{L'_1}$. Finalement, k_1 est un paramètre tel que la probabilité de tricher $1/{B'}^h$ soit $< 1/2^{k_1}$, $(k_1 > hL'_1)$. Alors que le paramètre de sécurité k_1 contrôle la consistance et le caractère statistiquement zero-knowledge, le paramètre de sécurité k_2 contrôle la significativité du protocole. Nous présentons le protocole pour un seul tour.

Le prouveur choisit un aléa r dans $[0, A[$, puis calcule $t = (v'_1, \ldots, v'_k, x') = (\mathsf{vk}_1^r, \ldots, \mathsf{vk}_k^r, x^{4\Delta r})$. Soit e les $b' = \log(B') - 1$ premiers bits de la valeur de la fonction de hachage

$$e = [H(\mathsf{vk}_1, \ldots, \mathsf{vk}_k, x^{4\Delta}, \mathsf{vk}_{1,i}, \ldots, \mathsf{vk}_{k,i}, {\sigma_i}^2, v'_1, \ldots, v'_k, x')]_{b'}$$

si nous notons $[x]_{b'}$ les b' premiers bits de x. Ensuite, le prouveur calcule z où $z = r + ed_i$. La preuve est la paire $(e, z) \in [0, B'[\times[0, A[$. Pour la vérifier, le vérifieur doit calculer si

$$e \stackrel{?}{=} [H(\mathsf{vk}_1, \ldots, \mathsf{vk}_k, x^{4\Delta}, \mathsf{vk}_{1,i}, \ldots, \mathsf{vk}_{k,i}, {\sigma_i}^2, {\mathsf{vk}_1}^z {\mathsf{vk}_{1,i}}^{-e}, \ldots, {\mathsf{vk}_k}^z {\mathsf{vk}_{k,i}}^{-e}, x^{4\Delta z} {\sigma_i}^{-2e})]_{b'} \qquad (3.5)$$

et vérifier si $0 \leq z < A$.

Analyse de sécurité de la preuve de robustesse

Consistance.

Théorème 14. L'exécution du protocole entre un prouveur qui connaît le secret d_i et un vérifieur est réussi avec probabilité écrasante si $B'Mh/A$ est négligeable, où h est le nombre de tours.

Preuve: Si le prouveur connaît $d_i \in [0, M[$ et suit le protocole, il échoue seulement si $z \geq A$. Pour toute valeur $x \in [0, M[$, la probabilité d'échec d'un tel événement pris sur l'ensemble des choix possibles

de r est inférieure à $B'M/A$. Par conséquent l'exécution de la preuve est réussie avec probabilité $\geq (1 - \frac{B'M}{A})^h \geq 1 - \frac{B'Mh}{A}$. □

Significatif. On va montrer que la version interactive du schéma est sûre face à un vérifieur honnête, ce qui prouvera la sécurité de la version non-interactive en utilisant le lemme de bifurcation [151].

Lemme 4 *Si le vérifieur accepte la preuve, avec probabilité $\geq 1/B' + \varepsilon$ où ε est une quantité non-negligeable, alors en utilisant le prouveur comme une "boîte-noire" il est possible de calculer σ et τ tels que $|\sigma| < A$, et $|\tau| < B'$ et $\mathsf{vk}_1{}^{\sigma} = \mathsf{vk}_{1,i}{}^{\tau}, \ldots, \mathsf{vk}_k{}^{\sigma} = \mathsf{vk}_{k,i}{}^{\tau}, x^{4\Delta\sigma} = {\sigma_i}^{2\tau}$.*

Preuve: Si nous redémarrons l'adversaire et que nous obtenons deux preuves valides pour la même mise en gage t, (e, z) et (e', z'), nous avons pour $u = 1, \ldots, k$ *i.e.* pour toutes les clés de vérification, $\mathsf{vk}_u{}^{r} = \mathsf{vk}_u{}^{z}\mathsf{vk}_{u,i}^{-e} = \mathsf{vk}_u{}^{z'}\mathsf{vk}_{u,i}^{-e'}$. Donc, nous obtenons $\mathsf{vk}_u{}^{\sigma} = \mathsf{vk}_{u,i}{}^{\tau} \bmod N$ si nous fixons $\sigma = z' - z$ et $\tau = e - e'$. Par conséquent nous pouvons écrire $\mathsf{vk}_1{}^{\sigma} = \mathsf{vk}_{1,i}{}^{\tau}, \ldots, \mathsf{vk}_k{}^{\sigma} = \mathsf{vk}_{k,i}{}^{\tau}, x^{4\Delta\sigma} = {\sigma_i}^{2\tau}$. □

Théorème 15. (Significatif) Faisons l'hypothèse qu'une machine de Turing probabiliste en temps polynomial $\tilde{P}$ soit acceptée avec probabilité non-negligeable. Si $B' < B$, $h \times \log(B') = \Theta(k_1)$, $k = \Theta(k_1/\log(B))$ et $\log(A)$ est un polynôme en k_1 et $\log(N)$, nous pouvons montrer que $x^{4\Delta d_i} = {\sigma_i}^2$ et donc σ_i est une part de signature correcte.

Preuve: D'après le lemme précédent, nous pouvons faire comme hypothèse que nous avons τ et σ tels que $\mathsf{vk}_u{}^{\sigma} = \mathsf{vk}_u{}^{d_i\tau}$ pour $u = 1, \ldots, k$ et $x^{4\Delta\sigma} = {\sigma_i}^{2\tau}$.

Alors, nous pouvons écrire $x^{4\Delta}$ avec l'ensemble de générateurs de Q_N car c'est un carré : $x^{4\Delta} = \mathsf{vk}_1{}^{\beta_1} \times \ldots \times \mathsf{vk}_k{}^{\beta_k}$.

Par conséquent, si nous élevons cette équation à la puissance σ, nous obtenons $x^{4\Delta\sigma} = \mathsf{vk}_1{}^{\sigma\beta_1} \times \ldots \times \mathsf{vk}_k{}^{\sigma\beta_k}$. Mais, $x^{4\Delta\sigma}$ est égal à ${\sigma_i}^{2\tau}$ et $\mathsf{vk}_1{}^{\sigma\beta_1} \times \ldots \times \mathsf{vk}_k{}^{\sigma\beta_k}$ est égal à $(\mathsf{vk}_1{}^{\beta_1} \times \ldots \times \mathsf{vk}_k{}^{\beta_k})^{\tau d_i}$ car $\mathsf{vk}_u{}^{\sigma} = \mathsf{vk}_u{}^{d_i\tau}$ pour $u = 1, \ldots, k$.

D'où, ${\sigma_i}^{2\tau} = (x^{4\Delta})^{\tau d_i}$ avec $|\tau| < B'$. Nous pouvons simplifier cette équation par τ si τ est premier avec $p'q'$. Nous obtenons donc $x^{4\Delta d_i} = {\sigma_i}^2$ si $B' < B$.

Soit $\tilde{\pi}(k_1)$ la probabilité de succès de $\tilde{P}$. Si $\tilde{\pi}(k_1)$ est non-négligeable, il existe un entier c tel que $\tilde{\pi}(k_1) \geq 1/{k_1}^c$ pour infiniment beaucoup de valeurs k_1. La probabilité pour $\tilde{P}$ de générer une part de signature correcte alors que les vk_i génèrent le groupe Q_N est plus grande que $\tilde{\pi}(k_1) - 2 \times \frac{2}{k-1} \times \frac{1}{B^{k-1}}$ selon le résultat de la section 15 (Génération d'un groupe cyclique avec plusieurs éléments.). Donc si $k = \Theta(k_1/\log(B))$, pour infiniment beaucoup de valeurs k_1, $2 \times \frac{2}{k-1} \times \frac{1}{B^{k-1}} \leq 1/3{k_1}^c$.

De plus, pour k_1 suffisamment grand, $1/{B'}^h < 1/3{k_1}^c$ si $h \times \log(B') = \Theta(k_1)$. Donc en prenant $\varepsilon = \tilde{\pi}(k_1)/3$ dans le lemme 4 nous pouvons en conclure qu'il est possible d'obtenir (σ, τ) en temps polynomial $O(1/\varepsilon) = O({k_1}^c)$. □

Statistiquement Zero-Knowledge.

Preuve: De plus, nous pouvons prouver que si A est beaucoup plus grand que $B' \times N$, le protocole ne donne statistiquement aucune information sur le secret. Dans le modèle de l'oracle aléatoire où l'attaquant a un contrôle total des valeurs retournées par la fonction de hachage H, on définit les b' premiers bits de la fonction de hachage H au point

$$(\mathsf{vk}_1, \ldots, \mathsf{vk}_k, x^{4\Delta}, \mathsf{vk}_{1,i}, \ldots, \mathsf{vk}_{k,i}, {\sigma_i}^2, \mathsf{vk}_1{}^{z}\mathsf{vk}_{1,i}{}^{-e}, \ldots, \mathsf{vk}_k{}^{z}\mathsf{vk}_{k,i}{}^{-e}, x^{4\Delta z}{\sigma_i}^{-2e})$$

comme valant e. Avec probabilité écrasante, l'attaquant n'a pas encore défini l'oracle aléatoire en ce point et l'adversaire $\mathcal{A}$ ne peut pas détecter la fraude. □

Génération d'un groupe cyclique avec plusieurs éléments

Dans cette section nous prouvons qu'avec forte probabilité nous pouvons générer le groupe des carrés Q_N en entier avec seulement quelques éléments pris au hasard.

Théorème 16. Avec probabilité supérieure à $1 - 2 \times \frac{2}{k-1} \times \frac{1}{B^{k-1}}$, un k-uplet $(v_1, \ldots, v_k)$ pris au hasard génère Q_N.

Définissons d'abord quelques notations supplémentaires. Si $(v_1, \ldots, v_k)$ est un k-uplet de $(Q_N)^k$, nous utilisons la notation $\langle v_1, \ldots, v_k \rangle$ pour représenter le sous-groupe de Q_N qui est généré par les v_i, *i.e.*,

$$\langle v_1, \ldots, v_k \rangle = \{x \in Q_N | \exists (\lambda_1, \ldots, \lambda_k)\ x = \prod_{i=1}^{k} v_i^{\lambda_i} \bmod N\}$$

Nous notons aussi $\zeta(k)$ la fonction Zeta de Riemann définie par $\zeta(k) = \sum_{d=1}^{+\infty} \frac{1}{d^k}$ pout tout entier $k \geq 2$. Si $N = {q_1}^{e_1} \times {q_2}^{e_2} \times \ldots \times {q_j}^{e_j}$, nous notons par $\varphi_k(N)$ la valeur

$$N^k \times (1 - \frac{1}{{q_1}^k})(1 - \frac{1}{{q_2}^k}) \ldots (1 - \frac{1}{{q_j}^k})$$

qui est une généralisation de la fonction d'Euler dans le cas de k générateurs. Finalement, si N n'a pas de facteurs premiers inférieurs à B, nous définissons $\zeta_B(k)$ comme $\sum_{d=B}^{+\infty} \frac{1}{d^k}$.

Nous pouvons aussi noter que $\frac{p-1}{2}$ et $\frac{q-1}{2}$ sont premiers entre eux. Pour trouver un générateur v de Q_N, nous devons trouver un élément v tel que $(v \bmod p)$ génère Q_p et $(v \bmod q)$ génère Q_q.

Estimons la probabilité que $x \in Q_N$ soit un générateur de Q_N. La probabilité d'obtenir un tel nombre dépend de la factorisation de p' et de q'. Mais, même si $M = p'q'$ n'a pas de petits facteurs, la probabilité d'obtenir un tel générateur n'est pas écrasante. En effet, si on tire un élément v au hasard dans Q_p, la probabilité que v soit un générateur de Q_p est

$$\mathsf{Pr} \quad = \quad \Pr_{v \in Q_p}[\langle v \rangle = Q_p] = \frac{\varphi(p')}{p'} = \prod_{p_i \geq B, p_i | p-1} (1 - \frac{1}{p_i}) \leq 1 - \frac{1}{p_1}$$

et si $p_1 \leq 2B$, nous pouvons borner la probabilité par $\leq 1 - \frac{1}{2B}$. La probabilité que $B \leq p_1 \leq 2B$ est égale à la probabilité que p' soit divisible par au moins un nombre premier dans $[B, 2B]$, donc

$$\begin{aligned} \Pr_{p_1}[B \leq p_1 \leq 2B] &= \sum_{B \leq q_i \leq 2B, q_i\ premier} \frac{1}{q_i} \\ &\geq \frac{1}{2B} \times (\pi(2B) - \pi(B)) \end{aligned}$$

si on note par $\pi(x)$ le nombre de nombres premiers entre 2 et x. Si $B = 2^{16}$, avec probabilité $\geq 1/23$, $\mathsf{Pr} \leq 1 - \frac{1}{2^{17}}$, en fait avec probabilité $1/17$ exactement. Par conséquent, nous ne pouvons pas dire que cette probabilité est écrasante.

Cependant, si nous nous permettons de prendre plusieurs éléments au hasard dans Q_N, alors le sous-groupe $\langle v_1, \ldots, v_k \rangle$ est égal à Q_N avec forte probabilité. Un k-uplet $(v_1, \ldots, v_k)$ est un ensemble de générateurs de Q_N si $(v_1 \bmod p, \ldots, v_k \bmod p)$ est un ensemble de générateurs de Q_p et si $(v_1 \bmod q, \ldots, v_k \bmod q)$ est un ensemble de générateurs de Q_q. Ainsi, le nombre de k-uplets de $(Q_N)^k$ qui

génèrent Q_N est le nombre de k-uplets qui vu comme éléments de $(Q_p)^k$ génère Q_p et qui vu comme éléments de $(Q_q)^k$ génère Q_q.

Il y a $p' = \frac{p-1}{2}$ éléments dans Q_p. Pour générer ce sous-groupe cyclique de $\mathbb{Z}_p^*$ (car un sous-groupe d'un groupe cyclique est un groupe cyclique), il y a $\varphi(p')$ générateurs.

L'analyse faite par Poupard et Stern dans [157] peut être étendue dans notre contexte car elle est vraie en général dans tout groupe cyclique et pas seulement dans $\mathbb{Z}_{p^e}^*$. Nous présentons tout d'abord un lemme préliminaire.

Lemme 5 *Le nombre des k-uplets de $(Q_p)^k$ qui génèrent Q_p est $\varphi_k(p')$.*

Preuve: Soit $(v_1, \ldots, v_k)$ un k-uplet de $(Q_p)^k$ et v un générateur de Q_p; pour $i = 1, \ldots, k$, nous définissons $\alpha_i \in \mathbb{Z}_{p'}$ par la relation $v^{\alpha_i} = v_i \bmod p$.

Nous remarquons d'abors que $(v_1, \ldots, v_k)$ génère Q_p si et seulement si l'idéal engendré par $\alpha_1, \ldots, \alpha_k$ dans l'anneau $\mathbb{Z}_{p'}$ est l'anneau tout entier. L'égalité de Bezout montre que cet événement se produisait si et seulement si $\mathrm{pgcd}(\alpha_1, \ldots, \alpha_k, p') = 1$.

Comptons le nombre des k-uplets $(\alpha_1, \ldots, \alpha_k) \in (Q_p)^k$ tels que $\mathrm{pgcd}(\alpha_1, \ldots, \alpha_k, p') = 1$.

Soit $\prod_{i=1}^{t'} {q_i}^{f_i}$ la décomposition en facteurs premiers de p'. Nous savons que

$$\mathrm{pgcd}(x, \prod_{i=1}^{t'} {q_i}^{f_i}) = 1 \iff \forall i \leq t',\ \mathrm{pgcd}(x, {q_i}^{f_i}) = 1 \iff \forall i \leq t',\ \mathrm{pgcd}(x \bmod {q_i}^{f_i}, {q_i}^{f_i}) = 1$$

En utilisant le théorème des restes chinois, le problème se réduit à compter le nombre de k-uplets $(\beta_1, \ldots, \beta_k)$ de $(\mathbb{Z}_{{q_i}^{f_i}})^k$ tels que $\mathrm{pgcd}(\beta_1 \bmod {q_i}^{f_i}, \ldots, \beta_k \bmod {q_i}^{f_i}, {q_i}^{f_i}) = 1$ pour $i = 1, \ldots, t'$. Les k-uplets qui ne vérifient pas cette relation pour un indice i fixé sont de la forme $(q_i\gamma_1, \ldots, q_i\gamma_k)$ où $(\gamma_1, \ldots, \gamma_k) \in {\mathbb{Z}_{{q_i}^{f_i-1}}}^k$ et il y en a exactement ${q_i}^{k(f_i-1)}$.

Finalement, il y a $\prod_{i=1}^{t'}({q_i}^{kf_i} - {q_i}^{k(f_i-1)})$ k-uplets de $(\mathbb{Z}_{p'})^k$ tels que $\mathrm{pgcd}(\alpha_1, \ldots, \alpha_k, p') = 1$ et ceci est égal à $\prod_{i=1}^{t'} \varphi_k({q_i}^{f_i}) = \varphi_k(p')$ car φ_k est une fonction faiblement multiplicative. □

Maintenant, nous retournons vers la preuve du théorème 16. Voici quelques notations supplémentaires : pour chaque entier x, S_x représente l'ensemble des indices i tels que p_i soit un facteur de x. D'après le lemme précédent, nous savons que la probabilité pour qu'un k-uplet de $(Q_p)^k$ génère Q_p est $\frac{\varphi_k(p')}{{p'}^k}$, notée $\mathsf{pr} = \prod_{i \in S_{p'}} 1 - \frac{1}{{p_i}^k}$. L'inverse de chaque terme $1 - \frac{1}{{p_i}^k}$ peut être développé en série de puissance : $(1 - \frac{1}{{p_i}^k})^{-1} = \sum_{j=0}^{\infty}(1/{p_i}^k)^j$. La probabilité pr est un produit de puissances avec des termes positifs, $\mathsf{pr} = (\prod_{i \in S_{p'}} \sum_{\alpha_i=0}^{\infty} \frac{1}{{p_i}^{\alpha_i k}})^{-1}$ et par conséquent, nous pouvons distribuer les termes et obtenir que pr^{-1} est la somme de $1/d^k$ où d parcourt les entiers dont les facteurs premiers sont parmi les p_i, $i \in S_{p'}$. Cette somme est inférieure à la somme non restreinte $\sum_{d=1}^{\infty} 1/d^k = \zeta(k)$. Finalement, nous obtenons que $\mathsf{pr} > 1/\zeta(k)$.

Dans notre cas, ni p' ni q' n'ont des petits facteurs premiers inférieurs à B, et ainsi $\mathsf{pr} > \frac{1}{1+\zeta_B(k)}$.

$$\begin{aligned} \zeta_B(k) = \sum_{d=B}^{\infty} 1/d^k &\leq 1/B^k + \int_B^{\infty} dx/x^k \\ 1 + \zeta_B(k) &\leq 1 + \frac{k-1+B}{k-1} \times \frac{1}{B^k} \end{aligned}$$

Comme pour tout $x > -1$, $1/(1+x) \geq 1 - x$, on a :

$$\frac{1}{1+\zeta_B(k)} \leq 1 - \frac{k-1+B}{k-1} \times \frac{1}{B^k}$$

Par conséquent, le nombre de k-uplets de $(Q_p)^k$ qui génèrent Q_p est $\varphi_k(p')$ et

$$\Pr_{(v_1,\ldots,v_k)\in(Q_p)^k}[\{\langle v_1,\ldots,v_k\rangle] = Q_p\} = \frac{\varphi_k(p')}{p'^k} > \frac{1}{1+\zeta_B(k)} \leq 1 - \frac{k+B-1}{k-1} \times \frac{1}{B^k}$$

Donc, avec probabilité supérieure à $1 - 2 \times \frac{2}{k-1} \times \frac{1}{B^{k-1}}$, les k-uplets $(v_1,\ldots,v_k)$ génèrent Q_p et Q_q, et donc Q_N. Par exemple, avec $k = 6$ et $B = 2^{16}$, cette probabilité est supérieure à $1 - 1/2^{80}$.

3.4.6 Paramètres pratiques

Dans la génération de clés, nous devons tester si p, q, p' et q' sont divisibles par de petits nombres premiers $\leq B$ et si $\mathrm{pgcd}(p', q') = 1$. Nous pouvons faire comme hypothèse que B est le premier nombre premier supérieur à 2^{16}. La perte dans la première phase de la génération de clé est un facteur 80 en moyenne. En effet, nous avons vu dans la preuve du théorème 2 section 11 que $\Pr_{p'}[p' \text{ n'a pas de petits facteurs premiers} \leq B] \approx \frac{1}{e^\gamma \ln(B)}$. Si nous fixons B à 2^{16}, nous avons $\Pr_{p'}[p' \text{ n'a pas de petits facteurs premiers} \leq B] > \frac{1}{20}$. Par conséquent, pour générer p et q tels que ni p' ni q' ont de petits facteurs premiers et tels que $\mathrm{pgcd}(p-1, q-1) = 2$, nous devons exécuter en moyenne $2 \times (20 + 20) = 80$ fois ce protocole. Ce facteur n'est pas critique car cet algorithme n'est exécuté qu'une seule fois. Dans le modèle *honnête-mais-curieux*, ceci représente environ 2 heures pour générer une clé RSA qui soit conforme à nos exigences.

Dans la preuve de robustesse, si nous voulons avoir un paramètre de sécurité de 2^{80}, nous devons choisir $B' = 2^{16} < B$. Ainsi, nous devons prendre $h = 5$ tours. Pour générer le groupe des carrés avec probabilité supérieure à $1 - 2^{80}$, nous devons utiliser $u = 6$ clés de vérification. Par conséquent, nous avons besoin de 30 preuves de robustesse mais ceci est acceptable par les applications visées.

De plus, la condition $n < B$ n'est pas très restrictive sur le nombre de serveurs.

3.5 Autre solution

Dans cette section, on mentionne rapidement la proposition de Damgård et Koprowski dans [57] pour distribuer complètement le schéma RSA. Les deux travaux ont été réalisés de manière indépendante. Ces derniers ont revisité la preuve de Shoup pour étudier les modifications qu'induisent l'utilisation d'un module RSA sans exigence particulière. Dans ce travail, ils font appel à deux hypothèses pour résoudre les difficultés rencontrées :

1. la génération d'un carré v d'ordre maximal dans Q_N, et d'une clé publique RSA (N, e), est indistinguable de la génération d'un carré v au hasard et d'une clé publique RSA (N, e), et

2. il est difficile de trouver un élément dont l'ordre ne soit pas divisible par le plus grand facteur k de $\varphi(N)/4$.

La seconde hypothèse permet d'avoir une probabilité de fraude (significatif) en $1/k$ où k est le plus grand facteur de $p-1$ ou de $q-1$ car trouver des éléments dont l'ordre ne soit pas divisible par k est difficile selon l'article[47].

47. Cette hypothèse est en fait très forte. Afin de prouver la sécurité de la preuve de Damgård et Koprowski, il suffit de supposer qu'il est difficile de trouver un élément dont l'ordre ne soit pas divisible par un grand facteur de $\varphi(N)$, disons dont la taille est > 80 bits, c'est-à-dire dont l'ordre soit composé de petits facteurs. Cette hypothèse est différente de celle de trouver un élément d'ordre "petit" dont les facteurs de l'ordre soient de taille < 40 bits, qui est équivalente à la factorisation dans le cas où $\mathrm{pgcd}(p-1, q-1) = 2$, c'est-à-dire en utilisant notre algorithme de génération de modules RSA.

Afin de réaliser cette hypothèse, la preuve de Shoup décrite précédemment et celle de Gennaro *et al.* (cf. annexe 9) utilisent des nombres premiers sûrs. Par conséquent, cette hypothèse clôt le problème sans vraiment le résoudre.

Pour éviter la seconde hypothèse dans ce chapitre, nous avons fait une preuve zero-knowledge dont la preuve de sécurité (significative) utilise certes la décomposition en facteur premiers de l'ordre du groupe, mais a seulement besoin de garantir qu'il n'y a pas de petits facteurs. En effet, on veut pouvoir inverser une petite quantité $< B$ modulo l'ordre du groupe. Dans le cas d'une preuve de la significativité *à la Shoup* [174], on doit prouver que le challenge est uniquement déterminé modulo p' ou q' qui est un grand nombre premier, ce qui est en contradiction avec le *modèle de l'oracle aléatoire* [9], et donc la probabilité de tomber sur ce cas est $< 1/\min(p', q')$. Damgård et Koprowski ont montré que, s'il est difficile de trouver un élément dont l'ordre n'est pas divisible par k, alors, le challenge est uniquement déterminé modulo k.

Enfin, pour contourner la première hypothèse, nous avons utilisé plusieurs générateurs. Tout ceci, rend bien évidemment notre schéma moins efficace. Cependant, nous pensons que les preuves de sécurité que nous avons faites, sont un gage de sécurité plus important que ces hypothèses. Suivant le besoin des utilisateurs (efficacité versus sécurité), les deux protocoles sont utiles pour partager complètement RSA.

3.6 Conclusion

Dans ce chapitre, nous avons montré comment éviter l'utilisation de module RSA sûrs de telle sorte que la preuve reste correcte. Nous avons considéré un environnement nécessitant un fort degré de sécurité comme les protocoles de vote électronique et par conséquent, nous avons besoin de protocoles utilisant uniquement des hypothèses standards.

Nous avons eu recours à trois techniques différentes pour prouver que :

- le groupe des carrés est cyclique,
- nous générons p et q tels que p' et q' ne contiennent pas de petits facteurs premiers, ce qui nous permet de générer le groupe Q_N,
- en choisissant au hasard quelques éléments dans Q_N, ces éléments engendrent Q_N avec forte probabilité car l'ordre de ce groupe n'a pas de petits facteurs.

Les protocoles proposés dans cette partie pour distribuer *complètement* la signature RSA peuvent aussi être utilisés pour distribuer le déchiffrement. Cependant, dans le cas du déchiffrement, nous montrerons dans les chapitres suivants que nous ne pouvons tolérer que des adversaires passifs avec RSA. Nous montrerons aussi que les protocoles de ce chapitre peuvent être utilisés pour distribuer d'autres cryptosystèmes comme celui de Paillier.

4

Partages du cryptosystème de Paillier

Dans ce chapitre nous exposons le partage du système de chiffrement de Paillier présenté au congrès Financial Crypto ’00 [72] avec Guillaume Poupard et Jacques Stern. Nous en profitons aussi pour présenter une version *complètement* distribuée de ce cryptosystème en utilisant les techniques du chapitre précédent.

Les applications de ce cryptosystème sont nombreuses et nous en décrirons quelques unes dans la troisième partie de cette thèse. Dans une première section, nous rappellons les différents cryptosystèmes homomorphes basés sur les hauts-résidus. Puis nous décrivons deux propositions pour partager le déchiffrement du cryptosystème de Paillier.

Sommaire

4.1 Introduction

Nous nous intéressons ici à une famille de schémas basés sur un système de chiffrement très simple qui consiste à calculer dans une certaine base une exponentiation du message à chiffrer. La sécurité est reliée à la difficulté de calculer le logarithme discret. La connaissance d’une trappe, la clé secrète,

permet de déchiffrer efficacement. Nous appelons TDLS (Trapdoor Discrete Log Schemes) les schémas pour lesquels une trappe permet de calculer un logarithme discret. Ces protocoles ont une propriété d'homomorphisme intéressante : un chiffré de la somme de deux messages est obtenu par produit de chiffrés de chacun d'entre eux. Ceci peut être utilisé dans les applications qui nécessitent des calculs avec des quantités secrètes : protocoles de vote [56, 1], protocoles de loterie [72], ou encore tableur avec des données secrètes. En effet, en utilisant des méthodes de réduction de réseau en dimension 2 que nous verrons au chapitre 6, il est possible de chiffrer et déchiffrer des nombres rationnels. Un tableur sous Excel avec des données secrètes a été réalisé par Geert-Jan Wackers [73].

Dans les applications de vote ou de loterie, la capacité à déchiffrer, c'est-à-dire la connaissance de la clé privée, donne un pouvoir important. Un moyen classique de réduire la confiance nécessaire dans l'autorité et d'augmenter la disponibilité du système consiste à partager le secret entre plusieurs autorités de sorte que la réunion d'un certain nombre d'entre elles soit nécessaire pour déchiffrer. Les protocoles à seuil sont très utiles pour ces applications et il est donc important de partager les cryptosystèmes mis en jeu.

Dans l'article [72], nous avons partagé un cryptosystème homomorphique où l'algorithme de déchiffrement est partagé entre plusieurs serveurs. Pour déchiffrer, chaque serveur commence par calculer une *part du déchiffrement*. Ensuite, un algorithme public permet de recombiner les parts et de retrouver le clair. La plupart des systèmes de chiffrement comme celui de Goldwasser-Micali [98], Benaloh [16, 50], Naccache-Stern [128], Okamoto-Uchiyama [136] ou Paillier [139] ont besoin de distribuer une valeur secrète reliée à la factorisation d'un module RSA. Pour ce faire, nous avons utilisé les techniques du chapitre précédent sur le cryptosystème RSA et nous les avons étendues à notre contexte de déchiffrement.

Ce faisant, nous avons dû revoir les définitions du modèle de sécurité afin d'analyser la sécurité sémantique de ces schémas contre des attaques à clairs choisis (CPA) dans le cas distribué. Des définitions précédentes pour les cryptosystèmes à seuil sûrs contre les attaques à chiffrés choisis (CCA) ont été formalisées comme une extension naturelle des définitions standards de la sécurité CCA dans [175]. En suivant ce travail, nous avons proposé des définitions adéquates pour garantir la sécurité CPA des cryptosystèmes à seuil. Nous reviendrons dans le chapitre suivant sur les notions IND-CCA.

4.2 Rappels sur les algorithmes de chiffrement homomorphe

Plusieurs cryptosystèmes basés sur des schémas de chiffrement randomisés $\mathcal{E}(M)$ chiffrent un message M en calculant g à la puissance M [98, 16, 50, 182, 128, 136, 139]. Leur sécurité est basée sur la difficulté de calculer le logarithme discret en base g. Ce calcul est facile si l'on utilise une trappe qui est la clé secrète. Une conséquence importante de ces schémas de chiffrement est qu'ils ont des propriétés d'homomorphisme qui peuvent être résumées de la manière suivante :

$$\mathcal{E}(M_1 + M_2) \equiv \mathcal{E}(M_1) \times \mathcal{E}(M_2) \quad \text{et} \quad \mathcal{E}(k \times M) \equiv \mathcal{E}(M)^k$$

4.2.1 Le cryptosystème Goldwasser-Micali

Goldwasser et Micali ont proposé en 1984 [98] le premier système de chiffrement sémantiquement sûr. Ce système permet de chiffrer un bit à la fois et a la propriété que $\mathcal{E}(b) \times \mathcal{E}(b') \equiv \mathcal{E}(b \oplus b')$. Cette propriété a été récemment utilisée dans [113] pour construire un compteur cryptographiquement sûr avec des registres à décalage.

Génération des clés

Soit $N = pq$ un module RSA. La clé publique pk est constituée de (N, y) où y est un non résidu quadratique, *i.e.* $(y|p) = (y|q) = -1$. La clé secrète sk est constituée de la factorisation de N, par exemple p.

Chiffrement

Pour chiffrer un bit b, l'émetteur tire aléatoirement $z \in \mathbb{Z}_N^*$ et calcule $c = \mathcal{E}(b, z) = y^b z^2 \bmod N$.

Déchiffrement

Pour déchiffrer c, le receveur calcule $(c|p) = c^{\frac{p-1}{2}} \bmod p$ qui vaut 1 si c est un résidu quadratique, *i.e.* si b vaut 0, et -1 si c est un non-résidu quadratique, *i.e.* si b vaut 1.

Sécurité

Il est clair que la sécurité sémantique de ce schéma est basée sur le problème de la résiduosité quadratique qui consiste à décider si un élément aléatoire x de $\mathbb{Z}_N^*$ est un carré ou non.

4.2.2 Le cryptosystème de Benaloh et Fisher

Afin d'augmenter la longueur du clair, Benaloh et Fisher dans [16, 50] ont proposé une variante fondée sur la difficulté de décider si un élément aléatoire de $\mathbb{Z}_N^*$ est une puissance r-ième modulo un nombre composé. Ceci généralise le problème de la résiduosité quadratique. L'inconvénient de ce cryptosystème est que le déchiffrement est lent. Une optimisation utilisant la méthode Pollard-Rho permet de déchiffrer avec une complexité en $O(\sqrt{r})$.

Génération des clés

Soit $N = pq$ un module RSA tel que r divise $p - 1$ mais sans diviser $(p - 1)/r$ et $(q - 1)$. Ainsi r divise $\varphi(N) = (p - 1)(q - 1)$ et $\mathrm{pgcd}(r, \frac{\varphi(N)}{r}) = 1$. La clé publique pk est constituée de N, r et d'un élément g de $\mathbb{Z}_N^*$ qui ne soit pas une puissance r-ième. La clé secrète sk est $\varphi(N)$.

Chiffrement

Pour chiffrer un message $m \in \mathbb{Z}_r$, l'émetteur tire aléatoirement $z \in \mathbb{Z}_N^*$ et calcule $c = \mathcal{E}(m, z) = g^m z^r \bmod N$.

Déchiffrement

Pour déchiffrer c, le receveur doit tester pour tout $j \in \mathbb{Z}_r$ si $c \times g^{-j}$ est un r-résidu, c'est-à-dire si $(c \times g^{-j})^{\frac{\varphi(N)}{r}} = 1 \bmod N$ et dans ce cas, $m = j$.

4.2.3 Le cryptosystème Naccache-Stern

Naccache et Stern dans [128] ont proposé en 1998 une variante permettant d'augmenter la taille du clair et de rendre plus efficace le déchiffrement. L'idée consiste à utiliser l'algorithme de Pohlig-Hellman [145].

Génération des clés

Soit N un module RSA et $\sigma = \prod_{i=1}^{k} p_i$ un diviseur de $\varphi(N)$ tel que $\mathrm{pgcd}(\sigma, \frac{\varphi(N)}{\sigma}) = 1$ où les p_i sont des nombres premiers deux à deux distincts. La clé publique pk est constituée de N, de σ et d'un élément g de $\mathbb{Z}_N^*$ d'ordre égal à un multiple de σ. La clé secrète sk est constituée du k-uplet $(p_1, \ldots, p_k)$ et de $\varphi(N)$.

Chiffrement

Pour chiffrer un message $m \in \mathbb{Z}_\sigma$, l'émetteur tire aléatoirement $z \in \mathbb{Z}_N^*$ et calcule $c = g^m z^\sigma \bmod N$.

Déchiffrement

Pour déchiffrer un chiffré c, le receveur utilise le mécanisme précédent modulo chaque p_i et reconstruit le clair modulo σ grâce au théorème des restes chinois.

Sécurité

Ce schéma est sémantiquement sûr si le problème dit des *hauts résidus* qui consiste à distinguer des puissances σ-ièmes est difficile.

4.2.4 Le cryptosystème d'Okamoto-Uchiyama

Le mécanisme de déchiffrement des systèmes précédents consiste à tester toutes les puissances, mais il n'existe pas à proprement parler de trappe pour calculer un logarithme discret. Okamoto et Uchiyama ont proposé en 1998 dans [136] un schéma qui permet de déchiffrer sans utiliser de recherche exhaustive. Ainsi, l'algorithme de déchiffrement est indépendant de la longueur du clair. Pour ce faire, ils ont utilisé un module de la forme $N = p^2 q$. L'idée consiste à remarquer que $(1+p)^m = 1+mp \bmod p^2$ en utilisant la formule du binôme de Newton.

Génération des clés

Soit $N = p^2 q$ où p est de taille k publique, $(2^{k-1} < p < 2^k)$, et g un élément de $\mathbb{Z}_N^*$ tel que $g^{p-1} \bmod p^2$ soit d'ordre p. La clé secrète sk est le facteur p.

Chiffrement

Pour chiffrer un message $m \in [0, 2^{k-1}[$, l'émetteur tire aléatoirement $z \in \mathbb{Z}_N^*$ et calcule $c = g^m z^N \bmod N$.

Déchiffrement

Pour déchiffrer c, il suffit d'appliquer la formule

$$m = \frac{(c^{p-1} - 1 \bmod p^2)/p}{(g^{p-1} - 1 \bmod p^2)/p} \bmod p \tag{4.1}$$

En réception, le receveur vérifie si $m < 2^{k-1}$. L'équation 4.1 peut être vérifiée en remarquant que g^{p-1} est d'ordre p modulo p^2. De plus, il est égal à $1 \bmod p$, donc il existe k tel que $g^{p-1} \bmod p^2 = 1 + kp$.

En outre, pour tout $x \in \mathbb{Z}_N^*$, $x^{p(p-1)} = 1 \bmod p^2$ car $\varphi(p^2) = p(p-1)$. Ainsi,

$$c^{p-1} = g^{m(p-1)} z^{N(p-1)} = g^{m(p-1)} = (1+kp)^m = 1 + kmp \bmod p^2$$

Ce schéma est sûr contre les attaques à clair choisi. Cependant, si on ne prend pas la précaution de vérifier si $m < 2^{k-1}$ et d'utiliser des conversions génériques comme [82], une attaque à chiffrés choisis peut être mise en œuvre aisément [111] en chiffrant un message $x > p$. Dans ce cas, le déchiffrement retournera le résultat $x' < p$ tel que $x - x'$ est un multiple de p. En calculant $\mathrm{pgcd}(N, x - x')$, on obtient p.

Sécurité

La sécurité one-wayness est basée sur la factorisation en utilisant la même technique que dans l'attaque précédente. En effet, s'il existe un adversaire qui sait inverser la fonction de chiffrement, alors en chiffrant une valeur supérieure à p, et en lui demandant de déchiffrer le chiffré obtenu, on peut construire un attaquant qui obtient p et sait casser le module RSA. Il reste à prouver que le chiffré obtenu en chiffrant une valeur supérieure à p est indistinguable de tout chiffré dont le clair est pris dans $[0, 2^{k-1}[$.

La sécurité sémantique de ce cryptosystème est basée sur l'hypothèse p-sous-groupe qui dit que l'on ne peut pas distinguer une puissance p-ième. L'inconvénient de cette hypothèse est que p est une quantité secrète.

4.2.5 Le cryptosystème de Paillier

En continuant l'idée d'Okamoto et Uchiyama, Paillier a proposé un cryptosystème utilisant des modules de la forme $N = p^2q^2$. Dans la suite, on utilisera un module RSA $N = pq$ et on calculera modulo $N^2 = p^2q^2$. Paillier a présenté trois cryptosystèmes reliés entre eux dans [139]. On utilise seulement le premier d'entre eux. Par rapport aux précédents schémas, ce cryptosystème est celui qui a la plus grande bande passante, appelée aussi taux d'expansion : rapport entre la longueur du clair et la longueur du chiffré.

Ce cryptosystème est basé sur les propriétés de la fonction lambda de Carmichael dans $\mathbb{Z}_{N^2}^*$. On rappelle les deux principales propriétés : pour tout $w \in \mathbb{Z}_{N^2}^*$,

$$w^{\lambda(N)} = 1 \bmod N, \quad \text{et} \quad w^{N\lambda(N)} = 1 \bmod N^2$$

Génération des clés

Soit N un module RSA $N = pq$, où p et q sont des nombres premiers. Soit g un entier d'ordre un multiple de N modulo N^2. La clé publique est $\mathsf{pk} = (N, g)$ et la clé secrète est $\mathsf{sk} = \lambda(N)$.

Chiffrement

Pour chiffrer un message $M \in \mathbb{Z}_N$, on choisit au hasard x dans $\mathbb{Z}_N^*$ et on calcule le chiffré $c = g^M x^N \bmod N^2$.

Déchiffrement

Pour déchiffrer c, on calcule

$$M = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod n \tag{4.2}$$

où la fonction L prend ses entrées dans l'ensemble $\mathcal{S}_N = \{u < N^2 | u = 1 \bmod N\}$ et calcule $L(u) = \frac{u-1}{N}$.

Les entiers $c^{\lambda(N)} \bmod N^2$ et $g^{\lambda(N)} \bmod N^2$ sont égaux à 1 quand ils sont élevés à la puissance N, ce sont donc des racines N-ième de l'unité. De telles racines[48] sont de la forme $(1+N)^\ell = 1+\ell N \bmod N^2$. L'ensemble $\mathcal{S}_N$ est le sous-groupe de $\mathbb{Z}^*_{N^2}$ des racines N-ièmes de l'unité[49]. La fonction L permet de calculer les valeurs $\ell \bmod N$ de ces racines. En effet : $L(1 + \ell N) = \frac{1+\ell N-1}{N} = \ell$. Ainsi, si on note g sous la forme $g = (1 + N)^a b^N \bmod N^2$, alors

$$\begin{aligned}
L(c^{\lambda(N)} \bmod N^2) &= L(g^{M\lambda(N)} x^{N\lambda(N)} \bmod N^2) \\
&= L((1 + N)^{aM\lambda(N)} (xb^M)^{N\lambda(N)} \bmod N^2) \\
&= L((1 + N)^{aM\lambda(N)} \bmod N^2) \\
&= Ma\lambda(N) \bmod N \\
&= M \times L((1 + N)^{a\lambda(N)} \bmod N^2) \\
&= M \times L((1 + N)^{a\lambda(N)} b^{N\lambda(N)} \bmod N^2) \\
&= M \times L(g^{\lambda(N)} \bmod N^2)
\end{aligned}$$

D'où la formule de déchiffrement.

Sécurité

Il est conjecturé que le problème de classe de résiduosité composite qui consiste exactement à inverser ce cryptosystème est difficile. La sécurité sémantique est basée sur la difficulté de distinguer les résidus d'ordre N modulo N^2 des non-résidus d'ordre N. Les résidus d'ordre N modulo N^2 sont les éléments x de $\mathbb{Z}_{N^2}$ tels qu'il existe y tel que $x = y^N \bmod N^2$. Ce problème est en rapport avec l'indistinguabilité des résidus quadratiques dans $\mathbb{Z}_N$.

On note $CR[N]$ le problème de décider les résidus de degré N, *i.e.* distinguer les résidus de degré N des non-résidus de degré N. La sécurité sémantique du schéma de Paillier avec comme modulo N est équivalent au problème $CR[N]$. Le lecteur intéressé par ce problème pourra lire la thèse de Pascal Paillier [138] pour plus de détails. Dans la suite, le problème *Decisional Composite Residuosity Assumption* (DCRA) fait comme hypothèse que $CR[N]$ est difficile.

4.3 Première proposition

Dans cette section, nous allons décrire un algorithme pour partager le déchiffrement du cryptosystème de Paillier. Nous avons présenté ce système dans l'article [72]. Finalement, nous décrivons une version *complètement* distribuée.

48. En effet, tout élément w de $\mathbb{Z}^*_{N^2}$ peut s'écrire de la forme $w = (1+N)^a b^N \bmod N^2$ pour $a \in \mathbb{Z}_N$ et $b \in \mathbb{Z}^*_N$ à cause de la bijection entre $\mathbb{Z}^*_{N^2}$ et $\mathbb{Z}_N \times \mathbb{Z}^*_N$. Ainsi, w est une racine N-ième de l'unité, si $w^N = \left((1 + aN)^N b^{N^2} = b^{N^2} =\right) 1 \bmod N^2$ d'après la formule du binôme de Newton. Par conséquent, $\text{ord}_{N^2}(b) | N^2$ et $\text{ord}_{N^2}(b) | N\lambda(N)$. Or $\lambda(N)$ ne divise pas N et donc $\text{ord}_{N^2}(b) | N$, donc $b^N = 1$. Les racines N-ièmes de l'unité sont donc de la forme $(1 + N)^a$ pour $a = 0, \ldots, N - 1$.

49. Il existe une bijection entre $\mathbb{Z}^*_{N^2}$ et $\mathcal{S}_N \times R_N$, où R_N représente le sous-groupe des résidus N-ièmes de $\mathbb{Z}^*_{N^2}$. C'est-à-dire, tout élément de $\mathbb{Z}^*_{N^2}$ peut s'écrire sous la forme $(1 + N)^a \times b^N \bmod N^2$.

4.3.1 Sécurité

Les joueurs et le scénario

Le protocole comprend les joueurs suivants : un distributeur, un combineur, un ensemble de n serveurs P_i, un adversaire et les utilisateurs. Ils sont tous représentés par des machines de Turing probabilistes. Considérons le scénario suivant :

- Dans une phase d'initialisation, le distributeur utilise l'algorithme de génération de clés pour créer les clés publiques, privées et de vérification. La clé publique pk et toutes les clés de vérification $\mathsf{vk}, \mathsf{vk}_i$ sont rendues publiques et chaque serveur reçoit sa part sk_i de la clé privée sk.
- Pour chiffrer un message, chaque joueur peut exécuter l'algorithme de chiffrement en utilisant la clé publique pk.
- Pour déchiffrer un chiffré c, le combineur envoie c à tous les serveurs. En utilisant leurs parts sk_i de la clé privée sk et leurs clés de vérification $\mathsf{vk}, \mathsf{vk}_i$, chaque serveur exécute l'algorithme de déchiffrement et retourne un déchiffrement partiel σ_i avec une preuve de validité de la part de déchiffrement preuve_i. Finalement, le combineur utilise l'algorithme de combinaison qui permet de retrouver le clair si suffisamment de parts de déchiffrement sont valides.

Définition formelle

Rappelons ici la définition formelle d'un cryptosystème partagé en présentant uniquement les algorithmes les plus significatifs. Un cryptosystème à seuil est composé de 4 algorithmes :

- Un *algorithme de génération de clés* prend en entrée un paramètre de sécurité k, le nombre n de serveurs de déchiffrement, le paramètre t de seuil et une chaîne aléatoire ω; il retourne une clé publique pk, une liste $\mathsf{sk}_1, \ldots \mathsf{sk}_n$ de clés privées et une liste $\mathsf{vk}, \mathsf{vk}_1, \ldots \mathsf{vk}_n$ de clés de vérification.
- Un *algorithme de chiffrement* prend en entrée la clé publique pk, une chaîne aléatoire ω et un clair M; il retourne un chiffré c.
- Un *algorithme de déchiffrement de part* prend en entrée une clé publique pk, un index $1 \leq i \leq n$, une clé privée sk_i et un chiffré c; il retourne une part du déchiffrement σ_i et une preuve de validité preuve_i.
- Un *algorithme de combinaison* prend en entrée la clé publique pk, un chiffré c, une liste $\sigma_1, \ldots \sigma_n$ de parts du déchiffré, une liste $\mathsf{vk}, \mathsf{vk}_1, \ldots \mathsf{vk}_n$ de clés de vérifications et une liste $\mathsf{preuve}_1, \ldots \mathsf{preuve}_n$ de preuves de validité ; il retourne un clair M ou échoue.

Modèle de sécurité

On considère un adversaire capable de corrompre jusqu'à t serveurs. Une telle corruption peut être passive, c'est-à-dire que l'attaquant est capable d'intercepter les communications de ces t serveurs et de lire le contenu de leur mémoire. On dit aussi que les serveurs sont *corrompus*. Cet adversaire peut aussi être actif : dans ce cas, il contrôle complètement le comportement des serveurs corrompus, c'est-à-dire qu'il peut écrire dans leur mémoire et envoyer des mauvaises parts de déchiffrement à leur place. Dans la suite, on considère seulement les adversaires non-adaptatifs qui choisissent les serveurs qu'ils veulent corrompre avant la phase de génération des clés. Un adversaire est appelé adaptatif s'il peut corrompre les serveurs à n'importe quel moment du protocole.

Un cryptosystème à seuil est dit *robuste* si le combineur est capable de déchiffrer correctement n'importe quel chiffré, même en présence d'un adversaire qui contrôle activement jusqu'à t serveurs.

Tous les messages sont envoyés en clair entre les serveurs et le combineur. De plus, l'algorithme de combinaison prend chaque part du déchiffrement et permet de retrouver le texte clair. Cet algorithme est dit "public", c'est-à-dire qu'il peut être exécuté par tout le monde et n'a pas besoin de la connaissance d'une quantité secrète. Il peut être exécuté par n'importe quel serveur qui voit passer toutes les parts de déchiffrement. Par conséquent, la seule hypothèse que l'on fait sur le canal de communication est l'existence d'un canal de diffusion *intègre* entre tous les participants. Par canal intègre, on entend que les messages envoyés dessus ne peuvent pas être modifiés par un adversaire. En effet, dans le cas contraire, l'adversaire pourrait changer les messages transmis au combineur par chaque serveur. Ceci aurait pour conséquence de rendre indisponible le service de déchiffrement et le protocole ne serait plus robuste.

Sécurité sémantique à seuil On étend la définition de la sécurité sémantique aux cryptosystèmes dans le cadre où l'adversaire corrompt activement, mais non adaptativement, t serveurs, apprend les paramètres publics comme dans le cryptosystème d'origine mais aussi les clés secrètes des serveurs corrompus, les clés de vérifications publiques et toutes les parts de déchiffrement et les preuves de validité de ces parts.

Soit le jeu A suivant :

A1 L'adversaire choisit de corrompre t serveurs. Il apprend toutes leurs informations secrètes et contrôle activement leur comportement.

A2 L'algorithme de génération des clés s'exécute alors; les clés publiques sont rendues publiques, chaque serveur reçoit sa clé secrète et l'adversaire apprend les secrets des joueurs corrompus.

A3 L'attaquant choisit un message M et un *oracle de déchiffrement partiel* lui donne n parts de déchiffrement valides d'un chiffrement de M avec les preuves de validité. Cette étape est répétée un nombre polynomial de fois par l'adversaire.

A4 L'adversaire émet alors deux messages M_0 et M_1 et les envoie à *l'oracle de chiffrement* qui choisit au hasard un bit b et renvoie le chiffrement c de M_b à l'attaquant.

A5 L'attaquant répète l'étape A3, en demandant les parts de déchiffrement pour des chiffrés $\neq c$ dont le clair est connu.

A6 L'attaquant retourne un bit b'.

Définition :

> Un schéma de chiffrement à seuil est dit sémantiquement sûr contre un adversaire actif non-adaptatif si pour n'importe quel attaquant s'exécutant en temps polynomial, $b = b'$ avec probabilité négligeable par rapport à $1/2$.

On remarque que cette définition de la sécurité sémantique se réduit à la définition d'origine quand on considère un seul serveur $(n = 1)$ qui connaît la totalité de la clé secrète, et un adversaire qui ne corrompt aucun serveur $(t = 0)$. Dans ce cas, les étapes A3 à A5 consistent simplement à chiffrer des clairs choisis et ceci peut être fait sans l'aide d'un *oracle de déchiffrement partiel*.

Finalement, le jeu précédent ne doit pas être confondu avec la sécurité contre les attaques à chiffré choisis décrite par Gennaro et Shoup [175]. L'attaquant peut seulement demander les parts de déchiffrement pour lesquels il connaît le clair correspondant. Le but des étapes A3 et A5 est de prouver que les

déchiffrements partiels ne donnent aucune information sur les clés privées des serveurs non corrompus. Comme les cryptosystèmes utilisés ne sont pas sûrs contre les attaques à chiffrés choisis dans la version originale, on ne peut pas s'attendre à obtenir une telle propriété dans la version à seuil.

Preuve de sécurité: Notre but est de construire une version robuste et sémantiquement sûre du cryptosystème à seuil. Les preuves de sécurité sont basées sur une réduction ; on prouve que si un attaquant peut casser la sécurité sémantique du cryptosystème à seuil, alors il doit pouvoir casser la sécurité sémantique du cryptosystème initial.

À partir d'un adversaire qui peut casser la sécurité sémantique du cryptosystème à seuil, on montre comment construire un adversaire pour attaquer la sécurité sémantique du cryptosystème initial . L'idée de base d'une telle construction est de simuler à l'adversaire toutes les informations supplémentaires qui ne sont pas fournies dans une attaque traditionnelle à l'adversaire.

4.3.2 Description du cryptosystème de Paillier distribué

Le problème essentiel que nous devons résoudre lorsque nous voulons partager ce cryptosystème est que nous devons calculer $L(c^{\mathsf{sk}} \bmod N^2)$ et $L(g^{\mathsf{sk}} \bmod N^2)$. En effet, il n'est pas possible de déchiffrer c, sans calculer auparavant ces deux entiers. Ainsi, à la fin du premier déchiffrement, tous les serveurs obtiennent $L(g^{\lambda(N)} \bmod N^2)$ qui vaut $a\lambda(N) \bmod N$ si $g = (1+N)^a b^N \bmod N^2$.

La question que nous devons résoudre est la suivante : est-ce que cette quantité révèle de l'information sur $\mathsf{sk} = \lambda(N)$? En effet, le théorème de Miller [126] permet de factoriser N lorsque l'on connaît $\lambda(N)$.

Notre solution a été de randomiser la clé secrète $\lambda(N)$ en $\beta\lambda(N)$ dans $\mathbb{Z}^*_{mN}$, où $m = \frac{p-1}{2} \times \frac{q-1}{2}$, pour que la quantité $\theta = a\beta\lambda(N) \bmod N$ soit aléatoire dans $\mathbb{Z}_N$ et ne révèle aucune information sur a ou $\lambda(N)$. En effet, si β est aléatoire dans $\mathbb{Z}^*_N$, alors $\beta\lambda(N) \bmod N$ est aussi un entier aléatoire dans $\mathbb{Z}^*_N$. Ainsi, si β est secret, aucune information sur $\lambda(N)$ ne sera dévoilée. De plus, ce choix de $\mathsf{sk} = \beta\lambda(N)$ permet de déchiffrer de la même façon que si on avait posé $\mathsf{sk} = \lambda(N)$ car dans $\mathbb{Z}^*_{N^2}$, le groupe des carrés est d'ordre mN et non m comme dans le cas RSA que nous avons étudié au chapitre 3.

Enfin, comme cette information peut être connue par tous les serveurs, elle peut aussi être publique, ce qui évite que les serveurs la recalculent à chaque déchiffrement.

Rappellons que $\Delta = n!$ où n est le nombre de serveurs.

4.3.3 Algorithme de génération des clés

- Choisir un entier N, produit de deux nombres premiers sûrs p et q, tels que $p = 2p' + 1$ et $q = 2q' + 1$ et $\mathrm{pgcd}(N, \varphi(N)) = 1$. Soit $m = p'q'$ et β un élément choisi au hasard dans $\mathbb{Z}^*_N$.

- Choisir ensuite au hasard $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}^*_N$ et fixer $g = (1+N)^a \times b^N \bmod N^2$.

- La clé secrète $\mathsf{sk} = \beta \times m$ est partagée avec le schéma de Shamir : soit $a_0 = \beta m$, choisir au hasard t valeurs a_k dans $\{0, \ldots N \times m - 1\}$ et calculer $f(X) = \sum_{k=0}^{t} a_k X^k$. La part sk_i du i-ième serveur P_i est $f(i) \bmod Nm$.

- La clé publique pk consiste en g, N et la valeur $\theta = L(g^{m\beta}) = am\beta \bmod N$.

- Soit $\mathsf{vk} = v$ un générateur du groupe des carrés dans $\mathbb{Z}^*_{N^2}$. Les clés de vérification vk_i sont obtenues avec la formule $\mathsf{vk}_i = v^{\Delta \mathsf{sk}_i} \bmod N^2$.

Algorithme de chiffrement

Pour chiffrer un message M, choisir au hasard $x \in \mathbb{Z}_N^*$ et calculer $c = g^M x^N \bmod N^2$.

Algorithme de déchiffrement

Le i-ième joueur P_i calcule la part de déchiffrement $\sigma_i = c^{2\Delta \mathsf{sk}_i} \bmod N^2$ en utilisant sa part secrète sk_i. Il calcule une preuve de déchiffrement correct qui garantit que $c^{4\Delta} \bmod N^2$ et $v^{\Delta} \bmod N^2$ ont été élevés à la même puissance sk_i afin d'obtenir σ_i^2 et vk_i (cf. le protocole de la section 3.2.4).

Algorithme de combinaison

Si moins de $t+1$ parts de déchiffrement sont valides, l'algorithme échoue. Sinon, soit S un ensemble des $t+1$ parts valides. Le combineur peut calculer le clair

$$M = L\left(\prod_{j \in S} \sigma_j^{2\lambda_{0,j}^S} \bmod N^2\right) \times \frac{1}{4\Delta^2\theta} \bmod N$$

où $\lambda_{0,j}^S = \Delta \times \prod_{j' \in S \setminus \{j\}} \frac{j'}{j'-j} \in \mathbb{Z}$

Remarques sur l'exactitude du schéma

1. Notons tout d'abord que l'on choisit N tel que $\mathrm{pgcd}(N, \varphi(N)) = 1$. Cette condition garantit que la fonction définie par $f(a,b) = (1+N)^a \times b^N \bmod N^2$ est une bijection de $\mathbb{Z}_N \times \mathbb{Z}_N^*$ dans $\mathbb{Z}_{N^2}^*$.
2. L'ordre de $g = (1+N)b^N$ dans $\mathbb{Z}_{N^2}^*$ est $N \times \alpha$ où α est l'ordre de b dans $\mathbb{Z}_N^*$.
3. De plus, on peut voir que le sous-groupe des carrés dans $\mathbb{Z}_{N^2}^*$ est cyclique et d'ordre Nm. Le nombre de générateurs de ce groupe est $\varphi(Nm)$ et est tel que la probabilité pour un carré choisi aléatoirement dans $\mathbb{Z}_{N^2}^*$ d'être un générateur est environ $1 - 1/\sqrt{N}$. Cette probabilité est écrasante.
4. Ensuite, les clés de vérifications vk_i peuvent être vues comme des témoins de la connaissance du logarithme discret de vk_i en base $v^{\Delta} \bmod N^2$. Elles sont utilisées pour construire les preuves de validité des parts de déchiffrement.

Finalement, on considère un sous-ensemble S de $t+1$ parts correctes; le calcul de $c^{4\Delta^2 m\beta}$ peut être effectué en utilisant la formule d'interpolation de Lagrange:

$$\Delta f(0) = \Delta m\beta = \sum_{j \in S} \lambda_{0,j}^S f(j) \bmod Nm$$

donc

$$c^{4\Delta^2 m\beta} = \prod_{j \in S} c^{4\Delta s_j \lambda_{0,j}^S} = \prod_{j \in S} \sigma_j^{2\lambda_{0,j}^S} \bmod N^2$$

Si c est le chiffrement d'un message M,

$$c^{4\Delta^2 m\beta} = g^{4\Delta^2 M\beta m} = (1+n)^{4\Delta^2 Mam\beta} = 1 + 4\Delta^2 Mam\beta N \bmod N^2$$

Par conséquent, $L\left(\prod_{j \in S} \sigma_j^{2\lambda_{0,j}^S} \bmod N^2\right) = 4M\Delta^2 am\beta = M \times 4\Delta^2\theta \bmod N$. Comme θ est une part de la clé publique, on obtient le message M.

4.3.4 Preuve de sécurité

On prouve maintenant que la version à seuil du schéma de Paillier est sûre selon les définitions de la sécurité sémantique proposées en section 4.3.1. Une telle preuve montre que si un adversaire est capable de casser la sécurité sémantique à seuil, il peut être utilisé pour casser la sécurité sémantique du schéma d'origine.

De manière plus précise, comme on l'a dit précédemment en section 4.3.1, nous devons simuler les informations qu'un adversaire peut obtenir aux étapes A2, A3, A4, A5 et A6. A l'étape A2, nous simulons les paramètres du cryptosystème à seuil. Dans les étapes A3 et A5, nous simulons les parts de déchiffrement des serveurs non corrompus avec les preuves de déchiffrement correct. Finalement, les étapes A4 et A6 représentent les deux étapes de la définition de sécurité sémantique standard pour les cryptosystèmes non partagés.

Théorème 17. Sous l'hypothèse de résiduosité composite et dans le modèle de l'oracle aléatoire, la version à seuil du cryptosystème de Paillier est sémantiquement sûre contre des adversaires actifs et non-adaptatifs.

Preuve: Faisons l'hypothèse d'un adversaire $\mathcal{A}$ capable de casser la sécurité sémantique du schéma partagé. On décrit maintenant un attaquant qui utilise l'adversaire $\mathcal{A}$ pour casser la sécurité sémantique du schéma originel de Paillier. Dans une première phase, appelée la phase "find", l'attaquant obtient d'un challengeur la clé publique (N, g) et il choisit deux messages M_0 et M_1 qui sont envoyés à un oracle de chiffrement. Cet oracle choisit aléatoirement un bit b et retourne un chiffrement c de M_b. Dans une seconde phase, appelée phase "guess", l'attaquant essaie de deviner quel message a été chiffré.

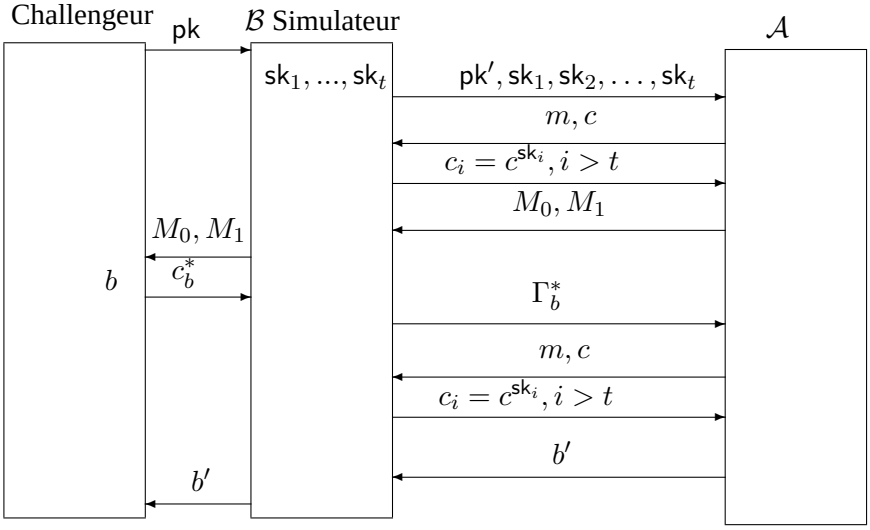


FIG. 4.1: *Preuve de sécurité pour Paillier IND-TCPA.*

Décrivons maintenant comment alimenter un adversaire $\mathcal{A}$ pour le schéma à seuil afin de construire un attaquant contre la sécurité sémantique. Dans l'étape A1 du jeu A, l'adversaire choisit de corrompre t serveurs. Sans perte de généralité, on fait l'hypothèse que les t premiers serveurs $P_1, \ldots P_t$ sont corrompus.

Dans la phase "find", l'attaquant obtient la clé publique $\mathsf{pk} = (N, g)$ du schéma régulier de Paillier. Il choisit au hasard $(a_1, b_1, \theta) \in \mathbb{Z}_N \times \mathbb{Z}_N^* \times \mathbb{Z}_N^*$ et calcule $g_1 = g^{a_1} \times b_1^N \bmod N^2$. Il choisit aussi au

hasard t valeurs $\mathsf{sk}_1, \ldots \mathsf{sk}_t$ dans l'intervalle $\{0, \ldots \lfloor N^2/4 \rfloor\}$, un élément α au hasard dans $\mathbb{Z}_N^*$ et calcule $v = g_1^{2\alpha} \bmod N^2$.

Ensuite, il calcule $\mathsf{vk}_i = v^{\Delta s_i} \bmod N^2$, pour $i = 1, \ldots t$, et les autres clés de vérifications

$$\mathsf{vk}_i = (1 + 2\alpha\theta N)^{\lambda_{i,0}^S} \times \prod_{j \in S \setminus \{0\}} v^{\mathsf{sk}_j \lambda_{i,j}^S} \bmod N^2$$

où $S = \{0, 1, \ldots t\}$. L'attaquant envoie $(N, g_1, \theta, v, \mathsf{vk}_1, \ldots \mathsf{vk}_n, \mathsf{sk}_1, \ldots \mathsf{sk}_t)$ à $\mathcal{A}$ à l'étape A2 du jeu A. La nouvelle clé publique est notée pk' dans la figure 4.1 et comprend l'ensemble des clés de vérification vk_i.

Durant l'étape A3, $\mathcal{A}$ choisit un message M et l'envoie à l'attaquant. Il calcule $c = g_1^M x^N \bmod N^2$, un chiffrement valide de M. Les parts de déchiffrement des joueurs corrompus sont calculées correctement en utilisant les sk_i : $\sigma_i = c^{2\Delta \mathsf{sk}_i} \bmod N^2$, pour $i = 1, \ldots t$. Les autres parts sont obtenues par interpolation de la manière suivante :

$$\sigma_i = (1 + 2M\theta N)^{\lambda_{i,0}^S} \times \prod_{j \in S \setminus \{0\}} c^{2\mathsf{sk}_i \lambda_{i,j}^S} \bmod N^2$$

Finalement, l'attaquant fabrique une preuve preuve_i. La simulation d'une telle preuve a été vue dans la section 3.2.4. Il retourne $(c, \sigma_1, \ldots \sigma_n, \mathsf{preuve}_1, \ldots \mathsf{preuve}_n)$.

Dans l'étape A4, $\mathcal{A}$ retourne deux messages M_0 et M_1. L'attaquant retourne au challengeur ces deux messages comme le résultat de la phase "find".

Ensuite, le challengeur pour le schéma non distribué de Paillier choisit un bit aléatoire et envoie un chiffré c_b^* de M_b à l'attaquant. Ce dernier randomise le chiffré $\Gamma_b^* = c_b^{a_1} \bmod N^2$ avec a_1 et envoie Γ_b^* à l'adversaire $\mathcal{A}$.

L'étape A5 est similaire à l'étape A3. Finalement, à l'étape A6, $\mathcal{A}$ répond un bit b' qui est renvoyé par l'adversaire dans la phase "guess".

On prouve maintenant que les données simulées par l'attaquant ne peuvent pas être distinguées des données réelles par $\mathcal{A}$. Par conséquent, s'il existe un adversaire $\mathcal{A}$ fonctionnant en temps polynomial capable de casser la sécurité sémantique du schéma à seuil, on peut construire un attaquant capable de casser la sécurité sémantique du schéma de Paillier originel.

Indistinguabilité des données reçues par $\mathcal{A}$ durant l'étape A2.
On observe tout d'abord que $g_1 = g^{a_1} b_1^N \bmod N^2$ est uniformément distribué dans l'ensemble des éléments d'ordre un multiple de N, si l'ordre de g est aussi un multiple de N. Une telle modification sur g est nécessaire. En effet, on choisit v comme une puissance paire de g_1 et nous voulons que v génère le sous-groupe des carrés modulo N^2. Par conséquent, g doit être randomisé afin d'obtenir avec forte probabilité une base d'un ordre très grand. Par exemple, la base valide $g = 1 + N$ ne fournirait jamais un v correct.

Nous notons aussi que θ et v sont uniformément distribués respectivement dans $\mathbb{Z}_N^*$ et dans Q_{N^2}, l'ensemble des carrés modulo N^2. De plus, v est un générateur de Q_{N^2} avec probabilité écrasante : la distance statistique entre la distribution uniforme sur le sous-ensemble des générateurs Q_{N^2}, d'ordre $\varphi(Nm)$, et la distribution uniforme sur Q_{N^2}, d'ordre Nm, est $O(N^{-1/2})$.

Ensuite, l'attaquant choisit les clés secrètes $\mathsf{sk}_1, \ldots \mathsf{sk}_t$ des joueurs corrompus ; les sk_i doivent être dans l'intervalle $[0, Nm[$ mais, comme m est inconnu, nous prenons sk_i dans $[0, \lfloor N^2/4 \rfloor[$. Néanmoins, la distance statistique entre la distribution uniforme sur $[0, \lfloor N^2/4 \rfloor[$ et la distribution uniforme sur $[0, Nm[$ est $O(N^{-1/2})$ et par conséquent l'adversaire ne peut pas distinguer les clés secrètes réelles des clés simulées.

Quand le distributeur distribue correctement les parts, les deux conditions suivantes sont vérifiées :

– Pour tout ensemble S de taille $t+1$ et pour tout $i \notin S$, $\mathsf{vk}_i{}^{\Delta} = \prod_{j \in S} \mathsf{vk}_j{}^{\lambda_{i,j}^S} \bmod N^2$

– Pour tout S de taille $t+1$, $\prod_{j \in S} \mathsf{vk}_j{}^{\lambda_{0,j}^S} \bmod N^2 \in \{u < N^2 | u = 1 \bmod N\}$

Dans la simulation, on choisit $v^{m\beta} = 1 + 2\alpha\theta N \bmod N^2$ sans connaître m mais juste en choisissant au hasard θ. Les clés de vérification des serveurs corrompus sont calculées en utilisant les clés secrètes connues sk_i et les clés de vérification manquantes vk_i sont obtenues par la formule d'interpolation de Lagrange. Bien sûr, nous ne sommes pas capables de trouver les clés secrètes manquantes mais en fait nous n'en avons pas besoin. Ainsi, la distribution reçue par $\mathcal{A}$ durant l'étape de génération des clés est indistinguable de la distribution réelle.

Indistinguabilité des données reçues par $\mathcal{A}$ durant les étapes A3 et A5.
Aux étapes A3 et A5, le chiffrement du message M est calculé par $c = g_1^M x^N \bmod N^2$. Ensuite, les parts $\sigma_1, \ldots \sigma_t$ des serveurs corrompus sont calculées en utilisant les clés secrètes $\mathsf{sk}_1, \ldots \mathsf{sk}_t$ de la manière suivante $\sigma_i = c^{2\Delta \mathsf{sk}_i} \bmod N^2$. Enfin, les σ_i manquantes sont obtenues par interpolation, comme les vk_i, en utilisant les $\sigma_1, \ldots \sigma_t$ et le $(t+1)$-ième point $c^{m\beta} \bmod N^2$ que nous pouvons calculer sans aucun secret supplémentaire car il est égal à $1 + 2M\theta N$. Finalement, dans la preuve de la simulation, la distribution produite par l'attaquant est statistiquement proche d'une simulation parfaite comme il est rappelé en section 3.2.4. Cette simulation est apparue auparavant dans [174]. Dans le modèle de l'oracle aléatoire où l'attaquant a un contrôle total des valeurs retournées par la fonction de hachage H, on définit la valeur de H au point $(v, c^{4\Delta}, \mathsf{vk}_i, \sigma_i^2, v^y/\mathsf{vk}_i{}^e, c^{4\Delta^2 y}/\sigma_i^{2e})$ comme valant e. Avec probabilité écrasante, l'attaquant n'a pas encore défini l'oracle aléatoire en ce point et l'adversaire $\mathcal{A}$ ne peut pas détecter la fraude. □

4.3.5 Partage complet du déchiffrement de Paillier

On observe que pour partager complètement le déchiffrement précédent, nous devons avoir un algorithme distribué génération de clés. Pour distribuer la génération du module RSA, nous pouvons employer les techniques du chapitre 3. Ensuite, les preuves de validité des parts de déchiffrement sont identiques à celles pour RSA afin de tenir compte du fait que N n'est plus le produit de deux nombres premiers sûrs.

Par rapport à RSA où la clé secrète est l'inverse de e modulo $\varphi(N)$, dans le cryptosystème partagé de Paillier, la clé secrète est $\mathsf{sk} = \beta m$ où $m = \frac{p-1}{2} \times \frac{q-1}{2}$ et β un nombre aléatoire dans $\mathbb{Z}_N^*$ tel que 2 divise β. La clé publique est composée de $\mathsf{pk} = (g, N, \mathsf{vk}, \mathsf{vk}_{u,i}, \theta = a\beta m \bmod N)$ pour $1 \leq i \leq n$, où n est le nombre de serveurs et $1 \leq u \leq k$, où k est un paramètre de sécurité du même ordre que dans le chapitre 3. Les générations partagées de $g, \mathsf{vk}, \mathsf{vk}_{u,i}$ ne posent pas de problème. Étudions le cas de la génération de sk et de θ.

Génération de la clé secrète

On observe qu'à la fin du protocole de génération de clé RSA, les serveurs ont un partage de $\varphi(N)$ car ils doivent calculer $\mathrm{pgcd}(N-1, \varphi(N))$. Cependant, obtenir un partage de $\lambda(N) = \varphi(N)/2$ ou de $m = \varphi(N)/4$ n'est pas aisé si aucune condition particulière n'est fixée sur le choix des p_i et q_i. La division de $\sum_{i=1}^n p_i$ et de $\sum_{i=1}^n q_i$ par 2 ou 4 pose problème si les p_i et q_i sont impairs. Comment savoir qui doit poser $\lfloor p_i/2 \rfloor$ et $\lceil p_j/2 \rceil$ ou $\lceil p_i/2 \rceil$ et $\lfloor p_j/2 \rfloor$ entre les participants P_i et P_j ?

Si 4 divise p_i et q_i pour tous les serveurs sauf pour $i = 1$, où $p_1 \equiv q_1 \equiv 3 \bmod 4$, on obtient le partage suivant de m :

$$m = \frac{\varphi(N)}{4} = \frac{N - p_1 - q_1 + 1}{4} - \sum_{i=2}^{n} \frac{p_i + q_i}{4}$$

car $\text{pgcd}(p-1, q-1) = 1$ et $p \equiv q \equiv 3 \bmod 4$. La part du premier serveur est $\frac{N-p_1-q_1+1}{4}$ qui est un entier car $p_1 + q_1 \equiv 2 \bmod 4$ et $N + 1 \equiv 2 \bmod 4$. La part des autres serveurs est clairement un entier et donc, chaque serveur peut calculer sa part de m sans protocole supplémentaire à partir du partage de $\varphi(N)$. De manière identique, on peut générer un partage de $\lambda(N)$. Pour assurer la robustesse de ce partage, les serveurs doivent être sûrs que P_1 et tous les autres serveurs ont correctement choisi leurs p_i et q_i. Pour ce faire, P_1 fait une preuve qu'il connaît le logarithme discret de g^{p_1-3} et de g^{q_1-3} en base g^4 et tous les autres, qu'ils connaissent le logarithme discret de g^{p_i} et g^{q_i} en base g^4. Ces preuves sont des preuves *à la Schnorr* pour g un générateur d'un sous-groupe d'ordre Q de $\mathbb{Z}_P^*$ où P est un nombre de la forme $P = 2Q + 1$ avec P et Q premiers et $Q \geq 2^{\frac{L(N)}{2}+1}$ où $L(N)$ représente la taille de N en bits.

Génération de θ

Les serveurs doivent encore générer un nombre aléatoire β et calculer $\theta = am\beta \bmod N$. Pour ce faire, les serveurs génèrent g aléatoirement dans $\mathbb{Z}_{N^2}^*$. Puis, chaque serveur tire aléatoirement un nombre, α_i dans l'intervalle $[0, N[$. Les serveurs calculent ensuite $Y = g^{\beta} u^N \bmod N^2$ en utilisant par exemple le protocole que l'on décrira dans le chapitre 6. Dans ce chapitre, nous décrivons un protocole pour générer une clé Diffie-Hellman de manière partagée, alors qu'ici nous voulons générer un chiffré de Paillier Y. Ainsi, au lieu de calculer une clé publique Diffie-Hellman, $g^{\alpha_i} \bmod p$, chaque serveur génère un chiffrement de α_i. A la fin, du protocole de génération de clé Diffie-Hellman, les serveurs ont engendré un chiffré de Paillier de la "clé secrète" β. Ensuite, les serveurs calculent de manière partagée $Y^{\lambda(N)} = g^{\beta\lambda(N)} u^{N\lambda(N)} \bmod N^2$ et obtiennent $a\beta\lambda(N) \bmod N$. Enfin, ils en déduisent $a\beta m \bmod N$ car $\lambda(N) = 2m$. Il reste que chaque serveur doit faire un partage de Feldman dans $\mathbb{Z}_{N^2}^*$ et non dans $\mathbb{Z}_p^*$ (cf. chapitre 2).

Chaque serveur effectue un partage *à la Shamir* de sa part α_i en choisissant un polynôme aléatoire f_i tel que $f_i(0) = \alpha_i$ et t coefficients $a_{i,k}$ au hasard dans $\mathbb{Z}_N$ et pose

$$f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k \in \mathbb{Z}_N$$

La part du joueur j est alors définie comme la valeur du polynôme f_i en j, soit $f_i(j) \bmod N$. Puis, il tire aléatoirement $u_0, u_1, \ldots, u_t$, et calcule pour chaque coefficient, $A_{i,k} = g^{a_{i,k}} u_k \bmod N^2$. Il broadcaste ces valeurs sur un canal public ainsi que $f_i(j)$ sous forme chiffrée au j-ème serveur en utilisant les clés publiques de Paillier (g_j, N_j) et (g, N). Pour ce faire, il calcule $u'_j = \prod_{k=0}^{t} u_k^{j^k} \bmod N$ et $c'_{i,j} = g^{f_i(j)} u'^{N}_j \bmod N^2$, tire aléatoirement $u''_j \in \mathbb{Z}_{N_j}^*$ et calcule $c''_{i,j} = g_j^{f_i(j)} u''^{N_j}_j \bmod N_j^2$. Enfin, le serveur i fait une preuve publiquement vérifiable que $\mathcal{D}_{\mathsf{sk}}(c'_{i,j}) = \mathcal{D}_{\mathsf{sk}_j}(c''_{i,j})$. Un tel exemple de preuve est donné dans le chapitre 5.

Tous les serveurs peuvent vérifier la part $f_i(j) = \alpha_{i,j}$ de α_i en calculant

$$c'_{i,j} \stackrel{?}{=} \prod_{k=0}^{t} A_{i,k}^{j^k} = g^{f_i(j)} \times \left[\prod_{k=0}^{t} u_k^{j^k}\right]^N = g^{\alpha_{i,j}} \times [u'_j]^N \bmod N^2$$

Ainsi, nous avons complètement distribué le déchiffrement de Paillier[50].

50. Si nous voulons partager le cryptosystème de Goldwasser et Micali, nous devons pouvoir décider si un nombre x est un

4.4 Seconde proposition

Damgård et Jurik ont ensuite proposé une autre solution pour partager le cryptosystème de Paillier [56]. Tout d'abord, ils ont remarqué que l'on peut prendre pour g, $1+N$, ce qui permet d'éviter la première exponentiation car $g^M = (1+N)^M = 1 + MN \bmod N^2$ et ainsi une multiplication suffit.

Ensuite, pour déchiffrer, au lieu de prendre $\lambda(N)$ comme clé secrète, (ou βm comme nous l'avons fait précédemment) ils choisissent $d \in \mathbb{Z}^*_{N^2}$ telle que $d = 0 \bmod \lambda(N)$ et $d = 1 \bmod N$. Avec une telle clé, on montre que la formule pour déchiffrer se simplifie en

$$c^d = ((1+N)^M r^N)^d = (1+N)^{Md \bmod N} \times (r^N)^{d \bmod \lambda(N)} = (1+N)^M \bmod N^2 = 1 + MN$$

car pour tout $x \in \mathbb{Z}^*_{N^2}$, $x^{N\lambda(N)} = 1 \bmod N^2$. Ce faisant, le déchiffrement de ce schéma peut être facilement partagé en utilisant [174] car la division par $L(g^{\lambda(N)} \bmod N^2)$ dans la formule de déchiffrement 4.2 est éliminée.

4.5 Conclusion

Dans ce chapitre, nous avons partagé le déchiffrement du système de chiffrement de Paillier et nous avons prouvé la sécurité sémantique contre des attaques à clairs choisis (passives) dans le modèle distribué. De plus, nous avons complètement distribué le déchiffrement de Paillier.

Dans le chapitre suivant, nous montrerons que ce schéma peut être distribué avec un nouvel algorithme de chiffrement pour résister aux attaques à chiffrés choisis. En effet, il est impossible de résister à ce genre d'adversaire avec le cryptosystème de Paillier initial car ce système est seulement sémantiquement sûr contre des attaques à clairs choisis dans le modèle "centralisé". Enfin, on peut mentionner qu'il existe déjà un système de chiffrement construit autour de Paillier sûr contre les attaques à chiffrés choisis [140]. Cependant, il est difficile de partager le déchiffrement de ce système pour les raisons que nous expliquerons dans le chapitre suivant.

résidu quadratique modulo $N = pq$ sans connaître p. Calculer le symbole de Legendre de x modulo p demande de savoir calculer une exponentielle modulo p et p doit être partagé. Comme nous ne savons pas faire de calcul modulo un nombre partagé, nous devons utiliser une autre méthode. Si $N = pq$, pour savoir si $x \stackrel{?}{\in} Q_N$, on commence par calculer son symbole de Jacobi. Si $(x|N) = -1$, x n'appartient pas à Q_N. Sinon, x peut être un carré modulo N ou un pseudo-carré, c'est-à-dire un nombre tel que $(x|p) = (x|q) = -1$. Pour distinguer entre ces deux cas, on peut calculer $x^{\frac{p-1}{2}\frac{q-1}{2}} \bmod N$. Si cette quantité vaut 1, x est un carré, sinon, si on obtient -1, x est un pseudo-carré. Ceci peut servir pour distribuer le cryptosystème de Goldwasser-Micali avec la même méthode que précédemment.

5

Cryptosystèmes partagés sûrs contre les attaques à chiffrés choisis

Dans ce chapitre, nous décrivons l'article [74] qui sera présenté avec David Pointcheval à la conférence Asiacrypt '01. Dans cet article nous avons réhabilité le modèle du double-chiffrement proposé par Naor et Yung pour proposer un schéma de conversion générique pour tout cryptosystème sémantiquement sûr contre les attaques passives en cryptosystème à seuil sémantiquement sûr contre les attaques adaptatives à chiffrés choisis. En particulier, nous proposons une version du cryptosystème de Paillier sûr dans ce modèle, ce qui fournit le premier schéma de ce type basé sur la factorisation. De plus, nous revisitons la notion de sécurité dans le cas distribué.

Sommaire

5.1 Introduction

5.1.1 Sécurité contre les attaques à chiffrés choisis

La sécurité sémantique contre les attaques adaptatives à chiffrés choisis représente la définition de sécurité correcte pour un système de chiffrement [98, 160, 5]. Par conséquent, un grand nombre de travaux [81, 83, 82, 147, 134] ont récemment proposé des schémas pour convertir toute fonction à sens unique en un système de chiffrement sûr selon cette notion de sécurité.

Avant cette notion, Naor et Yung dans [129] ont proposé une notion de sécurité plus faible, appelée attaque de midi (Lunch-time attack), ou encore attaque indifférente à chiffrés choisis ou non-adaptative. L'adversaire ne peut demander des déchiffrements qu'avant de recevoir le chiffré cible. Naor et Yung [129] ont présenté une conversion pour sécuriser des schémas contre des attaques à chiffrés choisis dans le scénario lunch-time. Ils ont utilisé des systèmes de preuves non-interactive zero-knowledge (preuve d'appartenance à un langage [23, 22]) pour montrer la consistance d'un chiffré, et non pour prouver que la personne qui a construit le chiffré "connaît nécessairement son déchiffrement".

Ensuite Rackoff et Simon [160] ont rafiné cette construction en remplaçant les preuves non-interactive d'appartenance à un langage par des preuves zero-knowledge de connaissance. Par conséquent, à tout chiffré est ajoutée une preuve non-interactive de connaissance du clair. Ceci conduit à des systèmes de chiffrement sûrs contre des attaques adaptatives à chiffrés choisis. En effet, l'émetteur prouve qu'il connaît le clair et par conséquent, les attaques CCA ne peuvent être que des attaques CPA.

Une notion similaire dite "plaintext-awareness" [10, 5] a été ensuite définie. Un schéma possède cette propriété si pour produire un chiffré valide, la connaissance du clair correspondant est nécessaire. Pour prouver cette propriété, il faut construire un "plaintext-extractor" qui est un algorithme permettant de déchiffrer en ayant accès à un oracle. Si nous ne sommes pas dans un modèle de sécurité à oracle, et que nous sommes dans le modèle standard, l'existence de cette machine de Turing revient à savoir déchiffrer sans avoir la clé secrète. Ainsi, cette dernière notion ne prend son sens que dans un modèle à oracle comme celui de l'oracle aléatoire [9][51]. Par exemple, le cryptosystème Cramer-Shoup [55] prouvé sûr IND-CCA dans le modèle standard ne vérifie pas cette propriété de "plaintext-awareness".

Pendant de nombreuses années, plusieurs schémas ont été proposés pour atteindre la notion de sécurité IND-CCA. La plupart d'entre eux a été prouvée dans le modèle de l'oracle aléatoire [10, 84, 175, 140, 81, 82, 83, 147, 134] en utilisant la propriété de plaintext-awareness. Seul le cryptosystème Cramer-Shoup [55] a été prouvé IND-CCA dans le modèle standard.

5.1.2 Problématique du partage de cryptosystèmes IND-CCA

Pour distribuer les processus de déchiffrement, des techniques similaires à celles pour signer un message peuvent être utilisées pour éviter les attaques à clairs choisis par des adversaires passifs. Cependant, ces méthodes ne permettent pas d'éviter les attaques à chiffrés choisis. En effet, en général, les serveurs ne peuvent pas commencer le déchiffrement sans savoir si le chiffré est valide ou non parce qu'un attaquant peut se faire passer pour un des serveurs et, dans le cas de chiffrés invalides, il peut alors apprendre de l'information.

Par conséquent, quand on cherche à partager un système de chiffrement, on ne peut pas attendre la fin du déchiffrement pour savoir si les serveurs peuvent réellement déchiffrer ou non. Ainsi, nous devons intégrer des preuves de validité du chiffré qui soient publiquement vérifiables. Malheureusement, la plupart des systèmes de chiffrement connus sûrs contre des attaques à chiffrés choisis ne sont pas facilement partageables. En effet, dans les processus de déchiffrement, les clairs *supposés* sont déchiffrés,

51. Le modèle de l'oracle aléatoire n'est pas le seul modèle à oracle. Le modèle générique [173] est aussi un modèle à oracle.

et ensuite une redondance est vérifiée juste avant de retourner le clair. Comme la redondance met en jeu une fonction de hachage, la vérification finale ne peut pas être distribuée de manière efficace.

5.1.3 Solutions existantes

Il existe deux méthodes pour distribuer le processus de déchiffrement d'un système de chiffrement. Alors que le premier utilise de l'aléa, le second suit le modèle décrit par Lee et Lim dans [121] dans lequel pour rendre un système de chiffrement sûr contre les attaques CCA le processus de déchiffrement initial est renversé : le destinataire commence par vérifier si le chiffré est valide avant de déchiffrer.

Preuve de validité vérifiable uniquement par le possesseur de la clé secrète

La *première méthode* a été proposée par Canetti et Goldwasser dans [37]. Dans le système de chiffrement Cramer-Shoup [55], le destinataire peut vérifier la validité d'un chiffré en utilisant une partie de la clé secrète, avant de déchiffrer le chiffré valide en utilisant la deuxième partie de la clé secrète. Par conséquent, on peut penser qu'il est facile de partager ce système de chiffrement. Mais au lieu de vérifier la validité du chiffré dans un premier tour et de déchiffrer selon la validité, Canetti et Goldwasser [37] ont proposé une nouvelle stratégie avec seulement un tour. Les serveurs déchiffrent tout chiffré soumis et le processus de déchiffrement est randomisé. Les serveurs calculent $m(v'/v)^s$ où s est un aléa partagé entre les serveurs (une partie de la clé secrète), v la preuve contenue dans le chiffré, et v' la preuve calculée par les serveurs. Dans la version centralisée, le processus de déchiffrement vérifie $v \stackrel{?}{=} v'$. Dans la version distribuée, si la preuve est correcte, $(v/v')^s = 1$ et le déchiffrement donne m, sinon il retourne une valeur aléatoire. Personne ne sait alors si le message déchiffré est correct ou non, s'il n'y a pas de redondance dans le clair m. Une solution est de déchiffrer deux fois le même chiffré. Si les résultats sont identiques, le message était bien formé. Le principal inconvénient est que les serveurs doivent conserver dans la clé secrète un partage de l'aléa s et ainsi, la longueur de la clé est linéaire en le nombre de messages déchiffrés. Par conséquent, même si la méthode basique en deux tours apparaît plus lente, *vérifier la preuve et déchiffrer*, elle possède des caractéristiques intéressantes en termes de stockage de la clé et évite l'utilisation d'un protocole de calcul d'un aléa partagé.

Preuve de validité universellement vérifiable

La première méthode de Canetti et Goldwasser est malheureusement spécifique au système de chiffrement de Cramer et de Shoup. La méthode qui commence par vérifier la preuve et déchiffrer ensuite est applicable à d'autres cryptosystèmes mais aucun autre n'a ce type de preuve. La *seconde méthode*, utilisée par Shoup et Gennaro [175], suit l'article de Lee et Lim [121], avec le système de chiffrement El Gamal [66], mais dans le modèle de l'oracle aléatoire [9]. Initialement, ils ont tenté d'ajouter une preuve non-interactive de connaissance d'un logarithme discret, en utilisant la signature de Schnorr [166]. Mais, ils ont remarqué que la simulation du déchiffrement sans la clé secrète nécessitait un temps exponentiel dû à l'explosion combinatoire du lemme de bifurcation [150, 151].

En effet, le problème est dans la simulation de l'oracle de déchiffrement. Pour simuler le déchiffrement, il faut extraire de la signature le random r qui a permis de chiffrer. La preuve de sécurité de la signature de Schnorr (cf. chapitre 2) faite par Pointcheval et Stern utilise le forking lemma qui permet de rembobiner la machine et fournir au signataire un deuxième challenge comme résultat de la fonction de hachage. Les deux équations de vérification obtenues à partir des deux challenges différents permettent d'extraire le random r. Soit par exemple un adversaire qui choisit ℓ randoms r_i et des messages clairs m_i de façon à ce qu'ils dépendent tous les deux des challenges e_i précédents, $r_i = m_i = H(e_1, \ldots, e_{i-1})$ fournis par l'oracle de hachage. Puis, l'adversaire demande à l'oracle de déchiffrement, le déchiffré des

chiffrés $c_\ell, \ldots, c_1$. L'adversaire fait alors une première requête de déchiffrement, c_ℓ. Le simulateur doit donc rembobiner l'adversaire jusqu'au point où il a obtenu le challenge e_ℓ de l'oracle H et envoie à l'adversaire un challenge différent e'_ℓ. On continue et quand il présente un chiffré c'_ℓ correspondant, on peut extraire r_ℓ et déchiffrer c_ℓ. Maintenant, l'adversaire continue et fait une requête de déchiffrement pour $c_{\ell-1}$. On rembobine l'adversaire jusqu'au point où il a obtenu la réponse $e_{\ell-1}$ de l'oracle H et on envoie un challenge $e'_{\ell-1}$ différent. Si l'on poursuit, avant que l'algorithme de l'adversaire émette la requête $c'_{\ell-1}$, il va demander de déchiffrer c''_ℓ. Malheureusement, on ne peut pas déchiffrer car en général $r''_\ell \neq r_\ell$ et $m''_\ell \neq m_\ell$ car ils sont tous les deux, (r''_ℓ, m''_ℓ), calculés comme le haché de $e_1, \ldots, e_{\ell-2}, e'_{\ell-1}$ avec $e'_{\ell-1} \neq e_{\ell-1}$. Nous devons donc rembobiner l'adversaire jusqu'au point où il a obtenu e''_ℓ et recommencer pour déchiffrer c''_ℓ. Il est clair qu'un tel adversaire force le simulateur à s'exécuter en temps proportionnel à 2^ℓ.

Cette explosion peut être évitée sous des hypothèses plus fortes [168]. Pour résoudre cette explosion combinatoire, Gennaro et Shoup ont utilisé des preuves non-interactives d'appartenance à un langage (comme dans [129]) pour éviter l'*extraction* (cf. chapitre 2), et ainsi l'explosion combinatoire dans la simulation du déchiffrement. En fait, la simulation du processus de déchiffrement ne peut pas éviter le rembobinage de la machine. Le problème est le même que dans le cadre du zero-knowledge resettable (cf. chapitre 2). Par conséquent, le même genre de techniques de preuve d'appartenance à un langage difficile peut être utilisé [6]. Nous pouvons remarquer que la preuve de connaissance de Rackoff et Simon est en même temps une preuve d'appartenance et une preuve de connaissance. Dans ce système de chiffrement, il y a deux clés comme dans [129] : la première appartient au destinataire, mais la seconde appartient à l'émetteur. Comme le prouveur possède une des deux clés, il peut déchiffrer et obtenir le clair. Par conséquent, la preuve devient une preuve de connaissance pour un émetteur spécifique. L'émetteur peut alors déchiffrer tous les messages et comme la preuve est une preuve d'appartenance, nous pouvons la simuler sans utiliser de technique de rembobinage.

5.1.4 Preuves Zero-Knowledge Non-Interactives

Le modèle proposé par Naor et Yung utilise les preuves non-interactives zero-knowledge d'appartenance à un langage dans le modèle d'un ruban aléatoire commun. Le même message m est chiffré sous deux clés publiques différentes. On obtient donc a_0 et a_1 et l'émetteur ajoute une preuve que le mot (a_0, a_1) appartient au langage $L = \{(a, b) \text{ tq } \mathcal{D}(a_0) = \mathcal{D}(a_1)\}$. En 1990, les preuves non-interactives d'appartenance à un langage [23, 22] supposaient l'existence d'un ruban aléatoire commun entre le prouveur et le vérifieur comme ruban d'aléa. A cause de ce ruban commun, ils ont dû restreindre la puissance du modèle de sécurité aux attaques lunch-time. En effet dans une attaque à chiffrés choisis, l'adversaire pourrait être capable à partir du chiffré cible $(a_0^*, a_1^*, \mathsf{preuve}^*)$ de forger une nouvelle preuve preuve' d'appartenance pour le même couple (a_0^*, a_1^*). Dans la phase "guess", dans le cas d'une attaque adaptative, l'adversaire a le droit de poser la requête de déchiffrement du chiffré suivant : $(a_0^*, a_1^*, \mathsf{preuve}'^*)$. Ainsi, l'oracle de déchiffrement renvoie le clair et l'adversaire pourrait gagner son jeu (one-wayness ou sécurité sémantique par exemple). Mais Naor et Yung n'ont pas pu prouver que la preuve d'appartenance ne pouvait pas être modifiée par l'adversaire s'il ne connaissait pas le clair m. Récemment, cette propriété a été considérée dans [164], mais seulement pour des systèmes de preuve théoriques.

Dans ce chapitre, nous utilisons l'hypothèse idéale du modèle de l'oracle aléatoire [9], qui dit que les fonctions de hachage se comportent comme de véritables fonctions aléatoires. Ceci permet de construire des preuves zero-knowledge non-interactives efficaces sans hypothèse de ruban d'aléa commun, et d'atteindre une forme plus faible de non-malléabilité, mais suffisamment forte pour notre propos, appelée *significativité simulable* [164].

Significativité simulable.

Soit un langage $\mathcal{L}$ et un système de preuve non-interactive zero-knowledge pour $\mathcal{L}$. Pour tout adversaire $\mathcal{A}$, ayant accès à une preuve $p^\star$, pour un mot $x^\star$, en dehors ou dans $\mathcal{L}$, nous considérons sa capacité à forger une nouvelle preuve p, pour un mot en dehors de $\mathcal{L}$. Par conséquent, pour tout adversaire $\mathcal{A}$, nous considérons

$$\mathsf{Succ}^{\mathsf{sim-nizk}}(\mathcal{A}) = \Pr\left[(x,p) \leftarrow \mathcal{A}(Q) \,|\, x \in \bar{\mathcal{L}} \wedge (x,p) \notin Q\right],$$

ayant accès à une liste bornée Q de mots prouvés $(x^\star, p^\star)$, où le mot $w^\star$ est n'importe quel mot (en dehors ou dans le langage $\mathcal{L}$) et $p^\star$ une preuve acceptée pour $w^\star$. Nous notons par $\bar{\mathcal{L}}$ le complément de $\mathcal{L}$, soit l'ensemble des mots en dehors du langage $\mathcal{L}$.

Plus généralement, nous notons $\mathsf{Succ}^{\mathsf{sim-nizk}}(t)$ la probabilité de succès maximale sur tous les adversaires, ayant un temps d'exécution borné par t, pour forger une nouvelle preuve acceptée pour un mot non valide, même après avoir vu un nombre borné de preuves pour des mots (non) valides. Dans notre situation, ce nombre borné sera simplement un.

Ceci est une notion plus forte que la significativité classique pour des preuves zero-knowledge non-interactives, mais est plus faible que la non-malléabilité. En effet, Sahai [164] a montré que la non-malléabilité de preuves non-interactives zero-knowledge impliquait cette notion, qu'il a appelée *significativité simulable*.

Comme nous le verrons dans la suite, dans le modèle de l'oracle aléatoire, nous pouvons faire des preuves efficaces qui atteignent ce niveau de sécurité.

5.1.5 Proposition de schémas IND-CCA à seuil

A PKC '99, Fujisaki et Okamoto [81] ont proposé une conversion générique de n'importe quel système de chiffrement IND-CPA en un schéma IND-CCA, dans le modèle de l'oracle aléatoire [9]. Dans ce chapitre, nous revisitons la technique de double-chiffrement de Naor etYung [129], en fournissant une conversion générique de tout système de chiffrement IND-CPA en un schéma IND-CCA avec une preuve de validité publiquement vérifiable. Cette conversion fournit des systèmes de chiffrement à seuil sûrs contre des attaques à chiffrés choisis. Nous présentons de plus des instanciations pratiques dans le modèle de l'oracle aléatoire qui atteignent le niveau de sécurité IND-CCA contre des adversaires actifs.

5.2 Modèle de sécurité

5.2.1 Hypothèses sur le canal de communication

Nous supposons un groupe de n serveurs probabilistes tous connectés à un medium commun de broadcast, appelé le canal de communication. Il peut être vu comme un canal asynchrone comme Internet.

5.2.2 Classification des adversaires

L'adversaire est calculatoirement borné. De plus, il peut corrompre des serveurs à n'importe quel moment en ayant accès à la mémoire des serveurs corrompus (adversaire passif), et/ou en modifiant leur fonctionnement (adversaire actif). L'adversaire décide quels serveurs il veut corrompre au début du protocole (adversaire statique). Nous supposons aussi que l'adversaire ne corrompt pas plus de t serveurs parmi n au cours du protocole, où $n \geq 2t + 1$.

5.2.3 Systèmes de chiffrement à seuil

Un système de chiffrement à seuil t parmi n comprend les composants suivants :

- Un *algorithme de génération de clés* $\mathcal{K}$ qui prend comme entrées un paramètre de sécurité en notation unaire 1^k, le nombre n de serveurs de déchiffrement, et un paramètre de seuil t; il retourne une *clé publique* pk, une liste $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ de clés privées (qui correspondent à un partage de la clé privée sk) et une liste $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ de clés de vérification.
- Un *algorithme de chiffrement* $\mathcal{E}$ qui prend en entrée la clé publique pk et un clair m, et retourne un chiffré c.
- Plusieurs *algorithmes de déchiffrement* $\mathcal{D}_i$ (pour $1 \leq i \leq n$) qui prennent en entrée la clé publique pk, la clé privée sk_i, un chiffré c, et retourne une *part de déchiffrement* σ_i (qui peut contenir une part de vérification pour atteindre la propriété de robustesse).
- Un *algorithme de combinaison* qui prend en entrée la clé publique pk, un chiffré c, et une liste $\sigma_1, \ldots, \sigma_n$ de parts de déchiffrement (ou au moins $t+1$ parmi elles), avec les clés de vérification $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, et retourne un clair m ou rejette si moins de $t+1$ parts de déchiffrement sont correctes dans le cas d'adversaire actif. Tous les utilisateurs peuvent faire tourner cet algorithme.

5.2.4 Notions de sécurité

Dans cette section, nous définissons le jeu qu'un adversaire joue et tente de gagner afin d'atteindre le but de l'attaque. Les adversaires contre les systèmes de chiffrement à seuil tentent d'attaquer les propriétés suivantes :

- Sécurité de la primitive sous-jacente. Dans le cas de système de chiffrement, ceci signifie la one-wayness, la sécurité sémantique [98], ou la non-malléabilité [65].
- Robustesse. Cette propriété assure que les joueurs corrompus ne sont pas capable d'empêcher les serveurs non-corrompus de déchiffrer des chiffrés. Cette notion est utile seulement en présence d'adversaires actifs. En d'autres termes, elle dit que le service de déchiffrement est toujours disponible même si l'adversaire peut envoyer de mauvaises parts de déchiffrement.

Un utilisateur qui souhaite déchiffrer un chiffré c l'envoie à un serveur spécial, appelé le *combineur*, qui le transmet à tous les serveurs. Les serveurs commencent par vérifier la validité du chiffré, calculent leur part de déchiffrement σ_i et la retournent au combineur. Ce dernier recombine les parts de déchiffrement pour obtenir le clair m et le retourne à l'utilisateur. Si nous voulons résister aux adversaires actifs, le combineur doit pouvoir décider de la validité des parts σ_i de déchiffrement qu'il reçoit. Une manière élégante consiste à utiliser des protocoles de vérifications [79], et des clés de vérification sont alors nécessaires. Le but des protocoles de vérification est de permettre à chaque serveur de prouver aux autres qu'il a effectué correctement sa tâche.

Sécurité sémantique

Dans la suite, nous nous focalisons sur l'objectif de sécurité sémantique [98], noté IND, et nous oublions toutes les autres notions de sécurité (one-wayness et non-malléabilité.) Par conséquent, le jeu à considérer est le suivant :

1. L'algorithme de génération de clés $\mathcal{K}$ est exécuté. L'adversaire reçoit par conséquent la clé publique pk. Avec cette clé publique, l'adversaire a la capacité de chiffrer tout clair de son choix (d'où le nom "attaque à clairs choisis").

2. L'adversaire choisit deux clairs m_0 et m_1. Ils sont envoyés à un "oracle de chiffrement" qui choisit un bit $b \in \{0,1\}$ au hasard, chiffre m_b et donne le chiffré cible c_b à l'adversaire.

3. A la fin du jeu, l'adversaire retourne un bit $b' \in \{0,1\}$. On dit que l'adversaire gagne le jeu si $b' = b$.

La sécurité sémantique contre des attaques à clairs choisis signifie que pour tout adversaire polynomialement borné, $b' = b$ avec probabilité seulement négligeablement plus grande que 1/2.

Attaques à Chiffrés Choisis

Une attaque plus forte est usuellement considérée, attaque à chiffrés choisis [160], durant laquelle l'adversaire a un accès complet à un oracle de déchiffrement $\mathcal{D}_{\mathsf{sk}}$ à qui il peut faire des requêtes de déchiffrement pour tout chiffré de son choix. L'adversaire obtient le clair correspondant ou la réponse "rejet". La seule restriction est qu'il ne peut pas faire une requête de déchiffrement avec chiffré cible c_b.

Sécurité à seuil

Les attaques précédentes sont des attaques classiques dans le cadre des systèmes de chiffrement (non distribués). Même dans le cas distribué, la vue de l'adversaire est la même que s'il n'y avait qu'une seule clé secrète. Cependant, nous devons considérer l'information révélée par des parts de déchiffré comme nous l'avons fait dans le chapitre précédent. Dans ce but, nous donnons à l'adversaire l'accès à un nouvel oracle de déchiffrement $\mathcal{D}_{\mathsf{sk}_i}$, mais avec la restriction qu'il peut seulement lui demander des parts pour des couples valides clair-chiffré. Il obtient alors la part de déchiffrement σ_i. Si la paire n'est pas valide (le chiffré ne correspond pas au clair donné) l'oracle peut ne rien retourner [72]. D'où, la définition de la notion de sécurité de base dans le cadre distribué : IND-TCPA et IND-TCCA respectivement[52].

Suivant le type d'adversaires, on a le comportement suivant pour les serveurs corrompus :

– ils continuent à jouer honnêtement — l'adversaire est dit **passif**. Il a accès à toutes les informations internes de ces serveurs, mais ne peut pas modifier leur comportement.

– ils jouent en envoyant de mauvaises parts de déchiffrement — l'adversaire est alors dit **actif**. Il a accès à toutes les informations internes et peut modifier le comportement, c'est-à-dire les réponses des serveurs ou le fonction de leur programme.

Pour résumer, nous avons plusieurs types mixtes d'attaques : les attaques à clairs choisis (CPA) ou à chiffrés choisis (CCA) avec des adversaires passifs (-Passive) ou actifs (-Active). Selon le moment où l'adversaire peut choisir les serveurs corrompus, nous pouvons considérer les adversaires adaptatifs ou statiques. Les adversaires statiques choisissent les serveurs à corrompre au début du jeu, alors que les adversaires adaptatifs peuvent le faire tout au long du jeu en fonction des messages échangés. Il a été prouvé que les adversaires passifs et adaptatifs sont équivalents aux adversaires passifs et non-adaptatifs si le nombre de serveurs est petit [31].

On peut remarquer que dans le cas particulier où $n = 1$ et $t = 0$, nous sommes dans la situation classique, pour laquelle les adversaires passifs et actifs n'ont plus de sens.

5.3 Conversion générique

Dans cette section, nous revisitons le modèle du double-chiffrement proposé par Naor et Yung [129], tout en supposant que $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ est un système de chiffrement (éventuellement à seuil) qui atteint déjà

52. Le T, *Threshold*, est là pour signifier protocole à seuil en anglais.

la sécurité sémantique contre les attaques à clairs choisis (IND-CPA ou IND-TCPA, dans le cas distribué). Ainsi, nous présentons un nouveau schéma qui évite les attaques CCA (ou TCCA, respectivement) quelque soit le type d'adversaire.

5.3.1 Description

La génération de clés : $\mathbf{K}(1^k)$ exécute deux fois $\mathcal{K}(1^k)$ pour obtenir deux clés publiques $(\mathsf{pk}, \mathsf{pk}')$, qui représentent la nouvelle clé publique $\mathbf{PK}$. De la même manière, on définit le nouvel ensemble de clés secrètes comme $\mathbf{SK} = \{\mathbf{SK}_i\}_{1\leq i\leq n} = \{\mathsf{sk}, \mathsf{sk}'\} = \{\mathsf{sk}_i, \mathsf{sk}'_i\}_{1\leq i\leq n}$ et le nouvel ensemble de clés de vérification $\mathbf{VK} = \{\mathbf{VK}_i\}_{1\leq i\leq n} = \{\mathsf{vk}, \mathsf{vk}'\} = \{\mathsf{vk}_i, \mathsf{vk}'_i\}_{1\leq i\leq n}$.

Chiffrement de m

- le message m est chiffré deux fois sous deux clés pk et pk' différentes, $a_0 = \mathcal{E}_{\mathsf{pk}}(m)$ et $a_1 = \mathcal{E}_{\mathsf{pk}'}(m)$;
- une preuve que les deux chiffrés a_0 et a_1 chiffrent le même clair m est ensuite générée, $c = \mathsf{preuve}[\mathcal{D}_{\mathsf{sk}}(a_0) = \mathcal{D}_{\mathsf{sk}'}(a_1)]$.

Déchiffrement partiel de (a_0, a_1, c)

- le serveur vérifie la validité de la preuve c;
- il calcule les deux parts de déchiffrement des chiffrés a_0 et a_1 (seule une peut être suffisante, mais alors le même choix aléatoire doit être fait par tous les serveurs).

Il est ensuite possible de reconstruire le clair en utilisant l'algorithme de reconstruction.

Avec cette construction générique, il n'est pas évident que la preuve c ne révèle pas plus d'information (comme cela a été remarqué dans [129]), de plus une telle preuve peut rarement être rendue efficace dans le modèle standard. Cependant, le modèle de l'oracle aléatoire permet de rendre efficace des preuves zero-knowledge non-interactives [151].

5.3.2 Preuve Non-Interactive Zero-Knowledge

Dans la suite de ce chapitre, nous avons besoin d'une notion de sécurité forte sur la preuve c et sur le langage

$$\mathcal{L} = \{(\mathsf{pk}, \mathsf{pk}', \mathcal{E}_{\mathsf{pk}}(m), \mathcal{E}_{\mathsf{pk}'}(m)) \,|\, \forall m\},$$

appelée *significativité simulable* [164].

En effet, nous voulons que tout adversaire $\mathcal{A}$, ayant vu une paire $(x^\star, c^\star)$, où $x^\star = (\mathsf{pk}, \mathsf{pk}', \mathcal{E}_{\mathsf{pk}}(m), \mathcal{E}_{\mathsf{pk}'}(m'))$ (avec $m = m'$ mais aussi probablement $m \neq m'$) et $c^\star$ une preuve acceptée pour $x^\star$, ait une probabilité de succès négligeable de forger une nouvelle preuve c pour un mot $x \notin \mathcal{L}$:

$$\mathsf{Succ}^{\mathsf{sim-nizk}}(\mathcal{A}) = \Pr\left[(x, c) \leftarrow \mathcal{A}(x^\star, c^\star) \,|\, x \in \bar{\mathcal{L}} \wedge (x, c) \neq (x^\star, c^\star)\right].$$

L'idée de cette probabilité de succès est que l'adversaire ne doit pas être capable de construire une nouvelle preuve à partir de preuves précédemment vues, excepté pour des mots valides (ce qui veut dire dans $\mathcal{L}$). En effet, on ne peut pas empêcher l'adversaire de construire une preuve acceptée pour un mot correct choisi par lui et dans ce cas, le chiffré est valide.

De plus, l'adversaire a accès à une preuve pour un mot dans $\mathcal{L}$, ou peut-être en dehors de $\mathcal{L}$, parce que le simulateur créera quelques fois une preuve acceptée pour un mot qui n'est pas dans $\mathcal{L}$. Une telle preuve ne devra pas donner des informations à l'adversaire non plus.

La preuve c convainc tout le monde que le chiffré est valide avant le début du déchiffrement. Dans la preuve de sécurité, le simulateur du déchiffrement connaît une clé secrète. Mais le chiffré cible ne sera pas nécessairement valide (éventuellement avec deux messages chiffrés distincts). Grâce au modèle de l'oracle aléatoire, il est toujours possible de simuler, de manière indistinguable, une preuve acceptée même pour un faux mot, sous l'hypothèse de la difficulté du problème de décider l'appartenance (ce qui est une hypothèse plus faible que la sécurité sémantique du cryptosystème sous-jacent).

Enfin, nous présentons des preuves pratiques non-interactives zero-knowledge qui prouvent essentiellement la significativité simulable en utilisant la technique du lemme de bifurcation [151].

5.3.3 Preuve de sécurité

Nous montrons qu'à partir de tout adversaire $\mathcal{A}$ contre un double schéma IND-CCA, il est possible de construire un adversaire $\mathcal{B}$ contre le schéma originel IND-CPA, en considérant seulement les adversaires passifs. On note par $\mathsf{Succ}^{\mathsf{sim-nizk}}(t)$ la probabilité de succès d'un adversaire qui peut générer une preuve zero-knowledge non-interactive correcte sans connaître le clair en temps inférieur à t.

5.3.4 Adversaire passif

Théorème 18. Étant donné un système de chiffrement IND-CPA (ou IND-TCPA) $\mathcal{S}$, la conversion double fournit un système de chiffrement IND-CCA (ou IND-TCCA, resp.) $\mathcal{S}_{tw}$, dans le modèle de l'oracle aléatoire.

Preuve: Notre preuve utilise une réduction. Étant donné un (t, ε)-adversaire $\mathcal{A}$ contre la sécurité de notre schéma $\mathcal{S}_{tw}$ au sens IND-CCA, on construit un (t', ε')-attaquant $\mathcal{B}$ contre la sécurité du schéma $\mathcal{S}$ ("tw" pour "twin") dans le sens IND-CPA où $t' = t$ et $\varepsilon' = \varepsilon/4 - 2\mathsf{Succ}^{\mathsf{sim-nizk}}(t)$.

Avant tout, remarquons que si un système de chiffrement (classique) est IND-CPA, alors si nous chiffrons le même message sous deux clés différentes, on obtient un système de double-chiffrement toujours IND-CPA. Ce résultat peut être montré en utilisant les techniques hybrides [98, 94] et il a déjà été formellement prouvé dans [4, 2], avec une perte d'avantage d'un facteur 2. Le lecteur intéressé peut lire la preuve au chapitre 10.

Maintenant, montrons comment faire la réduction (cf. figure 5.1). L'attaquant $\mathcal{B}$ reçoit une clé publique donnée pk du système $\mathcal{S}$ et va utiliser l'adversaire $\mathcal{A}$, qui gagne le jeu IND-CCA contre le système de double-chiffrement $\mathcal{S}_{tw}$, pour gagner le jeu IND-CPA. Le simulateur $\mathcal{B}$ exécute $\mathcal{K}(1^k)$ et obtient $(\mathsf{pk}', \mathsf{sk}' = \{\mathsf{sk}'_i\}_{1 \le i \le n})$. Il tire aléatoirement un bit b, et pose $\mathsf{pk}_b = \mathsf{pk}$, alors que $\mathsf{pk}_{1-b} = \mathsf{pk}'$. Puis, il envoie $(\mathsf{pk}_0, \mathsf{pk}_1)$ à $\mathcal{A}$.

A l'étape suivante du jeu, l'adversaire $\mathcal{A}$ retourne deux messages m_0, m_1. Le simulateur $\mathcal{B}$ les envoie au challengeur : ce dernier choisit au hasard un bit b' et chiffre $m_{b'}$ sous $\mathcal{E}_{\mathsf{pk}_b}$, ce qui produit le chiffré cible $a^\star_b = \mathcal{E}_{\mathsf{pk}_b}(m_{b'})$.

Ensuite, $\mathcal{B}$ choisit un bit b'' au hasard et calcule $a^\star_{1-b} = \mathcal{E}_{\mathsf{pk}_{1-b}}(m_{b''})$ et envoie à l'adversaire le double-chiffré cible suivant :

$$y^\star = (a^\star_0, a^\star_1, c^\star)$$

où $c^\star$ est une preuve simulée de la validité de $a^\star_0$ et $a^\star_1$. Cette preuve peut être simulée de manière indistinguable dans le modèle de l'oracle aléatoire. Comme $a^\star_0$ et $a^\star_1$ ne chiffrent pas forcément la même valeur,

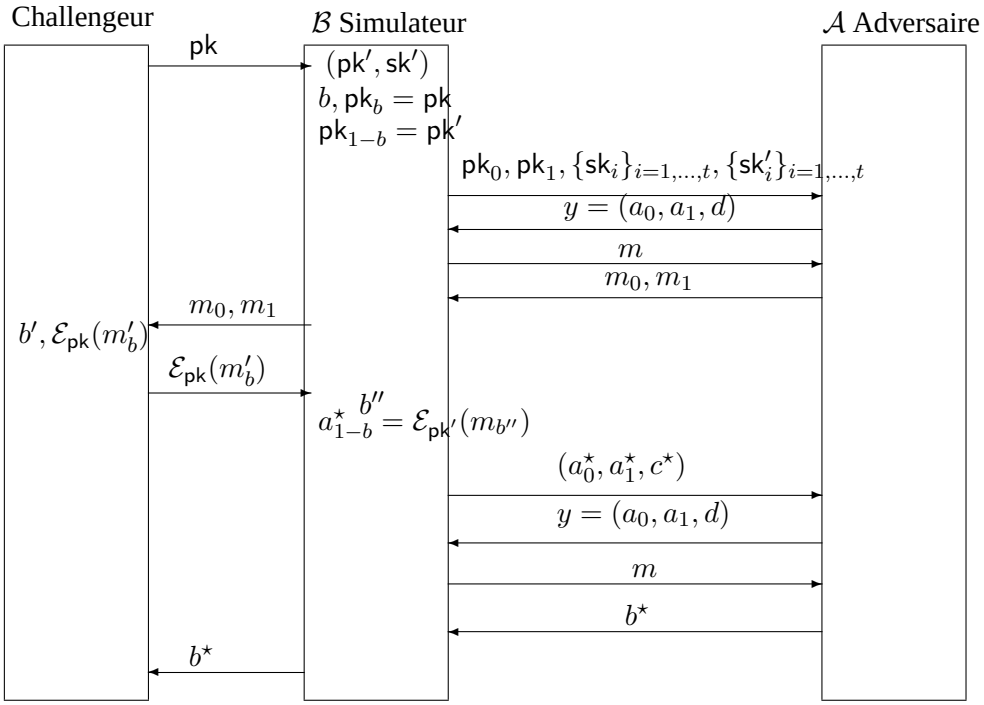


FIG. 5.1: *Preuve de sécurité de la conversion générique.*

l'indistinguabilité est prouvée sous la condition que l'adversaire ne sache pas résoudre le problème de décision : est-ce que $a_0^\star$ et $a_1^\star$ chiffrent le même message?

Montrons comment simuler l'oracle de déchiffrement. L'adversaire $\mathcal{A}$ peut alors faire des requêtes $y = (a_0, a_1, d)$ à l'oracle de déchiffrement, où $a_i = \mathcal{E}_{\mathsf{pk}_i}(m)$ et c est une preuve de validité des chiffrés. Le simulateur $\mathcal{B}$ peut alors déchiffrer a_{1-b}, car il connaît la clé secrète reliée à $\mathsf{pk}_{1-b} = \mathsf{pk}'$. Si la preuve est correcte, nous savons que a_0 et a_1 chiffrent la même valeur m. Cette simulation est parfaite. Si la preuve n'est pas correcte, mais acceptée, l'adversaire a cassé la significativité simulable, après avoir vu une seule preuve.

Enfin, $\mathcal{A}$ répond avec un bit $b^\star$, qui est retourné par $\mathcal{B}$ au challengeur. Comme la simulation peut ne pas être parfaite, l'adversaire peut ne jamais terminer. Dans ce cas, après un certain time-out, $\mathcal{B}$ tire un bit $b^\star$. $\mathcal{B}$ gagne si $b^\star = b'$, et donc avec probabilité :

$$\frac{\varepsilon' + 1}{2} = \Pr\left[b^\star = b' \wedge \mathsf{NIZK}\right] + \Pr\left[b^\star = b' \wedge \neg\mathsf{NIZK}\right] \geq \Pr\left[b^\star = b' \mid \mathsf{NIZK}\right] \cdot \Pr\left[\mathsf{NIZK}\right]$$

Dans la dernière formule, NIZK dénote l'événement qu'aucune des preuves envoyées par l'adversaire à l'oracle de déchiffrement brise la significativité simulable, même après avoir éventuellement vu une preuve.

En effet, si l'adversaire peut forger des preuves d'appartenance pour des faux mots, le simulateur ne répondra jamais avec le message chiffré sous pk'. Par conséquent, l'adversaire peut décider quelle clé a le simulateur.

Cependant, sous l'hypothèse NIZK, disant que l'adversaire ne forge pas une fausse preuve, notre simulateur pour l'oracle de déchiffrement est parfait. Alors, en utilisant la notation pr pour les probabilités sous cette hypothèse :

– dans le cas $b'' = b'$, la simulation est perfaite. En effet, le chiffré cible est un chiffré valide, et toutes

les requêtes de déchiffrement sont des chiffrés valides (sous l'hypothèse NIZK). Ainsi, l'avantage est plus grand que $\varepsilon/2$, grâce aux résultats sur le chiffrement multicast [2, 4] (excepté un avantage possible dans le jeu réel grâce à une attaque sur la significativité). Donc,

$$\mathsf{pr}[b^\star = b' \mid b'' = b'] \geq \frac{\varepsilon/2+1}{2} - \Pr[\neg\mathsf{NIZK}] = \frac{\varepsilon}{4} + \frac{1}{2} - \Pr[\neg\mathsf{NIZK}].$$

– dans le cas $b'' \neq b'$, même un adversaire tout puissant qui sait déchiffrer a_0 et a_1, obtiendra m_0 et m_1. Par conséquent, il ne pourra pas obtenir un avantage. Cependant, l'adversaire qui le détecte peut choisir de ne jamais s'arrêter ou de tricher. S'il décide de ne jamais s'arrêter, le time-out lancé par $\mathcal{B}$ tirera un bit. S'il décide de tricher, il n'aura pas d'information sur b'. Alors, $\mathsf{pr}[b^\star = b' \mid b'' \neq b'] = 1/2$.

Par conséquent,

$$\begin{aligned}\frac{\varepsilon'+1}{2} &\geq \left(\frac{\mathsf{pr}[b^\star = b' \mid b'' = b'] + \mathsf{pr}[b^\star = b' \mid b'' \neq b']}{2}\right) \cdot \Pr[\mathsf{NIZK}] \\ &\geq \frac{1}{2} \cdot \left(\frac{\varepsilon}{4} + 1 - \Pr[\neg\mathsf{NIZK}]\right) \cdot \Pr[\mathsf{NIZK}] \geq \frac{1}{2} \cdot \left(\frac{\varepsilon}{4} + 1 - \frac{9}{4} \cdot \Pr[\neg\mathsf{NIZK}]\right).\end{aligned}$$

Ainsi,

$$\varepsilon' = 2\Pr\left[b^\star = b'\right] - 1 \geq \frac{\varepsilon - 9 \cdot \Pr[\neg\mathsf{NIZK}]}{4}.$$

Afin de borner supérieurement $\Pr[\neg\mathsf{NIZK}]$, nous jouons le même jeu mais en connaissant les deux clés secrètes. Alors, dès que l'adversaire produit une preuve acceptée pour un mot non valide, nous le détectons, et nous le retournons. Ceci brise la significativité simulable en temps t: $\Pr[\neg\mathsf{NIZK}] \leq \mathsf{Succ}^{\mathsf{sim-nizk}}(t)$. □

□

5.3.5 Adversaire actif

Il est clair que dans le cas distribué cette preuve est toujours valable quelque soit l'adversaire. Nous avons fait une preuve rigoureuse sans corruption. Si le schéma sous-jacent est sûr contre les adversaires IND-TCPA, le nouveau évite aussi le même type d'adversaire IND-TCCA.

5.4 Exemples

Le premier exemple de système de chiffrement sémantiquement sûr avec des preuves faciles d'égalité de clair est certainement le cryptosystème El Gamal [66]. Même si des versions plus efficaces ont déjà été proposées [175] (même dans le modèle standard [37]), nous appliquons notre première conversion à ce schéma.

Le second exemple fournira le premier système de chiffrement à seuil sûr contre des attaques actives ou adaptatives à chiffrés choisis basé sur la factorisation. Il est basé sur le système de chiffrement de Paillier [139, 72]. Une autre version pour partager Paillier est aussi apparue dans [56], mais elle n'est pas CCA. Nous pouvons lui appliquer notre conversion pour la transformer.

Dans cette section, nous décrivons brièvement les systèmes de chiffrement et nous insistons sur les preuves d'appartenance à un langage qui sont spécifiques.

5.4.1 Cryptosystème El Gamal

Description du Cryptosystème El Gamal

Soit p un nombre premier sûr tel que $q|p-1$ soit aussi un nombre premier, et g un élément de $\mathbb{Z}_p^*$ d'ordre q. Nous notons G le sous-groupe de $\mathbb{Z}_p^*$ des éléments d'ordre q. Il est engendré par g. Posons $y = g^x$ la clé publique correspondant à la clé secrète x. Pour chiffrer un message $M \in G$, on choisit au hasard $r \in \mathbb{Z}_q$ et on calcule le chiffré $(M.y^r, g^r)$. Pour déchiffrer un chiffré $a = (\alpha, \beta)$, le receveur calcule $\frac{\alpha}{\beta^x} = \frac{M.y^r}{g^{r\,x}} = M$ car $y^r = g^{xr}$. Il est bien connu que la sécurité sémantique de El Gamal est basée sur le problème Diffie-Hellman Décisionnel (DDH) comme on l'a vu dans la première partie.

Version IND-CPA à Seuil du Cryptosystème El Gamal

La clé secrète x est partagée avec un partage de secret à la Shamir. Chaque serveur reçoit une part sk_i de la clé secrète sk et une clé de vérification $\mathsf{vk}_i = g^{\mathsf{sk}_i}$. Pour déchiffrer un chiffré $a = (\alpha, \beta)$, chaque serveur calcule une part de déchiffrement $\beta_i = \beta^{\mathsf{sk}_i}$, et prouve que $\log_g \mathsf{vk}_i = \log_\beta \beta_i$. Le combineur choisit un sous-ensemble S de $t+1$ parts correctes et calcule

$$\beta^x = \prod_{i \in S} \beta_i^{\lambda_{0,i}^S} \bmod p$$

où $\lambda_{i,0}^S$ représente le coefficient de Lagrange. Enfin, le combineur calcule $\alpha/\beta^x \bmod p$ pour reconstituer le clair. Il est facile de montrer que si un adversaire peut casser la sécurité sémantique de ce système de chiffrement, on peut alors construire un attaquant qui sait casser la sécurité sémantique de El Gamal.

Version IND-CCA à seuil du Cryptosystème El Gamal

Il est par conséquent possible d'appliquer notre conversion générique. Soit G un groupe engendré par g. L'algorithme de génération de clé est exécuté deux fois et les clés publiques sont $y_0 = g^{x_0}$ et $y_1 = g^{x_1}$. Pour chiffrer un message M, l'émetteur calcule $a_0 = (M \cdot y_0^r, g^r) = (\alpha_0, \beta_0)$ et $a_1 = (M \cdot y_1^s, g^s) = (\alpha_1, \beta_1)$. La preuve d'égalité de clairs consiste à prouver l'existence de r et s tels que $\beta_0 = g^r$, $\beta_1 = g^s$ et $\alpha_0/\alpha_1 = y_0^r y_1^{-s}$.

A cette fin, on choisit des randoms $a, b \in \mathbb{Z}_q$, et calcule $A = g^a$, $B = g^b$ et $C = y_0^a y_1^b$. Alors, on obtient le challenge aléatoire $e \in \mathbb{Z}_q$ d'une fonction de hachage qui est supposée se comporter comme un oracle aléatoire : $e = H(g, y_0, y_1, a_0, a_1, A, B, C)$. Enfin, on calcule $\rho = a - re \bmod q$ et $\sigma = b + se \bmod q$. Cette preuve peut être facilement vérifiée par $A = g^\rho \beta_0^e$, $B = g^\sigma \beta_1^{-e}$, et $C = y_0^\rho y_1^\sigma (\alpha_0/\alpha_1)^e$, ou de manière équivalente par

$$e = H(g, y_0, y_1, a_0, a_1, g^\rho \beta_0^e, g^\sigma \beta_1^{-e}, y_0^\rho y_1^\sigma (\alpha_0/\alpha_1)^e),$$

où la preuve consiste en le triplet (e, ρ, σ).

Le processus de déchiffrement est simple en utilisant la même technique que celle présentée avant, mais deux fois, après avoir vérifié la validité du chiffré.

Analyse de sécurité.

Le cryptosystème à seuil de base El Gamal est clairement IND-CPA comme nous l'avons prouvé dans le chapitre 2. La conversion générique fait que la nouvelle proposition est IND-TCCA, mais sous la condition que la preuve précédente d'égalité de clairs soit significative simulable. Nous allons donc le prouver.

D'abord, nous devons être capable de construire une liste Q de preuves acceptées pour des mots dans ou en dehors du langage. Ceci peut être fait facilement, grâce à la propriété de l'oracle aléatoire H : on choisit ρ, σ et e dans $\mathbb{Z}_q$, et définissons

$$H(g, y_0, y_1, a_0, a_1, g^\rho \beta_0^e, g^\sigma \beta_1^{-e}, y_0^\rho y_1^\sigma (\alpha_0/\alpha_1)^e) \leftarrow e.$$

Maintenant, supposons qu'avec un accès à cette liste de preuves, un adversaire soit capable de forger une nouvelle preuve pour un faux mot $(\mathsf{pk}_0, \mathsf{pk}_1, a_0, a_1)$, avec probabilité ν, en temps t. Comme tout est incorporé dans la question à l'oracle aléatoire H, on peut appliquer le lemme de bifurcation [151], qui dit que

Lemme 6 *Soit $\mathcal{A}$ une machine de Turing probabiliste en temps polynomial qui peut poser q_h questions à l'oracle aléatoire, où $q_h > 0$. Nous supposons qu'en temps borné t, $\mathcal{A}$ produise, avec probabilité $\nu \geq 7q_h/q$, une nouvelle preuve accepté $(g, y_0, y_1, a_0, a_1; A, B; e; \rho, \sigma)$ pour un faux mot $(\mathsf{pk}_0, \mathsf{pk}_1, a_0, a_1)$. Alors, en temps $t' \leq 16q_h t/\nu$, et avec probabilité $\nu' \geq 1/9$, un rejeu de cette machine retournera deux preuves acceptées pour un faux mot $(\mathsf{pk}_0, \mathsf{pk}_1, a_0, a_1)$:*

$$(g, y_0, y_1, a_0, a_1; A, B; e_0; \rho_0, \sigma_0) \text{ et } (g, y_0, y_1, a_0, a_1; A, B; e_1; \rho_1, \sigma_1),$$

avec $e_0 \neq e_1 \bmod q$.

Supposons que l'adversaire n'a pas cassé la collision de la fonction de hachage H, alors

$$\begin{array}{rcl@{\qquad}rcl} g^{\rho_0}\beta_0^{e_0} & = & g^{\rho_1}\beta_0^{e_1}, & g^{\sigma_0}\beta_1^{-e_0} & = & g^{\sigma_1}\beta_1^{-e_1} \\ y_0^{\rho_0} y_1^{\sigma_0}(\alpha_0/\alpha_1)^{e_0} & = & y_0^{\rho_1} y_1^{\sigma_1}(\alpha_0/\alpha_1)^{e_1} & & & \end{array}$$

et alors,

$$\beta_0 = g^\rho,\ \beta_1 = g^\sigma, \text{ et } \alpha_0/\alpha_1 = y_0^\rho y_1^{-\sigma},$$

où

$$\rho = \frac{\rho_1 - \rho_0}{e_0 - e_1} \bmod q, \text{ et } \sigma = \frac{\sigma_0 - \sigma_1}{e_0 - e_1} \bmod q.$$

Ainsi $\alpha_0 = M_0 y_0^\rho$, et $\alpha_1 = M_1 y_1^\sigma$, nous obtenons finalement $M_0 = M_1$, ce qui signifie que le mot est dans le langage, à moins que l'on ait trouvé une collision pour H. Mais sous l'hypothèse de l'oracle aléatoire, pour obtenir une probabilité plus grande que $1/9$ pour trouver une collision, on doit avoir demandé plus que $\sqrt{q}/3$ questions à H, en utilisant le paradoxe des anniversaire, et ainsi

$$\frac{16q_h t}{\nu} \geq t' \geq \frac{\sqrt{q}}{3}\tau,$$

où τ est le temps nécessaire pour une évaluation de H. Ceci mène à

$$\mathsf{Succ}^{\mathsf{sim-nizk}}(t) \leq \nu \leq 48\frac{q_h}{\sqrt{q}}\frac{t}{\tau}$$

et prouve la significativité du système de preuve. Mais comme ce lemme est toujours valable, même pour un adversaire avec des informations auxiliaires (la liste Q), ceci prouve aussi la significativité simulable.

5.4.2 Cryptosystème de Paillier

Pour être complet, nous rappelons le chiffrement de Paillier et sa version distribuée présentée au chapitre précédent.

Rappel du système de chiffrement

Le système de chiffrement de Paillier est basé sur les propriétés de la fonction lambda de Carmichael dans $\mathbb{Z}_{N^2}^*$. Posons N un module RSA $N = pq$, où p et q sont des nombres premiers. Soit g un entier d'ordre un multiple de N modulo N^2. La clé publique est $\mathsf{pk} = (N, g)$ et la clé secrète est $\mathsf{sk} = \lambda(N)$. Pour chiffrer un message $M \in \mathbb{Z}_N$, on choisit au hasard $x \in \mathbb{Z}_N^*$ et on calcule le chiffré $c = g^M x^N \bmod N^2$. Pour déchiffrer c, on calcule

$$M = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$$

où la fonction L prend ces éléments dans l'ensemble $\mathcal{S}_N = \{u < n^2 \mid u = 1 \bmod N\}$ et vérifie $L(u) = (u-1)/N$. La sécurité sémantique est basée sur la difficulté de distinguer les N-ièmes résidus modulo N^2.

Rappel de la version IND-CPA à seuil du Cryptosystème de Paillier

On rappelle brièvement le schéma décrit dans le chapitre précédent pour clarifier les notations. On rappelle que l'on note $\Delta = n!$ où n est le nombre de serveurs.

Algorithme de Génération de Clés : Soit N un entier, produit de deux nombres premiers sûrs p et q, tels que $p = 2p' + 1$ et $q = 2q' + 1$ et $\mathrm{pgcd}(N, \varphi(N)) = 1$. On peut remarquer que l'exigence sur les nombres premiers sûrs peut être évitée si on utilise [57, 75][53]. Cependant, pour simplifier la présentation, nous supposerons que le module RSA est composé de nombres premiers sûrs. Posons $m = p'q'$. Soit β un élément aléatoirement choisi dans $\mathbb{Z}_N^*$.

La clé secrète $\mathsf{sk} = \beta \times m$ est partagée avec un schéma de partage à la Shamir [170] modulo mN. Soit v un carré qui génère avec probabilité écrasante le groupe cyclique des carrés de $\mathbb{Z}_{N^2}^*$. Les clés de vérification vk_i sont obtenues avec la formule $v^{\Delta \mathsf{sk}_i} \bmod N^2$.

Algorithme de Chiffrement : Pour chiffrer un message M, on choisit aléatoirement $x \in \mathbb{Z}_N^*$ et on calcule $c = g^M x^N \bmod N^2$.

Algorithme de Déchiffrement Partiel : Le i-ième joueur P_i calcule la part de déchiffré $\sigma_i = c^{2\Delta \mathsf{sk}_i} \bmod N^2$ en utilisant sa part de la clé secrète sk_i. Il génère une preuve de validité qui garantit que $c^{4\Delta} \bmod N^2$ et $v^{\Delta} \bmod N^2$ ont été élevés à la même puissance sk_i afin d'obtenir σ_i^2 et vk_i.

Algorithme de Reconstruction : Si moins de $t+1$ parts de déchiffrement ont des preuves correctes de validité, l'algorithme de reconstruction échoue. Sinon, soit S l'ensemble de $t+1$ parts valides, on peut calculer le clair en utilisant la formule d'interpolation de Lagrange dans les exposants.

Dans le chapitre précédent, nous avons prouvé le théorème suivant :

> **Théorème 19.** Sous l'hypothèse décisionnelle de résiduosité modulo un nombre composé et dans le modèle de l'oracle aléatoire, la version à seuil du système de chiffrement de Paillier est IND-TCPA contre des adversaires actifs et statiques.

53. Ces articles permettent d'utiliser le protocole de Shoup [174] en évitant les nombres premiers sûrs. Ils partagent ainsi complètement le cryptosystème de Paillier du protocole de génération des clés au processus de déchiffrement. En effet, il est apparemment difficile de générer des modules RSA sûrs en utilisant le protocole [27].

Version IND-CCA à seuil du Cryptosystème de Paillier

Il est alors possible d'appliquer notre conversion générique.

Algorithme de Génération des Clés : Choisir, pour $j = 0, 1$, un entier N_j, produit de deux nombres premiers sûrs p_j et q_j. Posons $m_j = (p_j - 1)(q_j - 1)/4$. Soit β_j un élément choisi au hasard dans $\mathbb{Z}^*_{N_j}$.

Les clés secrètes $\mathsf{sk}_j = \beta_j \times m_j$ sont partagées avec un partage de secret à la Shamir [170] modulo $m_j N_j$. Soit v_j un carré générateur du groupe cyclique des carrés de $\mathbb{Z}^*_{N_j^2}$. Les clés de vérification vk_j sont obtenues avec la formule $v^{\Delta \mathsf{sk}_{i,j}} \bmod N_j^2$.

Algorithme de Chiffrement : Pour chiffrer un message M, il faut tirer au hasard $x_j \in \mathbb{Z}^*_{N_j}$ et calculer $a_j = g_j^M x_j^{N_j} \bmod N_j^2$. Ensuite, il faut calculer une preuve que a_0 et a_1 chiffrent la même valeur. Pour cela, soit r un élément choisi au hasard dans $[0, A[$, et des éléments aléatoires $\alpha_j \in \mathbb{Z}^*_{N_j}$. Calculons $y_j = g_j^r \alpha_j^{N_j} \bmod N_j^2$. Posons e la valeur de la fonction de hachage $H(g_0, g_1, a_0, a_1, y_0, y_1)$ où H est une fonction de hachage qui retourne des valeurs dans l'intervalle $[0, B[$. Alors, on pose $z = r + e \times M$, $u_j = \alpha_j x_j^e \bmod N_j$ Une preuve d'égalité est un quadruplet

$$(e, z, u_0, u_1) \in [0, B[\times[0, A[\times\mathbb{Z}^*_{N_1} \times \mathbb{Z}^*_{N_2}$$

Il est vérifié par l'équation

$$e = H(g_0, g_1, a_0, a_1, g_0^z u_0^{N_0}/a_0^e \bmod N_0^2, g_1^z u_1^{N_1}/a_1^e \bmod N_1^2)$$

Le processus de déchiffrement est identique à celui du chapitre précédent.

Il est remarquable que la conversion générique du cryptosystème de Paillier conserve les propriétés homomorphiques : $\mathcal{E}(M_1+M_2) \equiv \mathcal{E}(M_1)\times\mathcal{E}(M_2)$ et $\mathcal{E}(kM) \equiv \mathcal{E}(M)^k$. Par exemple, dans les schémas de vote, comme [56, 1], l'autorité peut vérifier les preuves universellement de validité des chiffrés et calculer le total. Cependant, le résultat (produit des chiffrés) ne sera plus un chiffré valide car on ne peut pas recalculer la preuve à moins de connaître une trappe comme on le verra dans la section suivante.

5.5 Conclusion

Dans ce chapitre nous avons construit des conversions génériques et hybrides de systèmes de chiffrement sûrs contre les attaques à chiffrés choisis à partir de n'importe quel schéma IND-CPA. Nous avons proposé la première version de cryptosystèmes sûrs CCA basés sur le problème de la factorisation. En outre, en employant les techniques des chapitres précédents et suivants, on peut *complètement* distribuer ce cryptosystème.

Cependant, comme il est dit dans [175], il apparaît difficile de partager RSA. Le cryptosystème RSA n'est pas sémantiquement sûr comme le prouve l'annexe 10. Cependant, en utilisant le padding OAEP, on peut prouver que RSA-OAEP est IND-CCA. Si on utilise ce schéma sans redondance, le cryptosystème est IND-CPA. Cependant, il paraît difficile de partager "RSA-OAEP sans redondance" même dans le modèle de l'oracle aléatoire.

6

Partage d'un générateur de clés Diffie-Hellman

Dans ce chapitre, nous présentons un algorithme de génération de clés Diffie-Hellman en un seul tour sans utiliser de canaux secrets entre les serveurs. Nous avons présenté avec Jacques Stern les résultats de ce chapitre dans un article à la conférence PKC '01 [76].

En 1991, Pedersen a conçu le premier schéma pour générer des clés basées sur le logarithme discret sans avoir besoin d'avoir de distributeur de confiance. Comme ce processus est simple et efficace, il a été ensuite utilisé dans de nombreux protocoles comme brique de base pour partager *complètement* des protocoles utilisant le logarithme discret comme DSS ou El Gamal. Ainsi, pendant quelques années, on a cru que ce protocole robuste était sûr. Cependant, en 1999, Gennaro, Jarecki, Krawczyk, et Rabin [90] ont prouvé qu'une des exigences n'était pas garantie. En particulier, des adversaires peuvent biaiser la distribution de la clé pour qu'elle ne soit plus uniforme.

Gennaro *et al.* ont alors tenté de réparer ce schéma en corrigeant la faiblesse découverte dans la distribution de la clé secrète. Le nouveau protocole proposé corrige certes la distribution, mais n'est plus aussi pratique que celui de Pedersen car il nécessite deux tours pour tous les serveurs même s'il n'y a pas d'attaquants. De plus, ils n'ont pas totalement éliminé la phase qui était au cœur de l'attaque, la "phase de gestion des plaintes". Elle sert à jeter les tricheurs hors de l'ensemble des joueurs honnêtes. De ce fait, le schéma résultant apparaît complexe et difficile à mettre en œuvre dans les situations pratiques.

Pour éviter cette phase et d'autres inconvénients comme la phase d'initialisation où les canaux secrets doivent être créés, nous avons présenté dans [76] un nouveau schéma en un *seul* tour qui génère une clé Diffie-Hellman en utilisant seulement des canaux publics.

Dans ce chapitre, nous décrivons et prouvons notre protocole *simplifié* de génération de clés Diffie-Hellman. Ce protocole est très efficace lorsque le nombre de serveurs n'est pas très important. Dans le cas où ce nombre devient trop grand, nous montrons enfin comment rendre le protocole plus efficace.

Sommaire

6.1 Problématique

Afin de concevoir des cryptosystèmes à seuil comme des schémas de chiffrement à clé publique ou de signature, la première étape consiste à partager la procédure de génération de clés. En effet, si un distributeur de confiance est utilisé dans le protocole de génération de clés, la sécurité de tout le système distribué dépend d'un seul serveur. Les protocoles de génération de clés sont basés sur des processus de distribution d'aléa. Les serveurs génèrent de manière conjointe une clé aléatoire de telle façon qu'à la fin du processus, tous les serveurs honnêtes ont une part de la clé secrète.

Les améliorations sur la génération aléatoire de clés privées pour la cryptographie à clé publique tombent usuellement dans deux domaines : la distribution d'un secret pour des cryptosystèmes basés sur le log discret et la distribution de clés RSA.

Dans le cas RSA, le problème est partiellement résolu par Boneh et Franklin [27]. Cependant, leur protocole ne permet pas de générer un module RSA *sûr* et n'est pas robuste contre des adversaires actifs. Nous avons étudié dans le chapitre 3 comment on pouvait se passer de tels modules pour partager *complètement* la signature RSA. Nous avons proposé un nouvel algorithme de génération de modules RSA pour des modules de forme spéciale. Ces modules présentent des caractéristiques presque similaires à celles des modules RSA sûrs, c'est-à-dire que $p' = \frac{p-1}{2}$ et $q' = \frac{q-1}{2}$ sont premiers entre eux et n'ont pas de facteurs premiers inférieurs à une certaine borne.

Des méthodes pour partager des clés de cryptosystèmes basés sur le logarithme discret sont connues depuis longtemps, en commencant par les articles de Feldman et Pedersen [68, 143, 144]. Cependant, une erreur dans les exigences a été découverte et une première solution ainsi qu'un modèle de sécurité pour les protocoles DKG [54] ont été définis par Gennaro *et al.* dans [90]. La solution a été améliorée par Canetti *et al.* dans [35] pour résister aux attaques adaptatives. Les schémas sont basés sur le schéma de partage de secret inconditionnellement sûr de Pedersen [144] et par conséquent nécessitent des canaux secrets. Jarecki dans [110] a proposé un protocole comportant uniquement des canaux publiques mais cette solution nécessite un schéma de chiffrement ayant des propriétés spéciales, dit *sans engagement* qui la rend peu efficace en pratique.

Alors que les solutions précédentes de DKG prouvent la sécurité dans un modèle de théorie de l'information, nous utilisons ici un modèle calculatoire le but d'un tel protocole étant de construire une clé publique. Nous éliminons donc les valeurs mises en gage de [110, 35] qui sont nécessaires pour prouver la sécurité contre des adversaires adaptatifs. Pour résister à de tels adversaires, nous avons conçu avec Jacques Stern un protocole en un seul tour.

Suite à l'article [155] de Poupard et Stern, nous introduisons ici des canaux publiques afin de réduire le nombre de tours de communications à un seul tour. L'utilisation d'un protocole non-interactif, nous permet d'ignorer les adversaires adaptatifs. En effet, ce genre d'attaquant n'a pas de sens dans un protocole à un tour. Pour obtenir un protocole non-interactif, nous avons besoin de primitives telles que tous les serveurs peuvent décider si les autres serveurs ont correctement exécuté leurs tâches et un

54. Distributed Key Generation en anglais.

réseau synchrone pour éviter les “rushing” attaques. Par conséquent, nous avons besoin de preuves non-interactives zero-knowledge sûres dans le modèle de l’oracle aléatoire et de canaux publiques. Dans la section 6.2.6, nous affaiblissons l’hypothèse de réseau synchrone et nous présentons un modèle dans lequel nous pouvons éviter les attaques de type “rushing” et les adversaires adaptatifs au prix d’un serveur particulier.

6.1.1 Protocole de Pedersen

Le protocole de génération de clés Diffie-Hellman sans distributeur de confiance de Pedersen [143] est un schéma non-interactif nécessitant un canal de broadcast et des canaux privés entre chaque paire de serveurs. Le schéma est organisé en deux étapes : dans la première étape, les serveurs sélectionnent la clé. Dans la seconde étape, la clé est distribuée entre les serveurs. Durant la phase de distribution, chaque serveur agit comme le distributeur dans un protocole de Feldman [68] de partage de secret vérifiable.

Le serveur P_i choisit $x_i \in \mathbb{Z}_q$ au hasard et prend t nombres aléatoires $a_{i,k}$ dans $\mathbb{Z}_q$. Ensuite, il fixe $f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k$ où $a_{i,0} = x_i$. Il envoie de manière privée la part secrète $s_{i,j} = f_i(j) \bmod q$ au serveur P_j et broadcaste comme information publique $y_{i,j} = g^{s_{i,j}} \bmod p$, et $A_{i,k} = g^{a_{i,k}} \bmod p$ pour $k = 0, \ldots, t$. Ces données peuvent être utilisées par tous les autres serveurs pour vérifier si le random x_i choisi par P_i, a été correctement distribué. Chaque serveur P_j vérifie si

$$y_{i,j} \stackrel{?}{=} \prod_{k=0}^{t} A_{i,k}^{j^k} \bmod p$$

Si cette égalité n’est pas vérifiée, $s_{i,j}$ n’est pas le logarithme discret de $y_{i,j}$, le serveur P_j broadcaste une complainte contre P_i.

Les complaintes sont alors gérées par différentes stratégies. L’une d’entre elle est la suivante : si plus de t serveurs se plaignent contre le serveur P_i, ce serveur est clairement en faute et est disqualifié. Sinon, le joueur P_i révèle la part $s_{i,j}$ à chaque joueur P_j qui s’est plaint. Si une des parts révélées échoue durant la vérification de l’équation $y_{i,j} = g^{s_{i,j}} \bmod p$, le joueur P_i est disqualifié, sinon P_i est toujours qualifié. On peut alors définir QUAL comme l’ensemble des joueurs non-disqualifiés. La clé publique est définie par $y = \prod_{i \in \mathrm{QUAL}} y_i$ où $y_i = A_{i,0} = g^{x_i} \bmod p$.

6.1.2 Attaque du schéma de Pedersen

Dans la section précédente, le serveur P_i choisit un secret x_i au hasard et le partage entre les autres serveurs. Les disqualifications ont lieu après la phase de distribution. Or, la phase de sélection dont le but est de fixer sans ambiguité la clé publique, n’est pas terminée avant le commencement de la phase de distribution. Ainsi, l’adversaire peut calculer la clé publique à la fin de la phase de distribution en utilisant les valeurs $A_{i,0}$. En fonction de la valeur calculée et du but de l’adversaire, ce joueur peut par exemple disqualifier quelques membres afin de modifier la distribution de la clé publique. Gennaro, Jarecki, Krawczyk et Rabin ont décrit une attaque dans laquelle deux membres malicieux biaisent la distribution du dernier bit de la clé publique avec probabilité 3/4 au lieu de 1/2. Leur attaque s’appuie sur le fait que le schéma de Pedersen utilise des canaux secrets entre chaque paire de serveurs. Ainsi, quand une erreur apparaît, il est impossible de savoir quelle serveur triche.

Pour éviter cette attaque, Gennaro *et al.* ont dupliqué le schéma: dans la première phase, le groupe honnête est sélectionné et dans une deuxième phase, la valeur publique associée au secret partagé est rendue publique. Dans ce cas, le groupe qualifié est déterminé à la fin de la première phase. Dans la phase de sélection, chaque serveur met en gage une valeur aléatoire avec un schéma inconditionnel de Pedersen [144] ; alors que dans la phase de distribution, les joueurs relâchent des informations permettant

à n'importe qui de calculer la valeur publique. Si un joueur triche dans la seconde phase, mais appartient au groupe QUAL, les autres joueurs exécutent un algorithme de code correcteur d'erreurs avec les valeurs que le joueur tricheur avait distribuées durant la première phase.

Le besoin de deux phases provient du fait que les canaux privés utilisés cachent les joueurs fautifs. Par conséquent, après la première phase qui utilise un algorithme symétrique, nous ne savons pas si le tricheur est :

– l'émetteur qui a envoyé une mauvaise part dans le canal symétrique, ou

– le receveur qui a déclaré avoir reçu une mauvaise part.

Par conséquent, la seconde phase est nécessaire pour gérer les "plaintes". Cependant, on peut s'attendre en général à ce qu'aucun des serveurs ne soit corrompu. Ainsi, cette deuxième phase apparaît comme étant redondante et inutile. De plus, c'est la phase la plus gourmande du protocole.

6.2 Nouvelle proposition

Notre proposition s'efforce de simplifier les protocoles précédents. Dans une implémentation réelle de canaux privés, un tour supplémentaire doit être exécuté pour partager une clé secrète entre chaque paire de serveurs. Ce premier tour est usuellement vu comme une hypothèse sur le canal mais cette phase est pénalisante dans les implémentations pratiques.

De plus, dans les canaux privés, lors de la première phase, il est impossible de savoir si le joueur fautif est l'émetteur ou le receveur. La seconde phase de [90] est par conséquent nécessaire pour résoudre les plaintes provenant de cette ambiguïté. Ainsi, si nous utilisons un schéma de partage de secret publiquement vérifiable (PVSS) et un schéma de chiffrement publiquement vérifiable (PVE) [30], nous sommes capables de détecter immédiatement si l'émetteur a envoyé de mauvaises parts. Alors, les joueurs malicieux sont détectés et nous ne les acceptons pas dans le groupe de membres qualifiés.

Par conséquent, notre schéma consiste en une seule phase dans laquelle chaque serveur i partage son nombre aléatoire x_i avec un schéma PVSS et transfère la part $s_{i,j}$ du serveur P_j, $j \neq i$, avec un schéma PVE. Ainsi, tout serveur P_k peut vérifier que le receveur P_j est capable de retrouver sa part $s_{i,j}$. Comme une seule phase avec un schéma PVSS est utilisée, tous les serveurs sont capables de calculer la clé publique à la fin de cette étape. Par conséquent, nous avons besoin d'un réseau synchronisé pour éviter l'attaque de "rushing", durant laquelle un adversaire attend que tous les autres serveurs aient joué avant de définir sa propre valeur. Dans ce scénario, un adversaire peut choisir une clé publique ou tout au moins fixer un biais dans la distribution. Par conséquent, la révélation des valeurs $A_{i,0}$ sera faite à un moment fixé pour chaque serveur. Dans 6.2.6 nous décrivons comment éviter l'hypothèse de réseau synchronisé grâce à un nouveau type de joueur.

Complexité du protocole. Dans notre schéma, tous les utilisateurs doivent vérifier beaucoup de preuves. En particulier, si le nombre de joueurs est n et si chacun d'entre eux partage son secret aléatoire en n parts, nous avons $O(n^2)$ parts dans ce schéma. De plus, dans notre scénario, toutes les parts sont broadcastées et doivent être vérifiées par tous les participants. On évite ainsi la "phase de complainte" dans laquelle un joueur menteur informe les autres que la vérification ne fonctionne pas. Ainsi, la complexité de calcul est en $O(kn^2)$, alors que d'autres schémas ont une complexité en $O(kn)$. Cependant, nous pensons que cette dégradation de la complexité ne remet pas en cause le caractère pratique puisque le nombre de participants est usuellement limité. Finalement, les constantes cachées dans la notation O des autres protocoles rendent la comparaison peu significative.

Amélioration de la complexité. En première lecture, notre schéma paraît coûteux en terme de calculs car tous les serveurs doivent vérifier les parts des autres. Cependant, comme on l'a dit précédemment, dans

les situations pratiques, nous n'avons qu'un petit nombre de serveurs. De plus, si nous voulons exécuter notre protocole avec plus de serveurs, nous fournissons à la section 6.2.6 une solution pour accélérer la phase de vérification. Grâce à des vérifications en batch efficaces (cf. annexe 11), nous prouvons que la complexité en calcul de notre schéma est comparable à celle des autres schémas.

6.2.1 Modèle et exigences de sécurité

Le réseau et les joueurs

Notre jeu comprend les joueurs suivants : un ensemble de n serveurs $P_1, \ldots, P_n$ et un adversaire qui peut contrôler jusqu'à t joueurs. Ils sont connectés à travers un canal de communication synchrone et à un canal de broadcast. Un joueur P_i est considéré *bon* aussi longtemps qu'il suit le protocole et *mauvais* dès qu'il dévie du protocole.

Définition formelle

Un t-parmi-n *schéma de génération de clé à seuil* est un protocole qui permet à n'importe quel sous-ensemble de $t+1$ joueurs parmi n de générer une clé secrète, mais interdit la génération si moins de t joueurs participent au protocole.

Une génération de clé à seuil t-parmi-n est composée d'un *algorithme de génération de clé* qui prend en entrée un paramètre de sécurité k, le nombre n de serveurs de génération, et le paramètre de seuil t ; il retourne une clé publique pk, et une liste $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ de parts de la clé privée associée à la liste $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ des parts de la clé publique.

Exigences de sécurité

Les exigences de sécurité d'un schéma de génération de clés sont : l'exactitude et la sécurité. Nous présentons ici les exigences pour la génération de clés basée sur le problème du logarithme discret $\mathsf{pk} = y = g^x \bmod p$ et $\mathsf{sk} = x$. La clé sk est partagée entre n serveurs.

L'exactitude du protocole consiste en les trois points suivants :

- Tout sous-ensemble de $t+1$ parts fournies par les joueurs honnêtes définit toujours la même unique clé secrète x.
- Tous les serveurs honnêtes ont la même valeur de clé publique $y = g^x \bmod p$, où x est l'unique secret garanti par le point précédent.
- La valeur x est uniformément distribuée dans $\mathbb{Z}_q$, et ainsi y est uniformément distribué dans le sous groupe engendré par g.

La sécurité signifie qu'aucune information sur x ne peut être apprise par un adversaire au-delà de l'égalité $y = g^x \bmod p$. La sécurité, confidentialité de la clé secrète, peut être exprimée plus formellement en terme de simulation. La simulation permet de prouver que l'adversaire $\mathcal{A}$ n'apprend rien sur les nombres aléatoires des serveurs non corrompus. Plus précisément, si $\mathcal{A}$ a connaissance de t nombres aléatoires des serveurs corrompus et connaît la valeur publique y, un programme, appelé le simulateur $\mathcal{S}$, peut être exécuté en moyenne en temps polynomial, de telle façon que la vue de l'adversaire durant une exécution réelle est indistinguable de la sortie du simulateur. Ainsi, l'adversaire ne peut pas savoir si la distribution provient d'un simulateur ou d'une exécution réelle. Par conséquent, comme le simulateur ne connaît pas l'information secrète, la sortie du simulateur ne peut pas être utilisée par l'adversaire pour apprendre des informations sur les secrets détenus par les serveurs non corrompus.

Le Jeu de l'adversaire

Pour définir l'exactitude et la sécurité contre un adversaire statique, on considère le jeu suivant joué contre un tel adversaire.

A1 L'adversaire $\mathcal{A}$ a connaissance de la sortie de l'algorithme distribué de génération de clé publique : la clé publique y.

A2 L'attaquant choisit de corrompre t serveurs. Il apprend leurs secrets et contrôle activement leur comportement.

A3 Chaque participant choisit un nombre aléatoire et le partage en utilisant un schéma de partage de secret publiquement vérifiable entre eux.

6.2.2 Preuve de chiffrement *équitable*

Dans cette section, nous présentons une preuve du même style que [156] qui permet de prouver que le déchiffrement (du chiffré de x par Paillier) $Y = G^x u^N \bmod N^2$ en base G permet de retrouver le logarithme discret de $y = g^x \bmod p$ en base g, où g est d'ordre premier q dans $\mathbb{Z}_p^*$.

Nous décrivons une preuve non-interactive statistiquement zero-knowledge de l'existence de deux petits nombres σ et τ tels que $|\sigma| < A$ et $|\tau| < B$ et qui vérifient que $G^\sigma Y^{-\tau}$ est un résidu N-ième et $\sigma\tau^{-1} = \log_g y$. Nous prouvons la sécurité de cette preuve dans le modèle de l'oracle aléatoire.

Description de la preuve. Soient A, B et S trois entiers tels que $L(A) \geq L(B) + L(S) + k'$ où k' est un paramètre de sécurité et soit $x \in [0, S[$ la valeur secrète. La valeur $L(B)$ est la longueur de la sortie de la fonction de hachage H. On rappelle que $y = g^x \bmod p$ est une valeur publique partagée entre le prouveur et le vérifieur.

Le prouveur choisit un aléa r dans $[0, A[$ et un random $s \in \mathbb{Z}_N^*$. Il calcule alors $g^r \bmod p$ et $G^r s^N \bmod N^2$. Soit e le résultat de la fonction de hachage $H(g, G, y, Y, g^r \bmod p, G^r s^N \bmod N^2)$. Ensuite, le prouveur calcule $z = r + ex$ et $w = su^e \bmod N$. Si $z \notin [0, A[$, le prouveur recommence avec d'autres valeurs aléatoires r et s jusqu'à ce que $z \in [0, A[$. La preuve est constituée du triplet $(e, z, w) \in [0, B[\times[0, A[\times[0, N[$. On la vérifie en calculant les équations suivantes :

$$e \stackrel{?}{=} H(g, G, y, Y, g^z y^{-e} \bmod p, G^z w^N Y^{-e} \bmod N^2),$$

$$z \in [0, A[\quad \text{et} \quad y^q \stackrel{?}{=} 1 \bmod p$$

Consistance.

Théorème 20. L'exécution entre un prouveur qui connaît le secret x et un vérifieur est réussie avec probabilité écrasante si $SB/A < 1/2^{k'}$ est négligeable.

Preuve: Le vérifieur a accès à (e, z, w) où $z = r + ex < A$, $w = su^e \bmod N$, et $e = H(g, G, y, Y, g^r \bmod p, G^r s^N \bmod N^2)$. Il peut vérifier que $z < A$,

$$g^z y^{-e} = g^{r+ex}(g^x)^{-e} = g^r \bmod p$$

et

$$G^z w^N Y^{-e} = G^{r+ex}(su^e)^N (G^x u^N)^{-e} = G^r s^N \bmod N^2$$

Si le prouveur suit le protocole, la preuve échoue si et seulement si $z \geq A$. La probabilité d'échec d'un tel événement pris sur tous les choix possible de r est plus petit que SB/A. Par conséquent, l'exécution du protocole est réussie avec probabilité plus grande que $1 - \frac{SB}{A}$. Alors, si SB/A est négligeable, la probabilité de succès est écrasante. □

Significatif.

Théorème 21. Si le vérifieur accepte la preuve, avec probabilité $\geq 1/B + \varepsilon$ où ε est une quantité non negligeable, alors en utilisant le prouveur comme une "boîte noire", il est possible de calculer σ et τ tels que $|\sigma| < A$ et $|\tau| < B$ tels que $\sigma\tau^{-1} = x \bmod q$, $g^x = y \bmod p$ et $G^\sigma Y^\tau$ est un résidu N-ième modulo N^2.

Preuve: Pour un t donné, si le prouveur peut trouver deux triplets (e, z, w) et (e', z', w') qui passent la preuve avec probabilité non negligeable, il peut obtenir les equations suivantes : $G^{z-z'}(w/w')^{Ne} = Y^{e-e'} \bmod N^2$ et $g^{z-z'} = y^{e-e'} \bmod p$.

Ainsi, si on note $\sigma = z - z'$ and $\tau = e - e'$:

$$G^\sigma (w/w')^{Ne} = Y^\tau \bmod N^2 \quad \text{et} \quad g^\sigma = y^\tau \bmod p \tag{6.1}$$

et $|\sigma| < A$ and $0 < |\tau| < B$.

Comme $y^q = 1 \bmod p$ et comme il existe un unique sous-groupe d'ordre q dans $\mathbb{Z}_p^*$, y est dans $\langle g \rangle$. Ainsi, la seconde equation, nous donne que $\sigma\tau^{-1} \bmod q$ est le logarithme discret de y en base g, c'est-à-dire $\sigma \times \tau^{-1} = x \bmod q$.

Notons $d = \mathrm{pgcd}(\sigma, \tau)$. Comme q est un nombre premier, on obtient $\sigma/d = \tau/d \times x \bmod q$. Soit $\sigma_0 = \sigma/d$, $\tau_0 = \tau/d$. La connaissance de (σ_0, τ_0) permet de calculer le secret $x = \sigma_0 {\tau_0}^{-1} \bmod q$.

Soit $\tilde{x}$ le résultat du déchiffrement de Y. Si $g^{\tilde{x}} = y \bmod p$, nous avons fini. Sinon, on recherche les valeurs σ_0 et τ_0 où $\sigma_0 = \sigma/d$, $\tau_0 = \tau/d$ et $d = \mathrm{pgcd}(\sigma, \tau)$ pour trouver x. Dans [156], Poupard et Stern ont décrit comment trouver σ_0 et τ_0 à partir de $\tilde{x}$ et N pourvu que la preuve soit correcte. Ils ont montré que le vecteur le plus court du réseau L de dimension deux dont une base est donnée par $((N, 0), (\tilde{x}, 1))$, correspond au vecteur (σ_0, τ_0) quand $N > 2AB$[55]. Ainsi, comme la dimension du réseau est deux, l'algorithme de Gauss peut être utilisé pour retrouver efficacement le vecteur le plus court en $O(\log N)$ opérations.

Par conséquent, si la preuve est bien formée, le participant P_i peut toujours retrouver la part recherchée x qui vérifie $y = g^x \bmod p$. Cette propriété de chiffrement *équitable* est utile pour garantir que le receveur recevra les données correctes. □

55. En effet, il est clair que ce vecteur est dans le réseau. De plus, on peut prouver que si σ et τ sont petits, alors il y a unicité de ces valeurs. Pour montrer l'unicité, on peut montrer le théorème suivant : Si $x = \sigma/\tau \bmod N$, tels que $-A \leq \sigma \leq A$ et $0 < \tau < B$, alors l'algorithme de Gauss recouvre σ et τ de manière unique si $2AB < N$. Montrons ce résultat par l'absurde. Supposons qu'il existe (σ_1, τ_1) et (σ_2, τ_2) deux couples tels que $-A \leq \sigma_i \leq A$ et $0 < \tau_i \leq B$ de solutions retournées par l'algorithme de Gauss si on lui donne comme entrée la base initiale $((N, 0), (x, 1))$. Alors les solutions (σ_1, τ_1) et (σ_2, τ_2) forment une base d'un sous-réseau L' du réseau L. Le déterminant du réseau L, qui vaut N, divise donc le déterminant du sous-réseau L'. Par conséquent, $N \left| \begin{vmatrix} \sigma_1 & \sigma_2 \\ \tau_1 & \tau_2 \end{vmatrix} \right.$, c'est-à-dire $N | |\sigma_1\tau_2 - \sigma_2\tau_1|$. De plus, $|\sigma_1\tau_2 - \sigma_2\tau_1| \leq 2AB$ et donc $|\sigma_1\tau_2 - \sigma_2\tau_1| < N$ car N>2AB. On en déduit que le déterminant du sous-réseau L' vaut 0 et par conséquent, les vecteurs (σ_1, τ_1) et (σ_2, τ_2) sont colinéaires.

Statistiquement Zero-Knowledge.

Théorème 22. Cette preuve est une preuve non-interactive statistiquement zero-knowledge.

Preuve: Dans le modèle de l'oracle aléatoire, il est possible de construire un simulateur qui simule la vue de l'adversaire sans connaître la valeur secrète x. Quand un joueur non corrompu est supposé générer la preuve pour un y et Y donnés, le simulateur choisit $e \in [0, B[$, $z \in [0, A[$ et $w \in \mathbb{Z}_N^*$ au hasard, et définit e comme étant la valeur de l'oracle aléatoire au point $(g, G, y, Y, g^z y^{-e} \bmod p, G^z w^N Y^{-e} \bmod N^2)$. Avec probabilité écrasante, le simulateur n'a pas déjà défini l'oracle aléatoire en ce point. La preuve est alors (z, w, e). Il est simple de vérifier que la distribution produite par le simulateur est statistiquement proche de la distribution parfaite lorsque BS/A est négligeable. □

6.2.3 Description

Chaque serveur doit vérifier les preuves broadcastées par les autres joueurs et sélectionner le groupe qualifié des serveurs. Tous les joueurs sont considérés comme des machines de Turing probabilistes fonctionnant en temps polynomial en la longueur du paramètre de sécurité. Dans la phase d'initialisation, chaque serveur exécute l'algorithme de génération de clés du cryptosystème de Paillier. Pour $i = 1$ à n, les clés $\mathsf{pk}_i = (G_i, N_i)$ sont rendues publiques et le serveur P_i conserve secrètement sk_i. La valeur N_i est un module RSA, G_i est un élément dans $\mathbb{Z}_{N_i^2}^*$ d'ordre un multiple de N_i et $\mathsf{sk}_i = \lambda(N_i)$.

Considérons le protocole suivant :

- Chaque serveur P_i génère un random $s_{i,0}$, fixe $a_{i,0} = s_{i,0}$ et choisit $a_{i,k}$ au hasard dans $\mathbb{Z}_q$ pour $1 \leq k \leq t$. Les nombres $a_{i,0}, \ldots, a_{i,t}$ définissent le polynôme $f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k \in \mathbb{Z}_q[X]$. Il calcule alors $s_{i,j} = f_i(j) \bmod q$, puis il broadcaste : pour $k = 0, \ldots, t$, $A_{i,k} = g^{a_{i,k}} \bmod p$ et $y_{i,j} = g^{s_{i,j}} \bmod p$, $Y_{i,j} = G_j^{s_{i,j}} u_{i,j}^{N_i} \bmod N_j^2$, et une preuve $(e_{i,j}, w_{i,j}, z_{i,j})$.

- Alors, pour chaque $1 \leq i, j \leq n$, les serveurs vérifient que :

$$\prod_{k=0}^{t} A_{i,k}^{j^k} = \prod_{k=0}^{t} g^{a_{i,k} j^k} = g^{\sum_{k=0}^{t} a_{i,k} j^k} = g^{f_i(j)} \bmod p$$

  et vérifient si $g^{f_i(j)} \bmod p$ est égal à $y_{i,j}$ afin de vérifier si la distribution est correcte. Les serveurs vérifient aussi les preuves $(e_{i,j}, w_{i,j}, z_{i,j})$ et si $y_{i,j}^q = 1 \bmod p$ pour $1 \leq i, j \leq n$.

- L'ensemble QUAL des serveurs qualifiés est défini dans l'ensemble des joueurs qui ont correctement joué. Les autres sont disqualifiés.

- Chaque serveur P_j déchiffre $Y_{i,j}$ et obtient $s_{i,j}$ pour $1 \leq i \leq n$. Il stocke les parts $s_{i,j}$ pour $i \in \mathrm{QUAL}$ et calcule la clé publique comme $\prod_{i \in \mathrm{QUAL}} A_{i,0} = g^{f(0)} \bmod p$ si on note $f(X) = \sum_{i \in \mathrm{QUAL}} f_i(X)$. La part de la clé obtenue par le participant P_j est égale à

$$\sum_{i \in \mathrm{QUAL}} s_{i,j} = f(j) \bmod q$$

  La clé secrète s est partagée sous forme polynomiale avec $f(j) \bmod q$ et sous forme additive avec $x_j \bmod q$ entre tous les participants appartenant à l'ensemble QUAL.

6.2.4 Preuve de sécurité

Dans cette section, on prouve la sécurité du schéma suivant le modèle de sécurité défini dans la section 6.2.1. On doit prouver l'exactitude et la sécurité du schéma.

Exactitude signifie que tous les joueurs obtiennent la même clé à la fin du protocole, que $t+1$ parts correctes permettent de retrouver la clé secrète et que la valeur secrète est uniformément distribuée dans le sous-groupe généré par g modulo p.

Sécurité signifie qu'aucune information sur x ne peut être apprise par l'adversaire, excepté ce qui suit de l'équation $y = g^x \bmod p$.

Théorème 23. Le schéma partagé est correct contre les adversaires actifs.

Preuve: Faisons comme hypothèse l'existence d'un adversaire $\mathcal{A}$ capable d'attaquer t serveurs.

Exactitude. Il est clair qu'à la fin du protocole, chaque serveur obtient la même clé publique. En effet, chaque serveur honnête a reçu la même information et déduit le même ensemble QUAL.

A la fin du protocole, la valeur secrète est partagée sous forme polynomiale de telle sorte que chaque ensemble de $t+1$ parts correctes permet d'interpoler le polynôme f de degré t dont le coefficient constant représente la clé secrète s.

Finalement, la clé publique y est uniformément distribuée dans le sous-groupe $\langle g \rangle$: si un des serveurs honnêtes n'est pas disqualifié et a choisi sa part additive x_i au hasard, le secret $\sum_{i \in \mathrm{QUAL}} x_i \bmod q$ est uniformément distribué dans $\mathbb{Z}_q$. Par conséquent, la valeur y est uniformément distribuée dans $\langle g \rangle$. □

Théorème 24. Sous l'hypothèse de résiduosité composite et dans le modèle de l'oracle aléatoire, le schéma partagé est sûr contre des adversaires statiques.

Preuve:

Sécurité. Nous décrivons un simulateur $\mathcal{S}$ qui prend comme entrée un élément $y \in \mathbb{Z}_p^*$ dans le sous-groupe généré par g et produit en sortie une distribution polynomialement indistinguable pour l'adversaire $\mathcal{A}$ d'une exécution correcte du protocole qui retourne y comme clé publique.

Soit $\mathcal{A}$ l'adversaire qui connaît y dans la phase **A1** et corrompt t serveurs au début du protocole dans la phase **A2**. L'adversaire $\mathcal{A}$ apprend tous leurs secrets et contrôle activement leur comportement.

Ici, nous tirons avantage du réseau synchronisé. Nous devons simuler la distribution de tous les serveurs. Cependant, quand nous simulons l'exécution du protocole, la synchronisation n'est pas nécessaire et nous pouvons attendre que tous les joueurs malicieux jouent. Ceci nous permet de déterminer la valeur publique y_i^* d'un joueur spécifique P_i^* pour lequel nous ne connaissons pas son état interne.

Chaque serveur P_i, excepté P_i^*, choisit au hasard $x_i \bmod q$ et t autres valeurs $a_{i,k}$ pour $k = 1, \ldots, t$. Il fixe $f_i(X) = \sum_{k=0}^{t} a_{i,k} X^k \in \mathbb{Z}_q[X]$. Puis, il calcule $A_{i,k} = g^{a_{i,k}} \bmod p$ pour $k = 0, \ldots, t$ et calcule $y_{i,j} = g^{f_i(j)} \bmod p$ et $Y_{i,j} = G_j^{f_i(j)} u_{i,j}^{N_j} \bmod N_j^2$. La distribution de ces valeurs et la distribution de celles du protocole réel sont les mêmes.

Maintenant, nous devons simuler la distribution du joueur P_i^*. Si nous voulons que la valeur finale soit y, posons $y_i^* = A_{i,0}^* = y \cdot \prod_{i \in \mathrm{QUAL} \setminus \{P_i^*\}} (y_i)^{-1} \bmod p$, car l'ensemble QUAL est défini à la fin de la synchronisation. Nous choisissons au hasard t valeurs $f_i^*(i_j)$ pour les t serveurs corrompus $\{i_1, \ldots, i_t\}$ et nous envoyons ces valeurs à ces serveurs. D'après la formule d'interpolation de Lagrange, nous pouvons calculer les valeurs publiques $y_{i,j}^*$ de toutes les autres part comme :

$$y_{i,j}^* = g^{f_i^*(j)} = (y_i^*)^{\lambda_{j,0}^S} \cdot \prod_{j=1}^{t} g^{\lambda_{j,i_j}^S f_i^*(i_j)}$$

où $\lambda_{i,j}^S = \prod_{j' \in S \setminus \{j\}} \frac{i-j'}{j-j'}$ et $S = \{0, i_1, \ldots, i_t\}$.

Remarques. On remarque que nous utilisons une hypothèse qui semble plus faible que l'hypothèse utilisée pour prouver la sécurité sémantique du cryptosystème de Paillier. En effet, l'hypothèse que nous faisons est qu'il est difficile de savoir si $Y(= G^x u^N \bmod N^2)$ chiffre la clé secrète de $y(= g^x \bmod p)$. La sécurité sémantique dit qu'il est difficile de savoir si $Y(= G^x u^N \bmod N^2)$ chiffre x ou non. Pour simuler $Y_{i,j}^*$, on peut par conséquent choisir au hasard $x_{i,j} \in \mathbb{Z}_{N_j}$ et $u_j \in \mathbb{Z}_{N_j}^*$, et fixer $Y_{i,j}^* = G_j^{x_{i,j}} u_j^{N_j} \bmod N_j^2$, mais on ne sait pas si le $x_{i,j}$ chiffré dans $Y_{i,j}^*$ correspond à celui qui est dans $y_{i,j}^*$.

Le joueur P_i^* a un état interne inconsistant parce qu'il ne connaît pas le logarithme discret de y_i en base g modulo p. Cependant, ce joueur ne sera pas attaqué parce que dans notre modèle, les serveurs corrompus sont choisis au début du jeu A.

Finalement, dans la simulation, la distribution produite par le simulateur est statistiquement proche de la distribution uniforme. Dans le modèle de l'oracle aléatoire, où le simulateur a un contrôle complet des valeurs retournées par la fonction de hachage H, on définit la valeur de H au point $(g, G, y, Y, g^z y^{-e}, G^z w^N Y^{-e})$ comme étant e. Avec probabilité écrasante, le simulateur n'a pas déjà définit l'oracle aléatoire en ce point, et donc l'adversaire $\mathcal{A}$ ne peut pas détecter la triche. □

6.2.5 Complexité du protocole

Tous les serveurs doivent exécuter $n \times (n-1)$ vérifications de la forme $y_{i,j} = \prod_{k=0}^{t} A_{i,k}^{j^k} \bmod p$ pour $1 \leq i \leq n$ et pour $1 \leq j \leq n$ sauf pour les n parts qu'ils ont eux-mêmes générées. Finalement, chaque serveur doit déchiffrer sa part du secret x.

Dans ce cas, P_i a calculé les $t+1$ valeurs $A_{i,k} = g^{a_{i,k}} \bmod p$ et les $3n$ valeurs $s_{i,j} = f_i(j)$, $y_{i,j} = g^{s_{i,j}} \bmod p$, et $Y_{i,j} = G_j^{s_{i,j}} u_{i,j}^{N_j} \bmod N_j^2$.

Pour les preuves, chaque serveur P_j doit vérifier si $e_{i,j} \stackrel{?}{=} H(g, G_j, y_{i,j}, Y_{i,j}, g^{z_{i,j}} y^{-e_{i,j}} \bmod p, G_j^{z_{i,j}} w_{i,j}^{N_j} Y_{i,j}^{-e_{i,j}} \bmod N_j^2)$ et $y_{i,j}^q \stackrel{?}{=} 1 \bmod p$ pour chaque $1 \leq j \leq n$ et pour chaque $1 \leq i \leq n$ sauf pour les preuves générées par P_i.

Dans ce cas, P_i doit calculer les preuves $(e_{i,j}, z_{i,j}, w_{i,j})$ où la part importante de calcul consiste à calculer $w_{i,j} = s_{i,j} u_{i,j}^{e_{i,j}} \bmod N_j$.

Finalement, chaque serveur déchiffre sa propre part x_j du secret commun x. Pour faire cette opération de manière efficace, le serveur P_j calcule le produit $Y_j = \prod_i Y_{i,j} = G_j{}^{\sum_i x_{i,j}} \bmod N_j{}^2$ en utilisant $n-1$ multiplications et exécute un seul déchiffrement de Y_j pour retrouver $x_j \bmod N_j$. Cette opération consiste en une seule exponentiation comme on l'a vu dans la section 4.2.5. De plus, comme $x_j = \sum_i x_{i,j}$ est majoré par $n^{n+2} \times q$, et $n \times \log_2(qn) < \log_2(N_j)$ pour tout j, la part $x_j \bmod N_j$ est égale à x_j.

La complexité de notre schéma est en $O(kn^2)$ exponentiations modulaires où n est le nombre de serveurs et k un paramètre de sécurité alors que la complexité de [90, 35] est en $O(kn)$. Cependant, pour un petit nombre de participants, notre protocole est plus efficace. Dans la sous-section suivante, nous présentons une amélioration du coût de calcul quand n devient grand et nous montrons que la complexité est du même ordre de grandeur que les schémas précédents.

6.2.6 Améliorations

Réseau Asynchrone

Dans certains cas, l'hypothèse d'avoir un réseau synchronisé peut apparaitre comme étant une exigence trop forte. Nous avons besoin de tel réseau pour contrer les attaques par "rushing". Une solution simple est de forcer un joueur particulier "incorruptible" à jouer à la fin du protocole. Nous appelons

un tel joueur une tierce partie incorruptible (TPI) parce que nous n'avons pas besoin que ce soit une partie de confiance, mais seulement qu'elle soit honnête. Au début, ce serveur choisit une valeur aléatoire a dans $\mathbb{Z}_q$ et met en gage $g^a h^b$ en utilisant la fonction de mise en gage de Pedersen. Les serveurs jouent et calculent $y' = g^{x'}$. A la fin, le TPI broadcast a et b à tous les joueurs. La valeur secrète est $x = x' + a \bmod q$. Chaque serveur peut calculer sa part du secret comme étant $x_i = {x'}_i + a \bmod q$. Ceci est dû à l'interpolation de Lagrange car la somme des coefficients de Lagrange est égale à 1. En effet, si S est un sous-ensemble de cardinalité $t+1$,

$$f(0) = \sum_{i \in S} \lambda_{i,0}^S f(i) \bmod q$$

Par conséquent, si nous partageons le polynôme constant valant 1 pour tout x, la valeur en 0 est toujours 1 et nous obtenons $1 = \sum_{i \in S} \lambda_{i,0}^S 1$. Par conséquent, si nous écrivons $f(i) = {x'}_i$, nous avons :

$$f(0) + a = \left(\sum_{i \in S} \lambda_{i,0}^S f(i) \right) + a = \sum_{i \in S} \lambda_{i,0}^S (f(i) + a) \bmod q$$

et la part secrète de chaque serveur est bien ${x'}_i + a$. Enfin, on peut simuler le joueur TPI car le simulateur peut connaître le logarithme de h en base g.

Note. Dans ce modèle, les simulations contre des adversaires adaptatifs sont faciles car nous pouvons fixer la valeur du joueur TPI dans le modèle de l'oracle aléatoire. Nous pouvons le voir comme le "joueur inconsistant de manière permanente" de [110].

Amélioration de la complexité

Quand le nombre de serveurs est relativement petit, notre protocole est pratique. Cependant, il peut devenir non-efficace quand le nombre n de serveurs augmente. Ici, nous fournissons des méthodes pour réduire le coût de calcul dans cette situation. Nous présentons la complexité calculatoire en terme de multiplications et nous montrons qu'asymptotiquement le coût de notre protocole a le même ordre de grandeur que les autres, *i.e.* $O(n^3 \log(n))$ multiplications. Ce résultat peut être atteint en réduisant toutes les vérifications à trois calculs qui sont utilisés dans tous les protocles. Ce dernier calcul représente la part principale de la complexité et ne peut pas être évitée dans la phase de vérification de l'interpolation.

De plus, comme les mauvais serveurs n'apparaissent que dans de rares situations, nous avons pour but de concevoir des protocoles efficaces quand tous les joueurs sont honnêtes. Cependant, nous devons être capables de détecter si de mauvais joueurs tentent de tricher et par conséquent, nous devons détecter rapidement les serveurs malicieux et actifs. Quand un tel serveur est détecté, nous avons besoin d'exécuter le protocole de la section 6.2.5 ou de "rebooter" le système car notre protocole est sans état[56].

La première remarque est que nous ne pouvons pas éviter dans la complexité le facteur $O(n^2 k)$, où k est la longueur en bit de $|p|$ ou $|N_j|/2$. En effet, d'un point de vue de la communication, en un seul tour, tous les serveurs doivent être capables de tester si les autres serveurs ont correctement joué. Mais, comme les réseaux d'aujourd'hui ont une grande bande passante et des performances très élevées, le goulot d'étranglement n'est plus la charge de communication mais le temps de calcul. Par conséquent, notre objectif principal est de diminuer la complexité en temps de calcul pour détecter les mauvais serveurs.

Il est simple de voir que le plus grand facteur de complexité provient des vérifications de $g^{f_i(j)} = \prod_{k=0}^{t} A_{i,k}^{j^k} \bmod p$. Nous devons vérifier n^2 telles équations alors que les précédents schémas n'en ont que n. Dans cette section, nous montrons comment réduire ce calcul pour chaque joueur à $3n$ vérifications.

56. Un protocole est dit *sans état*, s'il n'a pas besoin de stocker des valeurs entre chaque exécution du protocole.

Bellare, Garay et Rabin dans [7] ont décrit des algorithmes pour vérifier rapidement plusieurs signatures ou preuves. Le lecteur trouvera ces méthodes dans l'annexe 11.

Application des batch signatures. Avant tout, remarquons que vérifier si les n^2 valeurs $y_{i,j}$ chiffrent $g^{f_i(j)}$, exige n^2 exponentiations. En fait, il n'est pas nécessaire de tester exactement si ces équations sont correctes mais plutôt si un serveur envoie de mauvaises parts. Evidemment, chaque serveur doit vérifier ses propres parts mais peut seulement vérifier si les autres sont correctes avec forte probabilité.

A cette fin, on peut utiliser le RANDOM LINEAR COMBINATION TEST décrit dans l'annexe 11. On a besoin d'exécuter cet algorithme "dans les exposants" et la preuve provient du fait que g est un élément primitif dans le sous-groupe de $\mathbb{Z}_p^*$ d'ordre q. Dans notre situation, nous avons les valeurs $\alpha_{i,j}$ correspondant à $y_{i,j}$ et les valeurs β_i correspondant à i.

Cet algorithme fonctionne de la manière suivante :

1. Choisir $r \in_R \mathbb{Z}_q$
2. Calculer $\gamma_i = \alpha_{n,i}^{r^n} \times \ldots \times \alpha_{1,i}^{r}$. Ceci peut être calculé efficacement à partir des valeurs $y_{i,1}, \ldots, y_{i,n}$ en utilisant n fois l'algorithme FastMult décrit dans l'annexe 11.
3. Si $\mathrm{DEG}_{\mathbb{Z}_q,t,(1,\ldots,n)}(\gamma_1, \ldots, \gamma_n) = 1$, alors retourne "correct", et "incorrect" sinon.

Le DEG consiste à vérifier si pour tout $j = 1, \ldots, n$, si $g^{F(j)} = \prod_{k=0}^{t} A_{F,k}^{j^k}$ est égal à γ_j. Les valeurs $A_{F,k}$ correspondent aux coefficients du polynôme F et sont égales à $g^{\sum_{i=1}^{n} a_{i,k} r^i} = \prod_{i=1}^{n} A_{i,k}^{r^i}$. En fait, γ_j est égal à $y_{n,j}^{r^n} \times \ldots \times y_{1,j}^{r} = g^{f_n(j) r^n + \ldots + f_1(j) r} = g^{\sum_{i=1}^{n} r^i f_i(j)} = g^{F(j)} \bmod p$.

Par conséquent, nous devons calculer les n valeurs γ_j, les $t+1$ valeurs $A_{F,k}$ et les n relations DEG.

Cet algorithme échoue avec probabilité au plus $\frac{n}{q}$ qui est une quantité negligeable.

Comme d'habitude, on estime t à $n/2$ et la complexité du schéma en nombre de multiplications est :

1. Pour calculer les valeurs γ_j, n appels à l'algorithme FastMult pour n produits de puissance, où la taille des exposants est en $n|q|$; ainsi, $n \times [n|q| + \frac{n}{2} n|q|] = O(n^3|q|)$ sont nécessaires.
2. Pour calculer les coefficients de $g^{F(x)}$, $(t+1)$ appels à l'algorithme FastMult pour n produits de puissance, où la taille des exposants est en $|q|n$; ainsi, $(t+1) \times [|q|n + \frac{n^2}{2}|q|] = O(n^3|q|)$ sont nécessaires.
3. Pour vérifier la relation DEG, n appels à l'algorithme FastMult pour $(t+1)$ produits de puissance, où la taille des exposants est en $t \log(n)$; ainsi $n \times [t \log(n) + \frac{t(t+1)}{2} \log(n)] = O(n^3 \log(n))$ sont nécessaires.

Cet algorithme nécessite donc $O(n^3|q| + n^3 \log(n))$ multiplications.

Les algorithmes précédemment proposés ne peuvent pas utiliser cette astuce car ils ont seulement une valeur dans tous les ensembles S_i. Par conséquent, chaque serveur j doit exécuter n appels à l'algorithme FastMult pour vérifier si $y_j = \prod_{k=0}^{t} A_{i,k}^{j^k}$ pour $i = 1$ à n. Ceci conduit à n fois $t+1$ produits de puissance où la taille des exposants est en $t \log(n)$; ainsi, $n[t \log(n) + \frac{t(t+1)}{2} \log(n)] = O(n^3 \log(n))$. Si nous avions utilisé cette méthode au lieu de RANDOM LINEAR COMBINATION TEST "dans les exposants", la complexité aurait été de $O(n^4 \log(n))$.

En général, $|q| = 160$ et si nous prenons $n = 32 = 2^5$, alors $n \log(n) = 160$. Par conséquent, quand le nombre de serveurs est supérieur à 32, notre méthode de vérification par batch devient plus efficace que la méthode standard .

Afin d'être complet, nous fournissons ici une estimation de la complexité des preuves. La vérification des n^2 preuves a une complexité negligeable en comparaison de l'opération précédente. Nous pouvons

essentiellement résumer les preuves en vérifiant si la valeur précalculée g^r est égale à $g^z \times y^{-e} \bmod p$ et G^r est égal à $G^z w^N Y^{-e} \bmod N^2$. Nous devons exécuter n^2 produits de deux ou trois nombres où la taille des exposants est en $|A|$ ou $|N|$. Par conséquent, si nous appelons l'algorithme FastMult, nous obtenons une complexité en $|A|n^2$ multiplications pour les vérifications dans $\mathbb{Z}_p^*$ et en $|N|n^2$ multiplications pour les preuves dans N^2. Par conséquent, la complexité en temps de calcul est majorée par le temps d'exécution de l'algorithme Random Linear Combination Test dans les exposants pour tous les schémas.

6.3 Conclusion

Dans ce chapitre, nous avons proposé un schéma de partage de la phase de génération d'une clé cryptographique d'un cryptosystème basé sur le problème du logarithme discret. Nous avons décrit un algorithme ne nécessitant que des canaux publics et qu'un seul tour de communication. Alors que le coût en communication a diminué, la complexité a augmenté, mais si le nombre de serveurs est réduit, la surcharge n'est pas significative.

Notre approche a été de simplifier les travaux précédents qui avaient un but différent. Gennaro *et al.* ont voulu sécuriser ce protocole en vue de l'incorporer dans d'autres protocoles, en conséquence, la finalité d'une génération de clé a été oubliée. Dans le cas d'une simple génération de clé, leurs solutions sont moins efficaces que notre schéma. En effet, leur protocole nécessite la création d'un canal secret entre chaque paire d'utilisateurs. En revanche, si on souhaite utiliser un protocole de génération de clé pour proactiver l'algorithme de signature DSS par exemple, il faut considérer plus de critères et leur algorithme est efficace. Ainsi, chaque protocole présente des avantages et des inconvénients en fonction de l'application.

Cet algorithme permet donc de partager *complètement* un algorithme distribué utilisant des clés Diffie-Hellman car ces protocoles ont besoin de clés qui ne sont pas d'un format spécial comme dans le cas des clés de type RSA.

Troisième partie

Applications

7

Application à la loterie et au vote électronique

Dans ce chapitre, nous décrivons un protocole de loterie électronique et un protocole de vote électronique. Le premier a été présenté à Financial Crypto '00 [72] avec Guillaume Poupard et Jacques Stern et le second à PODC '01 [1] avec Olivier Baudron, David Pointcheval, Guillaume Poupard et Jacques Stern. Ces deux schémas sont des applications du partage du cryptosystème de Paillier. En effet, une autorité de loterie ou de vote électronique est un élément critique de ces protocoles et doit donc être distribuée pour garantir la confiance dans le système. En outre, les techniques de partage *complet* de RSA, étudiées au chapitre 3, permettent de partager *complètement* les systèmes de Paillier à seuil vus aux chapitres 4 et 5, et fournissent des solutions sûres pour nos protocoles de loterie et de vote électronique.

L'intérêt du cryptosystème de Paillier, par rapport aux autres systèmes de chiffrement ayant des propriétés d'homomorphisme, est d'avoir la plus grande bande passante. De plus, ce cryptosystème admet une version partagée comme on l'a vu dans les chapitres précédents. En effet, il est difficile de partager Okamoto-Uchiyama ou Naccache-Stern, car le déchiffrement doit être effectué modulo un secret partagé, ce qui est actuellement hors de portée des algorithmes distribués efficaces d'aujourd'hui. Le schéma de loterie électronique utilise le schéma du chapitre 5 alors que le schéma de vote électronique utilise le schéma du chapitre 4.

Le système de vote électronique a été implanté dans le cadre du contrat MASC[57] signé entre le laboratoire d'Informatique de l'École Normale Supérieure et France Télécom.

Sommaire

57. Mécanismes d'Anonymat pour la Signature, et le Chiffrement

7.1 Un schéma de loterie électronique

Le but d'une loterie est de générer un nombre aléatoire indiquant un ticket gagnant dans l'intervalle $\{0, \ldots, \ell - 1\}$ où ℓ est le nombre de tickets vendus. Dans un tel système, un ou plusieurs joueurs sont choisis comme gagnants en utilisant un processus tel que chaque ticket acheté a une même chance d'être choisi. Ce processus est surveillé par un juge arbitre qui garantit l'équité du jeu. Comme le processus est aléatoire, il ne peut pas être répété et les acheteurs doivent alors faire confiance au processus.

7.1.1 Principe

La première phase d'un schéma de loterie est la phase d'enregistrement durant laquelle chaque joueur reçoit un identifiant (qui peut même être un pseudonyme si l'anonymat des joueurs est nécessaire) et une paire de clés privée et publique.

Puis vient la phase d'achat des tickets. Chaque participant i à la loterie génère un aléa r_i, ajoute son certificat et son ordre de paiement et signe ce message

$$\text{message}_i = (r_i \| \text{certificat } i \| \text{ordre de paiement})_{\text{signature } i}$$

A la réception de ce message, la loterie vérifie la signature, et retourne le précédent message horodaté indiquant que la demande a été prise en compte avec un numéro de séquence $s_k \in [0, \ell[$ où ℓ représente le nombre total de tickets vendus.

$$\text{ticket}_k = (\text{message}_i \| s_k \| \text{date})_{\text{signature loterie}}$$

Ce message de retour correspond au k-ième ticket.

Enfin pendant la phase de calcul du ticket gagnant, la loterie calcule le ticket gagnant. Pour ce faire, tous les aléas issus des tickets valides sont injectés dans une fonction qui retourne un nombre uniformément distribué dans l'intervalle $[0, \ell[$. Cet entier indique le ticket gagnant.

Toute la difficulté de la conception d'un protocole de loterie réside dans la construction de la fonction de calcul du ticket gagnant. Nous décrivons dans le paragraphe suivant deux exemples de fonctions.

7.1.2 Exemple de réalisation

La fonction de calcul du ticket gagnant de Goldschlag et Stubblebine

Un premier système de loterie fut proposé par Goldschlag et Stubblebine dans [97]. Il utilise des *fonctions délai* pour éviter le calcul du résultat de la loterie par une personne avant la fin de la phase de vente des tickets. Dans ce protocole, tous les joueurs ont accès aux entrées aléatoires de la fonction délai et l'hypothèse faite sur ces fonctions est que personne ne peut calculer la sortie avant une certaine date dans le futur. Un exemple de fonction délai est $f(k) = 2^{2^k} \bmod N$ où $N = pq$ et k est un paramètre qui permet de fixer le temps dans le futur. Si la factorisation de N est connue, on peut calculer $\delta = 2^k \bmod \varphi(N)$ et $2^\delta \bmod N$ en temps $O(N)$. Sinon, la seule méthode connue est la méthode *séquentielle* naïve qui consiste à calculer $f(i) = 2^{2^i} \bmod N$ et $f(i+1) = (f(i))^2 \bmod N$ pour $i = 1$ à k. Ainsi, on peut estimer le paramètre de sécurité k, en fonction du temps mis à calculer un carré modulo

N. Comme ce calcul ne peut pas être distribué, on ne peut pas mettre plusieurs machines pour tenter de calculer la k-ième itération. Le calcul de cette fonction a été proposé comme challenge par Ron Rivest dans `http://theory.lcs.mit.edu/~rivest/lcs35-puzzle-description.txt`. Il s'agit essentiellement du générateur de Blum-Blum-Shub. La période de ce générateur pseudo-aléatoire est en $\lambda(\lambda(N))$.

Une nouvelle fonction pour le calcul du ticket gagnant

Contrairement au système de Goldschlag et Stubblebine, où l'hypothèse de sécurité repose sur le fait que pour calculer certaines fonctions, il n'y a pas d'autres algorithmes plus efficaces que ceux connus, la sécurité de notre schéma de loterie est basée seulement sur des hypothèses standards. Comme le schéma précédent, la loterie utilise des nombres aléatoires choisis par les joueurs afin de retourner un nombre imprédictible si au moins un joueur choisit son nombre au hasard.

Au lieu d'envoyer r_i en clair, chaque participant i chiffre r_i avec la clé publique de l'autorité de loterie du cryptosystème homomorphique de Paillier et obtient $c_i = \mathcal{E}(r_i, u) = g^{r_i} u^N \bmod N^2$. Ainsi, personne, exceptée la loterie, ne peut apprendre les entrées aléatoires. La loterie calcule $c = \prod_i c_i = g^{\sum_i r_i} v^N \bmod N^2$ (on ne se préoccupe pas des valeurs prises par le paramètre de randomisation v^N) et en déchiffrant c de manière partagée, obtient $s = \sum_i r_i \bmod N$. Enfin, elle désigne le ticket gagnant en calculant $s \bmod \ell$. Le processus de déchiffrement est partagé entre les serveurs de telle sorte que la loterie puisse calculer le résultat final même si t serveurs parmi les n essaient de retrouver le résultat avant la fin de la phase d'achat des tickets.

La valeur $(\sum_i r_i \bmod N \bmod \ell)$ correspond à un nombre aléatoire généré par la loterie quand au moins un joueur envoie une valeur aléatoire. De plus, comme on utilise un processus de déchiffrement à seuil publiquement vérifiable, les opérations exécutées par les serveurs peuvent être vérifiées par tous les joueurs ainsi que la prise en compte d'un ticket dans le calcul du ticket gagnant.

Équirépartition du tirage

L'exigence de sécurité cryptographique du protocole de loterie est que la distribution de la fonction de calcul du ticket gagnant soit uniforme et imprédictible. Posons $x = \sum r_i$. Le nombre $(x \bmod N) \bmod \ell$ est uniforme dans $[0, \ell[$ si ℓ divise N[58]. Sinon, pour assurer que la probabilité $x \in [0, \ell[$ soit uniforme, on vérifie si

$$x - (x \bmod \ell) + \ell \overset{?}{<} N \tag{7.1}$$

Si cette inégalité est vérifiée, le ticket gagnant est alors désigné en calculant $s \bmod \ell$. Sinon, il faut rejeter la valeur de x. En effet, si ℓ ne divise pas N et si on découpe l'intervalle $[0, N[$ en blocs de taille ℓ, le dernier bloc ne sera pas de taille ℓ. Ainsi, si la valeur de x ne tombe pas dans le dernier intervalle, x tombe dans un bloc de longueur ℓ et est uniformément distribué dans cet intervalle. Par conséquent, la valeur $x \bmod \ell$ sera elle aussi uniformément distribuée. Si la valeur de x tombe dans le dernier intervalle, la probabilité que x soit dans $[0, N - \ell \times \lfloor \frac{N}{\ell} \rfloor[$ est plus importante que la probabilité que x soit dans $[N - \ell \times \lfloor \frac{N}{\ell} \rfloor, \ell[$. On doit donc rejeter la valeur de x si x est dans l'intervalle $[\ell \times \lfloor \frac{N}{\ell} \rfloor, N[$.

Pour résoudre ce problème, on peut considérer la loterie comme un joueur particulier qui randomise le jeu. De la même façon que le joueur honnête du chapitre 6, on peut considérer que la loterie choisit deux valeurs aléatoires a_1 et a_2 telles que $|a_1 - a_2| > \ell$, où $|x|$ représente ici la valeur absolue de x, et les met en gage au début du jeu en calculant $c_1 = H(a_1)$ et $c_2 = H(a_2)$, où H est une fonction de hachage. Au moment de la phase de calcul, la loterie défait la mise en gage c_1 et calcule $s_1 = s + a_1 \bmod N$. Si la valeur de s_1 ne vérifie pas l'équation 7.1, la loterie défait la mise en gage c_2. Comme la distance entre

58. Ce qui n'arrive jamais car $N = pq$.

a_1 et a_2 est supérieure à ℓ, et on sera sûr que s_2 ne vérifie pas l'équation 7.1 car si s_1 est dans l'intervalle $[\ell \times \lfloor \frac{N}{\ell} \rfloor, N[$, la valeur de s_2 sera en dehors de cet intervalle.

Enfin, on peut calculer la probabilité de l'événement $x \in [\ell \times \lfloor \frac{N}{\ell} \rfloor, N[$ qui vaut $\frac{\ell}{N} < 2^{-990}$ si ℓ vaut un milliard (2^{30}) et N est un module de 1024 bits. Ainsi, cette probabilité est négligeable et il n'est pas nécessaire de prendre deux valeurs mises en gage par la loterie. Cependant, ces mises en gage peuvent aussi être utiles pour désigner plusieurs gagnants $s_i = s + a_i \bmod \ell$.

Nécessité d'un chiffrement IND-CCA

Considérons l'attaque suivante :

1. c' est un chiffré connu de l'attaquant correspondant au produit de tous les chiffrés.
2. $\mathcal{A}$ joue c/c' avec $c = g^s t^N \bmod N^2$. Donc, c est le nouveau produit des chiffrés et $\mathcal{A}$ sait que le clair correspondant vaut s. $\mathcal{A}$ reçoit s_k de la part de la loterie indiquant le numéro de séquence.
3. Enfin, $\mathcal{A}$ joue $g^r v^N \bmod N^2$ tel que s_k gagne.

Dans notre protocole de loterie, tous les joueurs peuvent calculer le chiffré de la somme des randoms $c = g^s v^N \bmod N^2$. A la fin du protocole de loterie, ils apprennent que c est le chiffré de s. Si 5 minutes avant la clôture, le produit des chiffrés vaut c', un attaquant peut alors acheter un ticket avec le chiffré $c/c' \bmod N^2$. Le nouveau résultat de la loterie sera $c' \times \frac{c}{c'} = c \bmod N$. Le joueur sait que c correspond à la somme s car l'autorité révèle ce nombre et pas uniquement $(s \bmod \ell)$ pour que le processus de vérification de la loterie soit publiquement vérifiable. De même, n'importe quel utilisateur peut vérifier le déchiffrement partagé de c et peut apprendre s. Ainsi, ce joueur peut racheter un ticket en chiffrant son aléa r tel que $(r + s \bmod N) \bmod \ell$ tombe sur le numéro de séquence précédent que lui a donné la loterie en retour de son premier ticket. Si ce participant est le dernier joueur, il aura alors gagné la loterie.

Cette attaque fonctionne car la loterie se comporte comme un oracle de déchiffrement. Pour contrer cette attaque, il faut imposer aux participants de fournir une preuve qu'ils connaissent le nombre qu'ils ont chiffré. Pour ce faire, on peut utiliser un chiffrement IND-CCA décrit dans le chapitre 5 au lieu du cryptosystème de Paillier IND-CPA du chapitre 4.

7.2 Protocoles de vote électronique

Le but des schémas de vote électronique est de fournir un ensemble de protocoles qui permettent aux électeurs de voter et à un groupe d'autorités de collecter les votes et calculer les résultats du scrutin.

7.2.1 Exigences de sécurité

Les exigences de sécurité d'un protocole de vote sont les suivantes :

- Protection de la confidentialité des électeurs (anonymat des votants),
- Détection de fraudes des électeurs,
- Détection de fraudes du centre de vote.

La première exigence est nécessaire pour que personne n'apprenne le contenu du vote d'un électeur, c'est-à-dire que le schéma de vote doit cacher l'identité des votants ou, il doit être impossible à un observateur de lier une identité au contenu d'un vote. La deuxième exigence de sécurité correspond à une tentative de fraude des électeurs qui peuvent par exemple tenter de voter plusieurs fois ou tenter de

voter alors qu'ils ne sont pas autorisés, ou encore dupliquer le vote de quelqu'un d'autre. La troisième exigence de sécurité correspond à un centre qui triche sur le résultat du scrutin. Ainsi, tout observateur doit être capable de vérifier si le calcul des résultats a été correctement effectué.

Les exigences de sécurité d'un protocole dépendent de la méthode utilisée. Il existe plusieurs méthodes pour voter électroniquement. La première méthode utilise un isoloir électronique. Chaque électeur entre dans l'isoloir et appuie sur le bouton qu'il souhaite pour désigner son candidat. L'avantage de cette technique est d'éviter le dépouillement manuel et donc d'obtenir le résultat du scrutin à 20 H et pas un mois plus tard ! Le problème est qu'il faut garantir que les électeurs n'ont voté qu'une fois dans l'isoloir et ces derniers doivent tout de même se déplacer.

Ainsi, le vote devrait pouvoir être effectué électroniquement à distance. Il faut dans ce cas recréer les conditions et les caractéristiques de l'isoloir. La première question à se poser est celle de la fonction d'un isoloir. Il permet aux électeurs de *se cacher du regard des autres pour effectuer leur choix, de prendre une décision sans être menacés par une tierce personne*, et *de ne pas pouvoir vendre un vote*. Ainsi, en plus des exigences précédentes d'un schéma de vote, il faut aussi empêcher :

– la coercition des votants (on ne peut pas forcer un électeur à voter pour un candidat donné),

– l'achat de vote (un utilisateur ne peut pas exhiber un reçu prouvant qu'il a voté pour un candidat précis.).

Enfin, la dernière exigence que l'on peut imposer est l'anonymat des électeurs, c'est-à-dire des votants et non-votants : savoir si une personne a voté. L'anonymat des non-votants peut se résoudre en utilisant des pseudonymes[59].

Exigences pratiques de sécurité

Une exigence importante en pratique aujourd'hui est que l'outil utilisé pour voter ne peut pas être un ordinateur quelconque connecté au réseau public Internet car aucune garantie de sécurité sur la machine ou sur un réseau public ne peut être assurée. Le lecteur pourra lire l'article de Rivest[60] concernant la sécurité des élections présidentielles. Il est évident que le niveau de sécurité pour ce type d'application est plus important que pour un simple vote. Dans ce cas, un appareil électronique dédié devrait être utilisé ainsi qu'un réseau propriétaire. La solution consiste donc toujours à utiliser des isoloirs électroniques et le réseau interne du Ministère de l'Intérieur par exemple. Les produits de vote peuvent donc être informatisés mais ne peuvent pas appartenir aux particuliers, comme un PC relié à Internet car la sécurité des postes clients n'est pas suffisante. Certaines phases peuvent être informatisées de manière publique comme la phase d'enregistrement permettant d'obtenir une carte à puce pour voter (une carte d'identité ayant des fonctions cryptographiques pourrait aussi être utilisée) ou la publication des résultats sur Internet.

La seconde exigence importante en pratique est qu'un système de vote doit avoir une capacité de stockage importante. En effet, il est nécessaire de stocker les éléments qui permettent de vérifier ultérieurement les résultats d'un vote. Ainsi, les preuves cryptographiques des protocoles de vote doivent pouvoir être stockées et ne pas être trop encombrantes.

59. On peut remarquer que cette propriété n'est pas assurée aujourd'hui car n'importe quel utilisateur peut rester devant le bureau de vote et contrôler l'identité des votants. Cependant, l'informatisation de certaines procédures exige des précautions d'anonymat pour empêcher la création de fichiers nominatifs. De plus, ces informations pourraient être utiles aux différents candidats pour savoir exactement quelles sont les populations qui ne votent pas.

60. Il s'agit de remarques faites par Ron Rivest à propos du panel sur le vote électronique au congrès Financial Crypto '01. Elles sont disponibles à l'URL suivante :
`http://theory.lcs.mit.edu/~rivest/Rivest-ElectronicVoring.pdf`.

7.2.2 Techniques générales

L'étude des schémas de vote électronique [16, 17, 114, 53, 54, 169, 114, 106] a débuté par la thèse de Benaloh [16]. De nombreux schémas ont ensuite été proposés dont la plupart présente presque uniquement des schémas de type référendum ("oui/non"). Trois modèles de vote électronique ont été proposés jusque là pour garantir l'anonymat des électeurs.

Signature en blanc. Un schéma de signature en blanc [41] est un protocole qui permet à un Client d'obtenir la signature d'un Serveur sans révéler au Serveur le contenu du message signé. Par exemple, un schéma de signature en blanc pour RSA est le suivant : le Client génère un message m et le convertit en $m' = m \times r^e \bmod N$ en utilisant un aléa r et la clé publique (N, e). Le Serveur reçoit le message m' et génère une signature $s' = (m')^d \bmod N$ avec la clé privée correspondante (N, d). Le Client obtient la signature s sur le message m en calculant $s'/r = (m \times r^e)^d/r = m^d = s \bmod N$. Ainsi, le serveur ne connaît pas m à partir de m', et le Client obtient une signature sur le message m à partir de s'.

En utilisant un schéma de signature en blanc, il est possible de créer un schéma de vote électronique anonyme. Chaque électeur peut construire le message :

$$\text{message}_i = (vote \| R \| H(vote \| R))$$

où H est une fonction de hachage et R une chaîne aléatoire suffisamment longue choisie par chaque électeur. Pour un bulletin généré par un électeur, l'autorité (Serveur) émet une signature en blanc :

$$\text{bulletin}_i = (\text{message}_i)_{\text{signature de l'autorite}}$$

Ensuite, les électeurs envoient sur un *canal anonyme* le bulletin de vote au centre qui poste le bulletin sur le tableau public. Les signatures en blanc empêchent de lier le message envoyé à l'autorité de signature et le bulletin envoyé au centre. Ce schéma empêche aussi des électeurs non autorisés de voter. Chaque électeur peut vérifier si son vote a été pris en compte. Cette vérification ne peut pas être publique car c'est le centre qui met les votes dans le tableau public de bulletin et non les utilisateurs. En effet, ces derniers ne peuvent pas mettre le bulletin dans le tableau public car dans ce cas, le canal ne pourrait pas être public. Il est donc difficile pour une personne qui n'a pas vu un bulletin de vote de dire s'il a été pris en compte .

Pour éviter qu'une personne vote plusieurs fois, les conditions suivantes doivent être vérifiées :

- il ne doit pas exister plus d'un vote valide généré par signature en blanc. Il est possible de montrer que le schéma de signature en blanc RSA de Chaum [42] est résistant face à des falsifications existentielles [8]. Le votant peut obtenir toutes les signatures pour les messages de la forme $mR^e \bmod N$ à partir de s', où R est un random choisi au hasard par le votant. Par conséquent, le format des bulletins de vote valides doit être restreint de telle sorte que le message soit acceptable. C'est le but de la fonction de hachage à la fin du bulletin précédent, pour empêcher les falsifications supplémentaires [148]. Le schéma de signature en blanc RSA de Chaum [42] a récemment été étudié dans sa version "full-domain hash" [11, 8]. En effet, pour éviter les falsifications supplémentaires d'une signature en blanc, il faut empêcher les forges existentielles [101] de la signature qui en résulte. D'où la xredondance imposée par la fonction de hachage.

- Il ne doit pas y avoir de confusion entre deux votes d'un même votant et deux votes de deux électeurs différents. Si chaque votant peut obtenir un unique vote valide, le seul moyen d'envoyer des bulletins multiples est de dupliquer un vote. Si le processus de création d'un bulletin est déterministe, il est impossible de distinguer si les deux votes proviennent d'un même électeur ou de deux électeurs différents qui ont la même opinion. Par conséquent, il est nécessaire de randomiser les

votes et de laisser ouverte la possibilité à deux électeurs de générer le même vote mais avec probabilité négligeable sinon un électeur peut envoyer deux fois le même vote au centre de publication. La chaîne de random R dans le bulletin précédent est utilisée pour cela.

Enfin, ces protocoles supposent l'existence d'un *canal anonyme*, ce qui n'est pas le cas du réseau Internet. En effet, un canal anonyme efface l'origine des messages émis. La conséquence de l'utilisation de ce type de canal est que le vote n'est plus publiquement vérifiable car le votant ne peut pas écrire dans le tableau de bord où il est le seul à pouvoir le faire. S'il veut écrire dans le tableau, il doit s'authentifier auprès du système et comme le canal est anonyme, cette authentification ne peut pas se faire.

Réseaux de mélangeurs [41]. Un réseau de mélangeurs est une fonction qui, étant donnée une liste de chiffrés, produit une liste de déchiffrés, et qui cache les correspondances entre les déchiffrés de la liste de sortie et les chiffrés de la liste d'entrée. Avec une telle fonction, un schéma de vote peut facilement être construit. Chaque vote est soumis sous forme chiffrée. Le centre de vote collecte les votes chiffrés et les place dans une liste en entrée d'un réseau de mélangeurs. Ce dernier retourne une liste de déchiffrés, avec laquelle le centre de vote peut calculer le résultat.

Décrivons schématiquement un protocole de réseau de mélangeurs. Dans ce schéma, le réseau est composé de plusieurs centres mélangeurs. Chaque message chiffré dans la liste est successivement traité par les centres mélangeurs. Le dernier centre retourne un ensemble de déchiffrés randomisés et sans rapport avec l'ordre d'entrée. A un haut-niveau, le centre i traite chaque message envoyé par le centre $i-1$ (ou la liste d'origine, quand $i = 1$) et transmet les résultats dans un ordre permuté. Il reste à spécifier comment un message m est initialement chiffré et comment le i-ième centre traite chaque message.

Nous décrivons un système basé sur El Gamal. Les informations publiques sont $p = kq + 1$, où p et q sont deux nombres premiers, et $g = (g')^k \bmod p$, où g' est un générateur de $\mathbb{Z}_p^*$. La clé publique du i-ème centre est $y_i = g^{x_i} \bmod p$ et sa clé privée est $x_i \in \mathbb{Z}_q^*$. Soit $m \in \mathbb{Z}_p^*$ d'ordre q et définissons $w_i = y_{i+1}y_{i+2}\ldots y_n$ et $w_n = 1$.

Pour chiffrer un message m, le votant génère un random r_0 et envoie $Z_1 = (G_1, M_1) = (g^{r_0} \bmod p, (w_0)^{r_0} \times m \bmod p)$ au centre 1.

Le i-ème centre (pour $i = 1, \ldots, n-1$) traite en entrée le chiffré (G_i, M_i) : il génère un nombre aléatoire r_i (indépendamment pour chaque chiffré) et calcule le chiffré suivant en utilisant sa clé secrète x_i :

$$\begin{aligned} G_{i+1} &= G_i \times g^{r_i} \bmod p \\ &= g^{r_0+\ldots+r_i} \bmod p \\ M_{i+1} &= M_i \times {w_i}^{r_i} / {G_i}^{x_i} \bmod p \\ &= {w_i}^{r_0+\ldots+r_i} \times m \bmod p \end{aligned}$$

Il transmet $Z_{i+1} = (G_{i+1}, M_{i+1})$ (permutés avec les autres messages traités) pour être traités par le centre $i+1$. Le centre n peut retrouver m en calculant

$$m = M_n / {G_n}^{x_n} \bmod p$$

On obtient une vérification universelle dans le schéma précédent en exigeant que chaque centre prouve qu'il a correctement traité ses messages. Dès lors, n'importe quel observateur peut vérifier les preuves résultantes pour confirmer que les messages ont été gérés correctement par le réseau de mélangeurs.

Enfin, le centre i doit prouver aux autres serveurs qu'il a correctement permuté la liste d'entrée. Ce genre de preuve n'est pas efficace aujourd'hui. Par conséquent, les schémas de vote utilisant des réseaux de mélangeurs ne sont pas efficaces en pratique, même si de récents progrès sont apparus [85].

Chiffrement homomorphe. Un chiffrement homomorphe est un cryptosystème qui, en plus de cacher le contenu d'un vote, possède des propriétés supplémentaires. Par exemple,

$$\mathcal{E}(x+y) \equiv \mathcal{E}(x) \times \mathcal{E}(y)$$

Comme on l'a déjà dit dans cette thèse, ceci permet de faire des calculs avec des données chiffrées sans être obligé de déchiffrer. Considérons le cas du vote "oui/non" et posons que "oui" correspond à 1 et "non" à 0. Si on ajoute toutes les représentations numériques de chaque vote[61], on obtient le nombre total de "oui". Afin de rendre ce vote secret, on peut soumettre 1 ou 0 en utilisant un chiffrement homomorphe. Grâce à la propriété homomorphique, le produit des chiffrés est égal au chiffré de la somme des votes. Puis, l'autorité déchiffre la somme des votes chiffrés et ce protocole permet de calculer le résultat du vote sans révéler chaque vote individuellement.

Il reste deux problèmes à résoudre avec ce type de vote. Le premier est que l'autorité peut déchiffrer les votes des utilisateurs et connaître le vote d'un électeur. Le second est que les électeurs peuvent tricher et chiffrer des valeurs différentes de 0 ou 1, comme par exemple 100 ou -1000, ce qui donne un poids plus important à ce vote. Le premier problème est résolu en utilisant plusieurs autorités. Il y a deux solutions : soit le vote 0 ou 1 est partagé et chaque part est envoyée à un serveur différent (schémas de vote homomorphique de Benaloh), soit la clé de déchiffrement est partagée entre les autorités de façon à ce que la réunion d'un certain nombre d'entre elles soit nécessaire (schéma de Cramer, Gennaro et Schoenmakers). Le second est résolu en demandant à chaque votant de prouver qu'il a chiffré 0 ou 1 sans révéler son vote. Pour ce faire, on utilise des preuves zero-knowledge.

Dans le schéma de vote de Cramer, Gennaro et Schoenmakers d'Eurocrypt '97 [54], tous les électeurs envoient leur vote chiffré à un unique combineur. En utilisant la propriété homomorphique du cryptosystème, le combineur calcule le total chiffré de manière publiquement vérifiable. Ensuite, il l'envoie aux n autorités et $(t+1)$ parmi elles calculent le résultat du scrutin en exécutant un déchiffrement partagé. Ce modèle est optimal pour les communications entre les électeurs et les autorités.

Dans ce schéma, les auteurs utilisent une variante du schéma de chiffrement El Gamal. Au lieu de chiffrer m avec $(g^k \bmod p, my^k \bmod p)$, on peut calculer $(g^k \bmod p, g^m y^k \bmod p)$. Malheureusement, un tel schéma ne peut pas être considéré comme un schéma à trappe car aucune trappe n'existe pour déterminer m étant donné $g^m \bmod p$. Néanmoins, dans les schémas de vote du type "0/1", le cryptosystème gère seulement des petits nombres parce que le nombre de votants est limité et chaque votant vote "0" ou "1". Par conséquent, le total ne peut pas être vraiment très grand, de l'ordre de 2^{20}, et une recherche exhaustive ou en utilisant quelques améliorations (Baby Step-Giant Step) permet d'obtenir le résultat.

Chiffrement homomorphe multi-candidats. Les schémas de vote multi-candidats ont été introduits par Cramer, Frankel, Schoenmakers et Yung dans [53] et utilisés dans [54].

La solution pour construire un système de vote homomorphe gérant plusieurs candidats est la suivante. Soit ℓ le nombre d'électeurs et k tel que $\ell < 2^k$. Les électeurs votent "1" pour le premier candidat, "2^k" pour le deuxième, "2^{2k}" pour le troisième, et ainsi et suite. Il est facile de voir que les k premiers bits de poids faible donnent le résultat du premier candidat, les k bits suivants donnent le résultat du deuxième candidat, etc.

En effet, dans les schémas d'élection multi-candidats, le total est plus grand que dans un système "oui/non" car l'encodage de plusieurs candidats ne peut pas être réduit. Si nous voulons continuer à calculer les différents résultats, nous devons utiliser une taille d'encodage en $\mathcal{O}(p \times L(\ell))$, où p est le

61. Cette association entre candidat et représentation numérique fige le schéma de vote. On ne peut pas rajouter n'importe quel candidat au schéma de vote. On remarque que les deux propositions précédentes (signature en blanc et réseaux de mélangeurs) permettent des ajouts de candidats. Les élections américaines exigent que les électeurs aient le droit de voter pour qui ils veulent. Les schémas de vote homomorphe ne peuvent donc pas être utilisés tel quel.

nombre de candidats et $L(\ell)$ est la taille du nombre d'utilisateurs. Par exemple, une élection nationale peut mettre en jeu 10 candidats et des centaines de millions d'électeurs ont besoin de pouvoir chiffrer des messages de 266 bits. Dans de telles applications, le schéma El Gamal modifié ne peut plus être utilisé car la recherche exhaustive ou des techniques plus efficaces comme la méthode λ de Pollard ne permettent plus de retrouver efficacement le résultat[62]. Une solution consiste à utiliser un schéma basé sur un logarithme discret à trappe avec une grande bande passante comme le schéma Naccache-Stern [128], Okamoto-Uchiyama [136], ou celui de Paillier [139]. Par exemple avec le cryptosystème à seuil de Paillier où N représente la bande passante, on peut accepter jusqu'à $\lfloor L(N)/k \rfloor$ candidats, où $L(N)$ est la taille du module RSA.

Deux propositions autour du schéma de Paillier. Dans l'article [1], nous nous sommes intéressés à un système d'élection où de multiples candidats peuvent être gérés et nous avons utilisé le cryptosystème de Paillier qui possède un déchiffrement efficace ainsi que la plus grande bande passante parmi tous les cryptosystèmes à trappe de calcul du logarithme discret. Ces deux arguments sont les principaux éléments pour construire des schémas d'élection multi-candidats basés sur ce cryptosystème. Une version partagée de ce cryptosystème est apparue dans [72] et a été redécouverte, indépendamment mais plus tard, dans [56] par Damgård et Jurik.

Dans ce dernier article, les auteurs proposent un nouveau point de vue pour le schéma de Paillier et fournissent une autre version. Ils appliquent ce schéma à seuil pour construire un schéma de vote avec plusieurs candidats comme nous l'avons fait mais avec des preuves zero-knowledge différentes pour prouver que le vote est un élément d'une liste de valeurs connues. La complexité de leurs preuves est logarithmique en le nombre de candidats alors que la complexité des nôtres est linéaire. Néanmoins pour un nombre raisonable de candidats, les deux preuves zero-knowledge ont la même efficacité. En effet, dans notre système, une seule preuve efficace est calculée alors que dans le système de Damgård et Jurik, deux preuves sont nécessaires dont une preuve de multiplication, logarithmique elle aussi en le nombre de candidats, mais très pénalisante en terme de complexité. Ainsi, la constante cachée dans le $O(\log(p))$ est plus importante que dans notre protocole. De plus, leur article est différent du nôtre car nous construisons un système global et nous ne proposons pas uniquement des primitives cryptographiques. Nous avons une vision globale de toutes les caractéristiques de sécurité du système de vote. Enfin, notre solution prend complètement en compte l'anonymat et l'achat de vote.

7.3 Nouveau système de vote électronique

Le schéma présenté dans l'article [1] est basé sur le cryptosystème de Paillier et sur des techniques de preuves zero-knowledge. Il est efficace et flexible pour prendre en compte les différents niveaux de hiérarchie. Le nouveau schéma de vote garantit les exigences suivantes : l'anonymat des électeurs, la vérification publique du résultat, la robustesse du dépouillement et d'empêcher l'achat de vote. L'anonymat des électeurs assure qu'un vote sera gardé secret contre n'importe quelle coalition de t autorités où t est un paramètre du système. La vérification publique assure que n'importe quel observateur externe sera convaincu que l'élection est équitable et que le total publié a été correctement calculé à partir de tous les votes correctement formés. La robustesse du schéma assure ensuite que le système peut tolérer des autorités fautives qui essaient de tricher durant le calcul du total. Enfin, la propriété d'être sans reçu assure que l'électeur ne peut pas construire un reçu prouvant le contenu de son vote afin d'éviter l'achat de vote.

62. En effet, cette méthode permet de tirer partie du fait que le chiffré est dans un intervalle donné et n'utilise pas de mémoire.

7.3.1 Organisation de l'élection

Architecture

Dans cette section, nous présentons un *système orienté grand groupe* qui peut être utilisé pour des élections dans un pays comme la France. Dans ce cas, quelques contraintes organisationnelles doivent être prises en compte avec prudence.

Par exemple, le système suivant présente des élections "directes" dans un scénario de la vie réelle :

1. Les bureaux locaux effectuent la certification des utilisateurs et vérifient que les électeurs ne peuvent voter qu'une fois. Ils vérifient aussi la validité des votes. Cette tâche peut aussi être vérifiée par n'importe quel observateur.

2. Les bureaux locaux envoient les résultats locaux à leur centre régional. Ce dernier vérifie que les centres locaux envoient des informations correctes et calcule les résultats régionaux.

3. Tous les centres régionaux envoient les résultats régionaux au centre national qui calcule le résultat final et vérifie si les centres régionaux ont correctement exécuté leur tâche.

Acteurs

L'architecture met en jeu plusieurs entités à différents niveaux.

Candidats Dans le cas le plus simple, l'élection a pour but de choisir un gagnant parmi plusieurs candidats. Les candidats sont potentiellement des personnes physiques, un référendum (choix "oui/non"), ou n'importe quel ensemble arbitraire de propositions parmi lesquelles un choix doit être fait[63]. Ils sont dans la suite désignés par les nombres $\{1, 2, \ldots, p\}$ et peuvent inclure l'élément "nul".

Électeur Un électeur est une personne enregistrée qui peut voter secrètement pour un candidat. Chaque vote a le même poids dans le résultat final.

Autorité locale Au niveau local, les électeurs s'enregistrent auprès d'une autorité locale qui recueille les votes, peut calculer le résultat local et l'envoie au niveau régional.

Autorité régionale Une autorité régionale reçoit les résults locaux, calcule le résultat régional, et l'envoie à l'autorité nationale.

Autorité nationale Au niveau "racine", l'autorité nationale collecte les résultats régionaux, et publie, pour chaque candidat, le nombre total de votes.

Serveur de temps de confiance Ce joueur garantit qu'un électeur a voté avant une certaine date et heure.

63. La différence entre un schéma homomorphique et les deux autres schémas de vote (utilisant les signatures en blanc et les réseaux de mélangeurs) est que dans un schéma de vote homomorphe, les électeurs ne peuvent pas imposer un nom, mais doivent chiffrer une valeur numérique qui représente un choix. Ce type de protocole ne semble donc pas adapté aux élections américaines où la possibilité d'écrire le nom d'un candidat en dehors d'une liste prédéfinie est possible ainsi que pour les municipales dans les petites communes en France.

Modèle de communication

Le modèle de communication que nous utilisons repose sur un canal de broadcast public avec mémoire et peut être implémenté avec un tableau public [16]. Toutes les communications avec le tableau sont publiques et peuvent être universellement surveillées. Aucune entité extérieure ne peut effacer des informations mais chaque électeur peut inscrire sa part dans le tableau.

Pour contrôler la connexion entre les électeurs et le tableau public, un contrôle d'accès est utilisé.

Autres attaques sur un système de Vote

Notre système de vote prend en compte deux attaques apparaissant dans un système pratique.

Attaque des autorités intermédiaires Cette attaque met en jeu les autorités intermédiaires qui essaient de falsifier le résultat final de l'autorité nationale.

Attaque de "rushing" A la fermeture d'un système de vote, les autorités locales doivent révéler le total local. Le système peut être protégé pour résister à une attaque de "rushing" des utilisateurs qui essaient de fausser le total s'ils attendent le résultat de l'autorité locale pour voter.

7.3.2 Cryptosystème de Paillier

On rappelle rapidement les propriétés utiles de ce cryptosystème. Le lecteur se reportera au chapitre 4 pour plus de précisions. Ce cryptosystème probabiliste $\mathcal{E}(M)$ chiffre un message M en élevant une base g à la puissance M et en randomisant convenablement ce résultat. Une conséquence de ce type de chiffrement est que ce schéma est homomorphe.

Propriétés homomorphiques

Elles peuvent être établies de manière informelle de la manière suivante :

$$\mathcal{E}(M_1 + M_2) \equiv \mathcal{E}(M_1) \times \mathcal{E}(M_2) \quad \text{et} \quad \mathcal{E}(k \times M) \equiv \mathcal{E}(M)^k$$

Ces "propriétés algébriques" permettent de calculer avec des valeurs chiffrées sans connaître le contenu des chiffrés. Elles sont utiles quand l'anonymat des utilisateurs est nécessaire. Par exemple, nous pouvons calculer le total sans déchiffrer chaque vote individuellement et par conséquent, nous pouvons garantir la confidentialité des utilisateurs.

Description du schéma de Paillier

Rappelons brièvement le fonctionnement du système de chiffrement.

Génération des clés Soit N un module RSA $N = pq$, où p et q sont des nombres premiers. Soit g un entier d'ordre un multiple de N modulo N^2. La clé publique est $\mathsf{pk} = (N, g)$ et la clé secrète est $\mathsf{sk} = \lambda(N)$ où $\lambda(N)$ est défini comme $\lambda(N) = \mathrm{ppcm}\left((p-1),(q-1)\right)$.

Chiffrement Pour chiffrer un message $M \in \mathbb{Z}_N$, choisissons au hasard x dans $\mathbb{Z}_N^*$ et calculons le chiffré $c = g^M x^N \bmod N^2$.

Déchiffrement Pour déchiffrer c, calculons $M = \dfrac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$ où la fonction L prend ces entrées dans l'ensemble $\mathcal{S}_N = \{u < N^2 | u = 1 \bmod N\}$ et $L(u) = \frac{u-1}{N}$.

La version partagée du cryptosystème de Paillier

Afin d'éviter que les autorités apprennent les votes et pour protéger la confidentialité des électeurs, nous pouvons utiliser une version partagée du cryptosystème de Paillier (ch. chapitre 4). Par conséquent, au lieu de simplement déchiffrer le total chiffré, chaque autorité fait appel à n serveurs pour partager sa clé secrète de telle sorte qu'au moins t soient nécessaires pour déchiffrer un vote. Dans ce cas, on peut utiliser un des deux protocoles de partage du déchiffrement décrit aux chapitres 4 et 5. Contrairement au cas de la loterie, où l'on a vu que le chiffrement devait être IND-CCA, ici, le système de chiffrement a seulement besoin d'être IND-CPA. En effet, l'attaque du protocole de loterie consistant à attendre la fin du vote pour modifier le résultat ne fonctionne plus car l'électeur doit faire une preuve que le clair est dans un ensemble de valeurs données. La combinaison de la preuve zero-knowledge et d'un chiffrement homomorphique IND-CPA suffit à garantir la sécurité du protocole de vote. En effet, dans le cas du vote, un attaquant n'a pas accès à un oracle de déchiffrement car les valeurs (vote) qu'il peut soumettre à l'autorité sont dans un ensemble de petite taille.

7.3.3 Preuves Zero-Knowledge

Considérons des chiffrés de Paillier. Nous allons décrire dans cette sous-section deux différentes preuves zero-knowledge que le message secret vérifie certaines propriétés.

Par simplicité, nous ne présentons que la version interactive des protocoles. En utilisant l'heuristique de Fiat-Shamir [69], le vérifieur peut être remplacé par une fonction de hachage pour obtenir une forme non-interactive. Dans le modèle de l'oracle aléatoire, la sécurité est garantie, pourvu que le schéma interactif soit zero-knowledge face à un vérifieur honnête [150, 132]. Dans la suite, k est un paramètre de sécurité, et nous considérons que tous les paramètres sont fonction de k. Afin de simplifier les notations, nous n'écrivons pas les dépendances en k mais si une expression f est dite négligeable, ceci signifie que f dépend de k et que, pour tout polynôme P et pour des k suffisamment grands, $f(k) < 1/P(k)$.

La première preuve présentée n'est pas utilisée dans le protocole de vote mais permet de mieux comprendre le fonctionnement de la deuxième preuve qui elle, est utile au protocole de vote.

Preuve de connaissance d'un message chiffré

Soit N un module RSA de k bits. Étant donné $c = g^m r^N \bmod N^2$, le prouveur P va convaincre le vérifieur V qu'il connaît m, de manière similaire à Okamoto [133] et Guillou-Quisquater [102].

On note $a \div b$ le quotient dans la division de a par b.

1. P choisit au hasard $x \in \mathbb{Z}_N$ et $s \in \mathbb{Z}_N^*$. Il calcule $u = g^x s^N \bmod N^2$ et envoie u.
2. V choisit un challenge $e \in [0, A[$[64] et envoie e à P.
3. P calcule $v = x - em \bmod N$, $w = sr^{-e} g^{(x-em)\div N} \bmod N$ et les envoie à V.
4. V vérifie que $g^v c^e w^N = u \bmod N^2$.

Théorème 25. Pour des paramètres A et t tels que $1/A^t$ est négligeable, t itérations du protocole constituent une preuve zero-knowledge de connaissance de m (contre un vérifieur honnête).

64. A est petit dans le cas du protocole interactif pour que l'on puisse prouver que ce protocole est zero-knowledge. Dans tous les cas, A est inférieur à N.

Preuve: Consistance. Supposons que P connaisse m. Suivant le protocole, vous vérifions que $g^v c^e w^N = g^{x-em} g^{me} r^{Ne} s^N r^{-eN} g^{N((x-em)\div N)} = g^x s^N = u \bmod N^2$ car $g^{x-em} \times g^{N((x-em)\div N)} = g^{x-em} \bmod N^2$. Le terme $(x-em) \div N$ dans w correspond au quotient de la réduction modulo N dans v. Par conséquent, un prouveur honnête est toujours accepté.

Significative. Supposons qu'un tricheur P^* soit accepté avec probabilité non-négligeable. Par un argument simple, P^* réussit le protocole pour au moins deux valeurs différentes et leur séparation peut être calculée en temps linéairement borné par l'inverse de l'avantage de P^*. Il s'ensuit qu'en utilisant P^* comme un oracle, on peut calculer en temps polynomial (e_1, v_1, w_1) et (e_2, v_2, w_2) tels que $g^{v_1} c^{e_1} w_1^N = u = g^{v_2} c^{e_2} w_2^N \bmod N^2$ en utilisant un algorithme similaire à [167]. Considérant les logarithmes partiels (le logarithme discret modulo N), il en résulte que $v_1 + me_1 = v_2 + me_2 \bmod N$ et donc $m = (v_2 - v_1)/(e_1 - e_2) \bmod N$ ce qui justifie l'extraction de m à partir de P^*.

Zero-Knowledge. Comme nous supposons un vérifieur honnête, le challenge est indépendant de la mise en gage. Par conséquent, le simulateur choisit e au hasard dans $[0, A[$ et tire $v \in \mathbb{Z}_N$ et $w \in \mathbb{Z}_N^*$ jusqu'à ce que $\mathrm{pgcd}(w, N) = 1$. Alors, il calcule u pour satisfaire l'équation de vérification. La simulation s'exécute en temps linéaire par rapport au nombre t de tours. □

Cette deuxième preuve est une preuve que le chiffré est un chiffré de m_1, ou un chiffré de m_2, ou ..., ou un chiffré de m_p.

Preuve qu'un message chiffré est dans un ensemble donné de messages

Quand on chiffre un message, il est possible d'ajouter une preuve que le message est dans un ensemble public $\mathcal{S} = \{m_1, \ldots, m_p\}$ de p messages sans révéler plus d'informations.

Soit N un module RSA de k bits, $\mathcal{S} = \{m_1, \ldots, m_p\}$ un ensemble public de p messages, et $c = g^{m_i} r^N \bmod N^2$ le chiffrement de m_i où i est secret. Dans le protocole, le prouveur P convainc le vérifieur V que c chiffre un message dans $\mathcal{S}$.

1. P tire au hasard ρ dans $\mathbb{Z}_N^*$. Il tire aléatoirement $p-1$ valeurs $\{e_j\}_{j \neq i}$ dans $\mathbb{Z}_N$ et $p-1$ valeurs $\{v_j\}_{j \neq i}$ dans $\mathbb{Z}_N^*$. Alors, il calcule $u_i = \rho^N \bmod N^2$ et $\{u_j = v_j^N (g^{m_j}/c)^{e_j} \bmod N^2\}_{j \neq i}$. Finalement, il envoie $\{u_j\}_{j \in \{1,\ldots,p\}}$ à V.
2. V choisit un challenge aléatoire e dans $[0, A[$ et l'envoie à P.
3. P calcule $e_i = e - \sum_{j \neq i} e_j \bmod N$ et $v_i = \rho r^{e_i} g^{[m(e - \sum_{j \neq i} e_j)] \div N} \bmod N$ et envoie $\{v_j, e_j\}_{j \in \{1,\ldots,p\}}$ à V.
4. V vérifie que $e = \sum_j e_j \bmod N$ et que $v_j^N = u_j (c/g^{m_j})^{e_j} \bmod N^2$ pour chaque $j \in \{1, \ldots, p\}$.

Théorème 26. Pour tout paramètre $A \neq 0$ et t tels que $1/A^t$ soit négligeable, on vérifie que t itérations du protocole précédent constituent une preuve zero-knowledge parfaite (face à un vérifieur honnête) que le déchiffrement de c est un élément de $\mathcal{S}$.

Preuve: Consistance. Dans le protocole, P a mis en gage $u_1, \ldots, u_p$ et veut prouver en parallèle que chaque c/g^{m_j} est un résidu N-ième modulo N^2. A cette fin, il utilise la malléabilité du challenge qui lui permet de choisir à l'avance $p-1$ valeurs e_j et calcule les "mauvaises" mises en gage correspondantes $u_j = v_j^N (g^{m_j}/c)^{e_j} \bmod N^2$ où les v_j sont les réponses finales tirées au hasard dans $\mathbb{Z}_N^*$.

Significative. Supposons que le déchiffrement de c ne soit pas un membre de $\mathcal{S}$ et qu'un prouveur

tricheur P^* termine avec succès une itération du protocole. A partir de l'équation de vérification finale et de l'expression de c, il résulte que pour chaque $j \in \{1,\ldots,p\}$, $v_j^N = u_j(c/g^{m_j})^{e_j} \bmod N^2$. En prenant les logarithmes partiels, il s'ensuit que $0 = \log u_j + (m - m_j)e_j \bmod N$ et comme $m \neq m_j \bmod N$, $e_j = \log u_j/(m_j - m) \bmod N$ pour chaque $j \in \{1,\ldots,p\}$. Finalement, à partir de la dernière équation de vérification $e = \sum_j e_j \bmod N$, il s'ensuit que $e = \sum_j \log u_j/(m_j - m) \bmod N$ qui est vérifié avec probabilité au plus $1/A$. Si le protocole est itéré t fois, alors les arguments standards montrent que la probabilité que P^* passe le protocole ne peut pas dépasser de manière significative $1/A^t$ qui est une fonction négligeable de k.

Zero-Knowledge. Le simulateur choisit aléatoirement $\{e_j\}_{j\in\{1,\ldots,p\}}$ dans $\mathbb{Z}_N$. Notons que la somme $e = \sum_j e_j \bmod N$ simule de manière parfaite le challenge donné par un vérifieur honnête. Ensuite le simulateur tire au hasard $\{v_j\}_{j\in\{1,\ldots,p\}}$ dans $\mathbb{Z}_N^*$ et calcule $\{u_j = v_j^N/c^{e_j} \bmod N^2\}_{j\in\{1,\ldots,p\}}$. La suite $\{u_j, e, v_j\}_{j\in\{1,\ldots,p\}}$ est une simulation parfaite d'un tour du protocole. La simulation complète s'exécute en temps linéaire en le nombre t de tours. □

Dans [56], les auteurs ont amélioré ce résultat. Contrairement à notre preuve qui est linéaire par rapport au nombre de candidats, la preuve de Damgård et Jurik est logarithmique.

Preuve d'égalité de clairs

Quand on chiffre un message, il est possible d'ajouter une preuve que deux messages chiffrés sont égaux.

Soient $N_1,\ldots,N_p$, p modules RSA de k bits. Étant donnés les p chiffrements $c_j = g_j^m r_j^{N_j} \bmod N_j^2$, sous l'hypothèse que les déchiffrements de c_j soient dans l'intervalle $[0, 2^\ell[$, le prouveur P convainc le vérifieur V que les c_j chiffrent le même message m.

1. P tire au hasard $\rho \in [0, 2^k[$ et $s_j \in \mathbb{Z}_{N_j}^*$ pour chaque j dans $\{1,\ldots,p\}$. Ensuite, il calcule $u_j = g_j^\rho s_j^{N_j} \bmod N_j^2$ et met en gage u_j.
2. V choisit au hasard un challenge e dans $[0, A[$ et l'envoie à P.
3. P calcule $z = \rho + me$ et $v_j = s_j r_j^e \bmod N_j$ et envoie z et v_j pour chaque $j \in \{1,\ldots,p\}$.
   V vérifie que $z \in [0, 2^k[$ et que $g_j^z v_j^{N_j} = u_j c_j^e \bmod N_j^2$ pour chaque $j \in \{1,\ldots,p\}$.

Théorème 27. Pour n'importe quels paramètres A, t et ℓ tels que $1/A^t$ et $2^{\ell-k}$A sont négligeables, il s'ensuit que t itérations du protocole précédent constituent une preuve d'appartenance statistiquement zero-knowledge (face à un vérifieur honnête) que les éléments $\{c_1, ..., c_p\}$ chiffrent le même message de ℓ bits.

Preuve: Consistance. Pour tout $j \in \{1, ..., p\}$, il vient que $g_j^z v_j^{N_j} = g_j^{\rho+xe} s_j^{N_j} r_j^{eN_j} = u_j c_j^e \bmod N_j^2$, avec probabilité 1. De plus, comme $z = \rho + me$, l'inégalité $z < 2^k$ est vérifiée avec probabilité au moins $1 - 2^\ell A/2^k$. Ainsi, un prouveur honnête P termine avec succès t itérations du protocole avec probabilité $(1 - 2^{\ell-k}A)^t \approx 1 - 2^{\ell-k}At$. D'après les hypothèses, cette probabilité est écrasante.

Significative. Supposons qu'il existe i_1 et i_2 dans $\{1,\ldots,p\}$ tels que c_{i_1} chiffre m_1 et c_{i_2} chiffre m_2 avec $m_1 \neq m_2$. Alors, à partir des égalités vérifiés par V

$$\begin{cases} g_{i_1}^z v_{i_1}^{N_{i_1}} &= u_{i_1} g_{i_1}^{m_1 e} r_{i_1}^{eN_{i_1}} & \bmod N_{i_1}^2 \\ g_{i_2}^z v_{i_2}^{N_{i_2}} &= u_{i_2} g_{i_2}^{m_2 e} r_{i_2}^{eN_{i_2}} & \bmod N_{i_2}^2 \end{cases}$$

En prenant les logarithmes partiels, il s'ensuit que

$$\begin{cases} z & = & \log_{g_{i_1}} u_{i_1} & + & em_1 & \bmod N_{i_1} \\ z & = & \log_{g_{i_2}} u_{i_2} & + & em_2 & \bmod N_{i_2} \end{cases}$$

Comme $0 \leq z < 2^k$, $0 \leq em_i < A \times 2^\ell$ ($\ll 2^k$) nous obtenons $-2^k < -A \times 2^\ell < z - em_i < 2^k$, il existe ε_1 et ε_2 dans $\{0,1\}$ tels que les égalités suivantes sont vérifiées dans les entiers :

$$\begin{cases} z & = & \log_{g_{i_1}} u_{i_1} & + & em_1 & - & \varepsilon_1 N_1 \\ z & = & \log_{g_{i_2}} u_{i_2} & + & em_2 & - & \varepsilon_2 N_2 \end{cases}$$

Par conséquent, si $m_1 \neq m_2$, $e = (\log u_{i_1} - \log u_{i_2} - \varepsilon_1 N_{i_1} + \varepsilon_2 N_{i_2})/(m_2 - m_1)$, avec probabilité au plus $4/A$. Après t tours, cette probabilité décroit à $(4/A)^t + \varepsilon$, où ε est négligeable.

Zero-Knowledge. Comme dans la preuve précédente, la même simulation fonctionne. Cependant, comme le simulateur tire uniformément z dans $[0, 2^k[$ et non dans $[me, 2^k + me[$, seule une indistinguabilité statistique peut être atteinte. □

7.3.4 Schéma de vote

Dans cette section, on décrit le protocole complet de notre schéma. Nous insistons sur la génération du vote d'un électeur et les communications entre les trois niveaux hiérarchiques; les niveaux : national, régional et local.

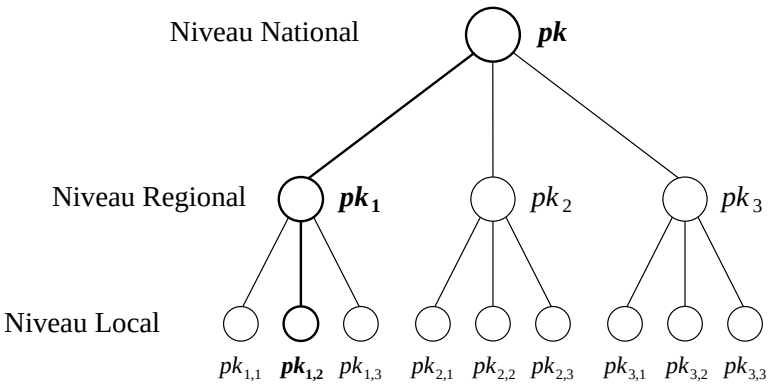


FIG. 7.1: *Organisation des autorités.*

Initialisation

Nous considérons une élection "1 parmi p" où un candidat est choisi parmi p. Dans le processus d'initialisation, chaque autorité, à n'importe quel niveau, génère sa clé publique et la certifie avec une autorité de certification indépendante. Nous utilisons les notations hiérarchiques suivantes : pk pour l'autorité nationale, pk_i pour les autorités régionales et $pk_{i,j}$ pour les autorités locales.

Après la phase préliminaire, chaque autorité publie sur son propre tableau de bord les clés publiques adéquates : par exemple, la i-ième autorité régionale, les clés pk, pk_i, et la j-ième autorité locale de la i-ième autorité régionale les clés $pk, pk_i, pk_{i,j}$.

On dénote par ℓ le nombre de votants et nous fixons M un entier supérieur à ℓ. Par exemple, nous pouvons choisir $M = 2^{\lceil \log_2 \ell \rceil}$, la puissance de 2 immédiatement supérieure à ℓ.

La phase de vote

Considérons un votant d'une autorié locale avec les clés publiques $pk, pk_i, pk_{i,j}$ qui souhaite voter pour le m-ième candidat. Il prépare un bulletin de la manière suivante :

1. Il télécharge à partir du tableau de bord les 3 clés publiques $pk, pk_i, pk_{i,j}$ de sa zone.

2. En utilisant chaque clé publique, il chiffre l'entier M^m avec le schéma de Paillier et génère les chiffrés C_n, C_r et C_ℓ respectivement avec la clé de l'autorité nationale, régionale et locale.

3. Il génère des preuves pour convaincre n'importe quel vérifieur que chaque chiffré représente un vote valide, i.e. un entier M^m avec $m \in \{1, \ldots, p\}$. Ceci est fait en utilisant la preuve que le message est dans un ensemble donné.

4. Il génère une preuve pour convaincre n'importe quel vérifieur que les trois chiffrés représentent le même vote. Ceci est fait en utilisant les preuves d'égalité des clairs. On peut remarquer qu'une telle preuve est correcte parce que les preuves précédentes garantissent que la taille du message chiffré est bornée.

Calcul du résultat du vote

Dans cette sous-section, nous présentons le calcul du total du vote. Nous décrivons d'abord le tableau public des autorités locales auquel tous les électeurs peuvent accéder, par exemple, à travers le site web de l'autorité locale $A_{i,j}$.

Nom	C_ℓ	C_r	C_n	Preuves	Signature	Signature du TTS	Droits
Electeur 1	$g_{i,j}^{v_{i,j,1}}$	$g_i^{v_{i,j,1}}$	$g^{v_{i,j,1}}$				R
Electeur 2	$g_{i,j}^{v_{i,j,2}}$	$g_i^{v_{i,j,2}}$	$g^{v_{i,j,2}}$				R
							R
Electeur k	$g_{i,j}^{v_{i,j,k}}$	$g_i^{v_{i,j,k}}$	$g^{v_{i,j,k}}$				R/W
							R
Electeur ℓ	$g_{i,j}^{v_{i,j,\ell}}$	$g_i^{v_{i,j,\ell}}$	$g^{v_{i,j,\ell}}$				R
Somme de $A_{i,j}$	$\sum_k v_{i,j,k}$	$\prod_k g_i^{v_{i,j,k}}$	$\prod_k g^{v_{i,j,k}}$				R

FIG. 7.2: *Tableau des bulletins de vote d'une autorité locale.*

La figure 7.2 représente un tableau de bord où le k-ième électeur peut lire et écrire dans sa propre ligne mais ne peut que lire dans les autres colonnes.

Les votants écrivent leur nom avec leur certificat, les trois votes et les preuves. Ensuite, ils signent les données de leur ligne et le vote est signé par le serveur de temps de confiance TTS.

Quand le système de vote est fermé, les autorités locales doivent vérifier toutes les preuves et les signatures des utilisateurs et du TTS. Puis, les autorités locales $A_{i,j}$ calculent le produit des votes corrects dans les colonnes 2 à 4. Ensuite, elles agissent comme le combineur dans le cryptosystème à seuil de Paillier pour le premier produit qui correspond à leur clé publique. Le déchiffrement peut être surveillé par des personnes externes car le processus de déchiffrement est publiquement vérifiable.

Autorités locales	$V_{j,k}$	$\prod_{j,k} C_r$	$\prod_{j,k} C_n$	Signature d'autorités	Droits
Aut. locale 1	$\sum_k v_{i,1,k}$	$\prod_k g_i^{v_{i,1,k}}$	$\prod_k g^{v_{i,1,k}}$		R
Aut. locale 2	$\sum_k v_{i,2,k}$	$\prod_k g_i^{v_{i,2,k}}$	$\prod_k g^{v_{i,2,k}}$		R
					R
Aut. locale j	$\sum_k v_{i,j,k}$	$\prod_k g_i^{v_{i,j,k}}$	$\prod_k g^{v_{i,j,k}}$		R/W
					R
Aut. locale ℓ	$\sum_k v_{i,\ell,k}$	$\prod_k g_i^{v_{i,\ell,k}}$	$\prod_k g^{v_{i,\ell,k}}$		R
Somme de A_i	$\sum_{j,k} v_{i,j,k}$	$\prod_{j,k} g_i^{v_{i,j,k}}$ $\Downarrow$ déchiffre $\sum_{j,k} v_{i,j,k}$	$\prod_{j,k} g^{v_{i,j,k}}$		R

FIG. 7.3: *Tableau des résultats de l'autorité régionale.*

Autorités régionales	$V_{i,j,k}$	$\prod_{i,j,k} C_n$	Signature Aut.	Droits
Aut. régionale 1	$\sum_{j,k} v_{1,j,k}$	$\prod_{j,k} g^{v_{1,j,k}}$		R
Aut. régionale 2	$\sum_{j,k} v_{2,j,k}$	$\prod_{j,k} g^{v_{2,j,k}}$		R
				R
Aut. régionale i	$\sum_{j,k} v_{i,j,k}$	$\prod_{j,k} g^{v_{i,j,k}}$		R/W
				R
Aut. régionale ℓ	$\sum_{j,k} v_{\ell,j,k}$	$\prod_{j,k} g^{v_{\ell,j,k}}$		R
Somme de A	$\sum_{i,j,k} v_{i,j,k}$	$\prod_{i,j,k} g^{v_{i,j,k}}$ $\Downarrow$ déchiffre $\sum_{i,j,k} v_{i,j,k}$		R

FIG. 7.4: *Tableau des résultats de l'autorité nationale.*

Les totaux locaux sont publiés dans le tableau de bord de l'autorité régionale A_i, dans la colonne 1 par les autorités locales. Dans les colonnes 2 et 3, l'autorité régionale écrit le produit des votes des électeurs de chaque autorité locale dans son tableau de résultat. Les différentes autorités $A_{i,j}$ qui dépendent de l'autorité A_i ont les mêmes droits que les utilisateurs sur le tableau de bord précédent. La colonne C_ℓ est remplacée par le total déchiffré. Les autorités locales A_i calculent la somme dans la première colonne, les produits des colonnes 2 et 3, et déchiffrent le produit final dans la colonne 2 avec le processus à seuil comme les autorités locales. A la fin, ils vérifient que les totaux calculés par les deux méthodes. Ils écrivent sur le tableau de bord de l'autorité nationale leur somme dans la colonne 1. Dans la colonne 2, ils écrivent le produit des autorités locales.

L'autorité nationale peut calculer le produit des éléments dans la colonne 2 de son tableau de bord. Ensuite, il déchiffre ce produit et vérifie si ce résultat est égal à la somme des éléments dans la colonne 1.

Fraudes dans le calcul

Les sous-divisions hiérarchiques fournissent un moyen efficace de vérification publique distribuée des résultats. Chaque votant est capable de vérifier si son vote a été pris en compte dans le total local.

Ensuite, il exécute récursivement des vérifications similaires qui le convainquent que le niveau supérieur a correctement pris en compte le total local. Finalement, il a la garantie que ce vote fait partie du résultat national.

De plus, si une erreur est détectée par une autorité donnée, le sous-résultat faux sera éliminé du total et sera ultérieurement recalculé.

Complexité pratique du schéma

Considérons la complexité pratique de la phase de vote décrite dans cette section. Un vote est composé de trois chiffrés C_n, C_r et C_ℓ et de preuves de validité. Certaines optimisations peuvent être appliquées. Par exemple, les mises en gage peuvent être remplacées par leur valeur de hachage comme décrit dans [93].

Notons $|H|$ la taille des mises en gage hachées, $|A|$ la taille des challenges et $|N|$ la taille du module utilisé dans le cryptosystème de Paillier. La taille d'un vote est exactement

$$4|H| + 4|A| + (11 + 9p)|N|$$

où p est le nombre de candidats. Par conséquent, la complexité en communication du schéma est linéaire en la taille du module de Paillier, en le nombre de candidats et en le nombre de votants. La complexité de calcul est donc de $\Theta(p \times |N|)$ multiplications modulaires à la fois pour la génération et la vérification du vote.

Dans les applications pratiques, nous pouvons choisir $|H| = 80$, $|A| = 80$ et $|N| = 1024$. Pour un schéma d'élection à 10 candidats, la taille d'un vote est d'environ 12.5 Ko.

Enfin, le découpage entre autorités locales et régionales permet de réduire le coût de calcul d'une seule autoritée. En effet, les vérifications des preuves et des signatures sont réduites car si on a d autorités locales et ℓ électeurs, chaque autorité aura en moyenne ℓ/d électeurs à vérifier.

7.3.5 Améliorations du schéma

Dans cette section, nous décrivons des solutions pour les exigences de sécurité suivante : anonymat des votants, et empêcher la création d'un reçu pour éviter l'achat de vote.

Anonymat

Dans les systèmes de vote actuels, la liste des électeurs actifs est connue, au moins par l'autorité. Mais elle est utile pour contrôler que personne ne vote deux fois.

Cependant, il n'y a pas de raisons pratiques pour avoir une telle liste de participants actifs et dans la plupart des situations seul le nombre total d'entre eux est nécessaire. De plus, à cause du format numérique, une analyse à grande échelle peut être nécessaire, sur beaucoup de processus de vote.

En conséquence, il peut être essentiel de protéger l'anonymat des (non)-votants, tout en étant capable d'éviter le vote double. Comme d'habitude, les signatures en blanc [41] sont un outil adéquat pour founir un tel anonymat, tout en préservant le vote double d'un même certificat.

Considérons un schéma de signature en blanc qui évite une falsification supplémentaire [149]. On peut soit utiliser la version Okamoto-Schnorr [133, 167] basée sur la difficulté de calculer les logarithmes discrets, ou la version de Okamoto-Guillou-Quisquater [133, 102] basée sur la difficulté de calculer les racines e-ième modulo un nombre composite. On peut remarquer que la dernière solution n'ajoute pas d'hypothèse calculatoire supplémentaire car le problème RSA est plus fort que le problème des hauts résidus.

Après avoir prouvé son identité, chaque électeur crée une nouvelle paire de clés publique et secrète. Puis, avec l'aide d'une (des) autorité(s), il obtient un certificat pour la clé publique en utilisant un processus de signature en blanc. Bien sûr, l'autorité doit accepter d'interagir au moins une fois avec l'électeur. A la fin du processus de certification, chaque électeur possède un certificat de clé publique qui peut être utilisé comme un pseudonyme.

Comme nous voulons aussi éviter les coalitions entre un électeur et les autorités, nous devons utiliser une signature en blanc distribuée qui exige que toutes les autorités, avec l'aide de l'électeur, obtiennent un pseudonyme valide. Par conséquent, il sera impossible à certaines autorités d'aider l'électeur à obtenir plusieurs pseudonymes pour signer plus d'une fois.

Décrivons un exemple de Certification Distribuée de Pseudonymes. Soit $Z \in \mathbb{Z}_N^*$ la clé publique de signature de l'autorité avec un exposant public suffisamment grand e, et les paires (x_i, y_i) les clés distribuées secrètes, avec $x_i \in \mathbb{Z}_N^*$ et $y_i \in [0, e[$ telles que $Z = \prod {x_i}^e a^{y_i} \bmod N$. Pour obtenir un certificat du pseudonyme V, l'électeur interagit avec les autorités après avoir prouvé son identité :

1. chaque autorité choisit au hasard $u_i \in \mathbb{Z}_N^*$ et $v_i \in [0, e[$, et calcule la mise en gage $w_i = u_i^e a^{v_i} \bmod N$, qu'il envoie à l'électeur.

2. Ce dernier choisit des facteurs aléatoires de masquage $\beta \in \mathbb{Z}_N^*$ et $\alpha, \gamma \in [0, e[$, et calcule $w = (\prod w_i) a^\alpha \beta^e Z^\gamma \bmod N$

3. Puis, il calcule le challenge $\varepsilon = H(w, V)$, $c = \varepsilon + \gamma \bmod e$ et le broadcaste aux autorités.

4. Chaque autorité calcule $r_i = v_i + cy_i \bmod e$, $s_i = (v_i + cy_i) \div e$ ainsi que $t_i = u_i x_i^c a^{s_i} \bmod N$. Elles envoient (r_i, t_i) à l'électeur.

5. Il calcule $r = \sum r_i + \alpha \bmod e$, $s = (\sum r_i + \alpha) \div e$, $s' = (c - \varepsilon) \div e$ et $t = \prod t_i \beta a^s Z^{-s'} \bmod N$

Par construction,

$$\begin{aligned} a^{r_i} t_i^e &= a^{r_i + es_i} u_i^e x_i^{ce} = a^{v_i + cy_i} u_i^e x_i^{ce} = a^{v_i} u_i^e (a^{y_i} x_i^e)^c = w_i Z^c \bmod N. \\ a^r t^e &= a^{r+es} Z^{-es'} (\prod t_i^e) \beta^e = a^{\sum r_i + \alpha} Z^{-es'} (\prod t_i^e) \beta^e = (\prod a^{r_i} t_i^e) a^\alpha \beta^e Z^{-es'} \\ &= Z^{c-es'} (\prod w_i) a^\alpha \beta^e = Z^{c-es'-\gamma} w = Z^{c-c+\varepsilon} w = Z^\varepsilon w \bmod N. \end{aligned}$$

avec $\varepsilon = H(w, V)$, où V est le pseudonyme de l'électeur. Grâce à l'indistinguabilité des témoins d'Okamoto-Guillou-Quisquater, la signature en blanc évite une falsification supplémentaire : aucun électeur ne peut obtenir deux pseudonymes pour voter deux fois, à moins que certaines autorités ne coopèrent. Cependant, grâce au partage de la clé secrète, toutes les autorités doivent coopérer pour produire un certificat pour un électeur qui tente de frauder.

On peut améliorer l'efficacité du schéma précédent, où l'électeur prouve son identité et demande une signature en blanc aux autorités. Au lieu de commencer par prouver son identité, l'électeur peut simplement signer le challenge c qu'il envoie aux autorités. De plus, au lieu de mettre en jeu toutes les autorités pour produire le pseudonyme, on peut utiliser un schéma de signature en blanc partagé [174]. Par conséquent, $t + 1$ autorités seraient suffisantes.

Grâce à la propriété de zero-knowledge, avec un paramètre de taille fixe e, elle est non-transférable. Mais pour réduire le nombre d'interactions, il est possible d'utiliser une preuve *non-interactive* à vérifieur désigné [109].

Les propriétés d'être "sans-reçu" et d'incoercition

Définissons les propriétés d'être *sans-reçu* et d'être d'*incoercible*.

Définitions

Sans-reçu La propriété d'être "sans-reçu" assure que le votant ne peut pas fournir un reçu prouvant le contenu de son vote, même s'il le veut et donc résout le problème de l'achat de vote.

Incoercible La propriété d'être "incoercible" assure que le votant ne peut pas être forcé à voter pour un candidat donné.

Par exemple, dans le système de vote décrit précédemment, on voit que le chiffrement de Paillier permet de générer un reçu très facilement en fournissant le random utilisé pour chiffrer le vote. En effet, le couple (M, u) tel que $c = g^M u^N \mod N^2$ fournit un reçu du vote c car tout le monde peut vérifier si

$$c \stackrel{?}{=} g^M u^N \mod N^2$$

De plus, ce système n'est pas incoercible car un attaquant peut imposer à un électeur de voter pour un candidat donné en lui imposant un random u.

Le problème de la coercition.
La notion de coercition demande des explications complémentaires. Par exemple, il est clair que si l'attaquant est derrière le votant, il peut espionner tout ce que fait l'électeur et le forcer à voter pour un candidat en le menaçant. Pour qu'un système soit incoercible, il faut donc supposer qu'à un moment donné au moins, l'électeur n'est pas en compagnie de la personne qui le menace. En particulier, au moment du vote, on peut supposer que l'attaquant n'espionne pas l'électeur, sinon, il peut voter à sa place. Pour garantir cette propriété, la seule solution semble être d'utiliser un isoloir[65]. Définissons donc une notion plus faible de la coercibilité, où l'attaquant rencontre l'électeur, mais n'est pas présent au moment du vote. Par contre, il peut espionner le canal de communication entre l'électeur et l'autorité de vote. On appelle cette notion, la coercition faible. On peut alors distinguer deux types de coercition : l'attaquant contacte le votant *avant* l'élection, soit *après* l'élection.

L'attaquant "avant" a un pouvoir très grand. Il peut dire au votant comment voter et aussi lui imposer le random à utiliser pendant le vote. Ceci correspond à fixer le comportement du votant durant le protocole. Si le votant n'obéit pas, il sera facile à l'attaquant de le détecter. Les solutions à ce problème utilisent des hypothèses physiques.

L'attaquant "après" a un pouvoir moins important. Il peut seulement aller voir le votant et demander à voir le vote v et le random ρ utilisé par l'électeur durant le protocole. Il y a peut-être un moyen pour le votant de construire un vote v' et ρ' différents qui correspondent à l'exécution du protocole. Ceci n'est pas possible dans le protocole ci-dessus. Canetti et Gennaro ont proposé un protocole pour résoudre ce problème dans [34]. Ils ont utilisé un système de chiffrement *déniable* inventé par Canetti, Dwork, Naor et Ostrovsky dans [33]. Il est donc plus simple de considérer la coercition *après* que la coercition avant. C'est ce que nous ferons dans la suite.

Comparaison entre la coercition et sans-reçu.
Dans un premier temps, on pourrait penser que la coercibilité faible est une notion plus faible que d'être sans-reçu. En effet, cette dernière signifie que même si l'électeur veut tricher, il n'arrivera pas à créer

65. Il serait aussi utile de pouvoir signer de chez soi sans avoir besoin d'aller voter dans un isoloir situé dans un bureau de vote. Pour ce faire, on peut supposer que l'électeur et l'autorité de vote aient échangé au préalable une permutation sur l'ensemble $\{1, 2, \ldots, p\}$. Ainsi au moment du vote, seul le votant est en mesure de savoir pour qui il vote en choisissant 1, 2 ou p. Il peut révéler n'importe quelle permutation à l'attaquant et ce dernier ne saura pas pour qui il vote.

un reçu, alors que dans le cas de la coercibilité faible, le reçu est construit par une personne qui a moins d'information car il n'est pas présent au moment du vote.

Cependant, dans la coercition, l'attaquant peut espionner le canal de communication entre l'électeur et l'autorité de vote. Quand un utilisateur construit un reçu, la vérification est *off-line*, ce qui veut dire que le vérifieur n'a pas vu les échanges entre l'autorité et le votant. Il n'a accès qu'au bulletin de vote dans le tableau public de l'autorité de vote et le votant doit fabriquer un reçu indiquant pour qui il a voté. Ainsi, le vérifieur a moins d'information qu'un attaquant coercitif qui veut vérifier *a posteriori* ce qu'a voté un électeur. Dans le cas de la coercition faible, on suppose que l'attaquant n'est pas présent au moment du vote et par conséquent, il ne connaît pas le random utilisé par le votant. En revanche, à la fin du vote, il peut forcer le votant à indiquer son random utilisé. Par conséquent, la coercition faible n'est pas une notion plus faible que d'être sans-reçu. On verra dans la suite deux protocoles sans-reçu qui ne sont pas faiblement incoercibles.

Enfin, on peut remarquer que si on impose à un attaquant coercitif faible de ne pas avoir accès au canal de communication entre le votant et l'autorité, mais seulement au tableau public des bulletins de vote, alors la propriété d'être sans-reçu est plus forte que la coercition faible car même si l'électeur veut construire un tel reçu, il ne le pourra pas.

Historique des systèmes sans-reçu et incoercibles.
La propriété d'être sans-reçu a été d'abord utilisée par Benaloh et Tuinstra [17]. Leur solution utilise un isoloir qui garantit physiquement une communication secrète[66] dans les deux sens entre les autorités de vote et les votants. En effet, si l'autorité peut envoyer de manière secrète le random utilisé dans le chiffrement de Paillier et si personne ne peut apprendre le vote c envoyé à l'autorité, personne ne pourra construire un reçu, ni contraindre l'électeur à voter pour un candidat donné. Cette hypothèse de canal secret bi-directionnel permet de construire un système de vote sans-reçu et incoercible.

Un autre protocole de vote sans reçu basé sur un réseau de mélangeurs a été proposé par Sako et Kilian [114] où un canal de communication secret des autorités vers les votants est uniquement supposé. Ce canal n'est pas bi-directionnel.

Une autre solution serait d'utiliser un système de *chiffrement déniable* [33, 34]. Dans ce type de chiffrement, un utilisateur peut présenter n'importe quel clair et prouver qu'il correspond à un chiffré donné, alors que seul le déchiffrement fournit le clair correct. Avec un tel système de chiffrement, les votants peuvent "mentir" sur le vote réellement effectué et cette technique assure l'incoercition faible des votants mais pas d'être sans-reçu. Ce type de système de chiffrement est aussi appelé, *chiffrement sans mise en gage* [58].

Sous l'hypothèse de l'effacement, où les valeurs aléatoires sont effacées et oubliées après leur utilisation, l'incoercition est simplement fournie : le votant ne peut pas révéler son vote chiffré avec un schéma de chiffrement sémantiquement sûr, s'il ne stocke pas la valeur aléatoire et s'il l'oublie. Cependant, si l'attaquant force l'électeur à se souvenir du random, le système ne résistera pas à la coercition.

Hypothèses Physiques.
Pour fournir la propriété d'être sans-reçu, nous pouvons nous appuyer sur un matériel résistant, comme une carte à puce. Ainsi, les randoms utilisés durant la phase de vote ne seront pas accessibles. Ce système est aussi incoercible *avant* et *après*. Par conséquent, grâce à la sécurité sémantique du cryptosystème de Paillier, nous sommes sûrs que personne n'apprendra d'information sur le vote, même avec l'aide du votant. Cependant, cette hypothèse peut apparaître trop forte dans certains cas. On pourra alors faire

66. Il est important de remarquer ici qu'il existe une différence entre un *canal secret* entre deux utilisateurs et un *canal privé*. Un canal secret est un canal qui ne peut pas être écouté, alors qu'un canal privé est un canal qui peut être écouté, mais qui est chiffré, en utilisant par exemple un chiffrement symétrique. Si on utilise un chiffrement déniable, on peut alors simuler un canal secret par un canal privé, mais dans le cas d'un chiffrement symétrique par exemple, il n'y a pas équivalence. En effet, dans le cas d'un attaquant coercitif, il peut demander à l'électeur de révéler la clé symétrique et déchiffrer le canal privé. La contrainte d'un canal secret est une hypothèse plus forte qu'une hypothèse d'un canal privé.

une hypothèse plus faible comme un canal de communication secret entre l'électeur et une autorité qui randomise les chiffrés que l'on appelera randomiseur.

Cependant, pour des applications pratiques, l'utilisation d'une carte à puce ayant cette propriété d'effacement permet de concevoir des systèmes résistants à un attaquant coercitif et sans-reçu.

Exemples de systèmes sans-reçu.

Réseaux de mélangeurs et randomiseurs.
Ce canal de communication secret a aussi été utilisé dans un article de Hirt et Sako [106]. Mais pour être sans-reçu, le réseau de mélangeurs utilisé a une complexité en communication très élevée. Le but de ce protocole est de randomiser les chiffrés par l'autorité de vote de telle sorte que l'électeur ne connaisse pas le random utilisé dans Paillier par exemple. Ainsi, il ne pourra pas fournir un reçu de son vote. L'électeur va interagir avec plusieurs autorités qui vont randomiser successivement un vecteur de chiffrés et permuter les chiffrés dans le vecteur.

En effet, ils utilisent un schéma de chiffrement $\mathcal{E}$ homomorphique et autorisent le rechiffrement aléatoire, noté "$\stackrel{R}{\leftarrow}$".

– chaque vote possible est tout d'abord chiffré par un "aléa fixe" (disons 0)

$$\mathcal{L}_0 = \{\mathcal{E}_{0,1} = \mathcal{E}(1;0), \mathcal{E}_{0,2} = \mathcal{E}(2;0), \ldots, \mathcal{E}_{0,p} = \mathcal{E}(p;0)\}$$

– chaque autorité permute et rechiffre tous les votes, avec une permutation aléatoire π_i, et de nouvelles valeurs de randoms. Le prouveur ne peut donc pas prouver le contenu d'un chiffré car il ne connaît pas le random utilisé.

$$\mathcal{L}_i = \{\mathcal{E}_{i,1} \stackrel{R}{\leftarrow} \mathcal{E}_{i-1,\pi_i(1)}, \mathcal{E}_{i,2} \stackrel{R}{\leftarrow} \mathcal{E}_{i-1,\pi_i(2)}, \ldots, \mathcal{E}_{i,p} \stackrel{R}{\leftarrow} \mathcal{E}_{i-1,\pi_i(p)}\}.$$

– chaque autorité communique en privé la permutation π_i à l'électeur, pour qu'il puisse continuer le vote en choisissant le chiffré correspondant à son vote.

– chaque autorité prouve à chaque électeur (de manière privée) que cette permutation a réellement été utilisée. Sans révéler le nouvel aléa, elle prouve de manière zero-knowledge, que pour chaque j, $\mathcal{E}_{i,j}$ et $\mathcal{E}_{i-1,\pi_i(j)}$ chiffrent le même message.

– chaque autorité prouve publiquement de manière zero-knowledge, qu'il existe une telle permutation: pour chaque j, $\mathcal{E}_{i-1,j}$ a été rechiffré dans $\mathcal{L}_i$.

– finalement, l'électeur qui est le seul capable de suivre son vote, le pointe dans la liste finale.

Ainsi, le random utilisé dans le chiffré n'est pas connu par l'électeur et ce dernier ne peut donc pas construire un reçu pour son vote. Ce protocole n'est pas faiblement incoercible car rien n'empêche l'attaquant d'avoir accès au canal et de suivre les preuves zero-knowledge faites par l'autorité et d'être persuadé comme l'électeur.

Le désavantage principal est que, dans le tableau de bord public, les preuves publiques doivent apparaître. Le réseau de mélangeurs utilisé rend cette solution impraticable : complexité de la preuve de permutations, stockage des preuves et des listes.

Randomiseurs indépendants.
Dans cette sous-section, nous décrivons notre solution pour résister à l'achat de vote. Notre proposition utilise des "self-scrambling anonymizers" décrits dans l'article [148]. L'idée est la même que dans le

schéma précédent, où l'électeur ne doit pas connaître le random utilisé dans le chiffrement, mais éviter l'utilisation coûteuse du réseau de mélangeurs. Les électeurs demandent à une entité externe de randomiser leur vote sans modifier le contenu. Mais, le votant exige plus qu'un nouveau chiffrement de son vote :

- il veut une preuve zero-knowledge que son nouveau chiffré chiffre le même vote que son vote d'origine. Cependant, cette preuve ne doit pas être transférable. Elle pourrait lui fournir un reçu car il pourrait prouver le contenu de son chiffrement d'origine. De plus, cette preuve doit être zero-knowledge afin d'éviter de perdre de l'information sur la nouvelle valeur randomisée et utilisée dans le vote chiffré. Par conséquent, soit ils utilisent une preuve interactive zero-knowledge, qui est non transférable comme n'importe quelle preuve zero-knowledge, grâce à la simulation du transcript, soit ils utilisent une preuve non-interactive à vérifieur désigné [109].
- L'électeur a aussi besoin d'une preuve zero-knowledge que le nouveau vote chiffré est valide. Cependant, comme il ne connaît pas la nouvelle valeur randomisée introduite par le randomiseur, il ne peut plus fournir une telle preuve. Par conséquent, il doit interagir avec le randomiseur pour l'obtenir. Mais on souhaite que n'importe quelle partie du transcript ne puisse pas être utilisée par un électeur comme un reçu. Ainsi, nous utiliserons la propriété *divertible*[67] [131, 29, 108, 45, 46] des preuves interactives précédentes qui prouvent que le message chiffré est dans un ensemble connu. Avec une telle preuve divertible, le transcript vu par un électeur est indépendant de la preuve résultante : la preuve résultante (signature) ne contient pas de canal subliminal qui permettrait à un électeur de cacher/choisir quelque chose pour faire un reçu.
- Comme nous utilisons trois niveaux (et peut-être plus) dans notre mécanisme, le randomiseur doit randomiser les trois votes, et l'électeur a besoin d'une preuve que les trois votes résultants chiffrent le même message. Une version divertible de la preuve d'égalité des clairs pourra aussi être utilisée.

Preuves zero-knowledge pour la version du système sans-reçu.
Décrivons rapidement la première étape, durant laquelle le randomiseur prouve l'équivalence entre deux valeurs chiffrées. L'électeur a utilisé l'aléa r pour chiffrer son vote dans $c = g^{m_i} r^N \bmod N^2$, qu'il envoie au randomiseur. Ce dernier introduit alors un nouvel aléa s pour transformer le vote c en $c' = cs^N = g^{m_i}(rs)^N \bmod N^2$.

1. Le randomiseur choisit au hasard $x \in \mathbb{Z}_N$ et calcule $u = x^N \bmod N^2$ qu'il envoie à l'électeur.
2. Ce dernier choisit un challenge $e \in [0, A[$ qu'il envoie au randomiseur.
3. Ce dernier calcule $v = xs^{-e} \bmod N^2$ et l'envoie à l'électeur.
4. Enfin, ce dernier peut vérifier si $v^N (c'/c)^e = u \bmod N^2$.

La seconde étape consiste simplement en une variante divertible de la preuve précédente qu'un message chiffré est dans un intervalle donné. Nous utilisons par conséquent les mêmes notations où le randomiseur peut être vu comme le vérifieur du point de vue de l'électeur, mais il retourne finalement une preuve non-interactive que personne ne peut lier avec l'original.

1. Le randomiseur reçoit la liste de $\{u_j\}_{j \in \{1,\ldots,p\}}$ envoyée par l'électeur. Alors il choisit les facteurs de masquage:
   - un aléa $\beta \in \mathbb{Z}_N$

67. Ce type de preuve permet de créer une nouvelle preuve à partir d'une ancienne preuve.

– ainsi que p valeurs $\{\beta_j\}_{j\in\{1,\ldots,p\}}$ dans $\mathbb{Z}_N$ telles que $\sum \beta_j = \beta \bmod N$
– et p valeurs $\{\alpha_j\}_{j\in\{1,\ldots,p\}}$ dans $\mathbb{Z}_N^*$.

Ensuite, il calcule

$$u'_j = u_j \alpha_j^N (c/g^{m_j})^{\beta_j} \bmod N^2.$$

Il obtient finalement $e' = H(u'_1, \ldots, u'_p)$ et envoie $e = e' + \beta \bmod N$ à l'électeur.

2. l'électeur envoie une liste $\{v_j, e_j\}_{j\in\{1,\ldots,p\}}$ au randomiseur qui satisfait

$$e = \sum_j e_j \bmod N \text{ et } v_j^N = u_j (c/g^{m_j})^{e_j} \bmod N^2 \text{ pour chaque } j \in \{1, \ldots, p\}$$

3. Le randomiseur calcule $e'_j = e_j - \beta_j \bmod N$ et $v'_j = v_j \alpha_j s^{e'_j} \bmod N$. Finalement, il envoie la liste $\{v'_j, e'_j\}_{j\in\{1,\ldots,p\}}$ à l'électeur.

On peut vérifier que

$$\begin{aligned} e' &= e - \beta = \sum_j e_j - \sum_j \beta_j = \sum_j (e_j - \beta_j) = \sum_j e'_j \bmod N, \\ {v'_j}^N &= v_j^N \alpha_j^N s^{N e'_j} = u_j (c/g^{m_j})^{e_j} \alpha_j^N s^{N e'_j} = u_j (cs^N/g^{m_j})^{e'_j} (c/g^{m_j})^{\beta_j} \alpha_j^N \\ &= u'_j (cs^N/g^{m_j})^{e'_j} = u'_j (c'/g^{m_j})^{e'_j} \bmod N^2 \text{ pour chaque } j \in \{1, \ldots, p\}, \end{aligned}$$

où $e' = H(u'_1, \ldots, u'_p)$. Par conséquent, la liste $\{u'_j, v'_j, e'_j\}_{j\in\{1,\ldots,p\}}$ est une preuve non-interactive de validité du vote chiffré c'.

Enfin, ce schéma n'est pas incoercible même faiblement car l'attaquant peut suivre les preuves zero-knowledge et être convaincu comme l'est le votant. Il faudrait utiliser un canal *secret* entre le votant et l'autorité, ce que nous souhaitons éviter.

7.3.6 Implémentation

L'implémentation faite à l'École Normale Supérieure est conforme au protocole décrit dans la section 7.3.4, mais n'utilise qu'une seule autorité de vote. En effet, dans cette maquette un unique centre de déchiffrement est utilisé par les différentes autorités de vote (locales, régionales, et nationale). En conséquence, il n'est plus nécessaire de chiffrer le même vote sous trois clés différentes et de prouver que ces chiffrés correspondent à la même valeur. Ceci améliore la complexité du protocole.

Ainsi, chaque électeur vote en choisissant son candidat et prouve qu'il a voté pour un candidat dans un ensemble prédéfini. Ce vote est ensuite signé et posté par l'électeur dans le tableau public des bulletins. A la clôture du scrutin, les autorités locales, régionales et nationale interrogent le serveur de déchiffrement pour obtenir les résultats locaux, régionaux, et national.

Enfin, le système garantit l'anonymat des votants, les fraudes de l'autorité, les fraudes des électeurs. En revanche, l'anonymat des électeurs, c'est-à-dire connaître si une personne a voté ou non et les problèmes d'achat de vote ne sont pas assurés.

7.4 Conclusion

Dans ce chapitre, nous avons proposé un protocole de loterie électronique et un schéma de vote électronique. Chacun de ces protocoles utilise une version de Paillier partagée différente.

Ce chapitre montre que les propriétés homomorphiques d'un schéma de chiffrement sont utiles : elles permettent de calculer avec des données chiffrées sans avoir besoin de les déchiffrer. Pour les applications pratiques, le cryptosystème de Paillier présente les meilleures caractéristiques : déchiffrement partagé et meilleure bande passante.

8

Application au recouvrement de clé

Dans ce chapitre, nous présentons un protocole de recouvrement de clé. Ce protocole a été implémenté dans le projet OPPIDUM Confiance [68] en partenariat avec le Laboratoire d'Informatique de l'École Normale Supérieure pour la partie description du système de recouvrement, la société CS Systèmes d'Information qui a développé le produit, et les utilisateurs testeurs : la Abel SA et le Conseil Supérieur du Notariat (CSN). Le produit TrustyRecovery a été développé et intégré dans le produit PKI de la société TrustyCom.

Nous décrivons la problématique du recouvrement de clé, différents types de solutions à ce problème, le système de recouvrement que nous avons proposé dans ce projet et la description du produit.

Sommaire

8.1 Problématique du recouvrement de clé

Dans cette section, nous présentons les buts du recouvrement, les différentes techniques de recouvrement et leurs limites.

8.1.1 Motivations

Dans les années 90, le recouvrement de clé a été le sujet de nombreuses discussions, de controverses, et d'intenses recherches accélérées par le développement rapide des réseaux de communication comme l'Internet. De plus, la mise en place d'infrastructures de gestion de clés publiques est nécessaire pour

68. Ce projet a été financé en partie par le Ministère de l'Économie, des Finances et de l'Industrie.

gérer les clés de signature et de chiffrement permettant les échanges sécurisés entre des utilisateurs qui ne se connaissent pas [69]. Cependant, une libéralisation complète de l'utilisation de la cryptographie à des fins de chiffrement n'est pas complètement acceptée par les gouvernements [70]. "La principale objection généralement faite à la banalisation de la cryptologie est qu'elle ouvre les portes de ce savoir au crime organisé et aux puissances hostiles", comme le dit Jacques Stern dans [179].

La deuxième motivation au recouvrement de clés est que les utilisateurs peuvent perdre leur clé de déchiffrement et qu'ils sont les seuls à détenir ces clés. Si cette clé est perdue, par exemple suite à la perte de la carte à puce contenant cette clé ou suite à une destruction du disque dur, les données stockées sur un serveur de fichiers ou sur un poste de travail ne pourront plus être déchiffrées. De même, en cas d'urgence, si seul le médecin traitant détient la clé de déchiffrement et qu'il est en vacances, les données concernant un patient ne pourront pas être lues par le médecin aux urgences. On voit ici, qu'il serait pratique d'avoir un double de ces clés ou des clés utilisables par un groupe d'utilisateurs.

Plusieurs problèmes se posent : d'une part, plus il y a de doubles des clés, plus les données chiffrées pourront être lues par différentes personnes et moins la confidentialité sera assurée. D'autre part, plus il y a de doubles, plus les données chiffrées seront vulnérables. Il s'agit donc de trouver un moyen terme entre ces deux extrêmes.

La complexité technique de ce problème a entraîné la publication de nombreuses propositions, malheureusement bien peu satisfaisantes. Certaines font un usage intensif de *tierces parties de confiance*. D'autres utilisent un composant matériel, comme une carte à puce, considéré inviolable. Dans un système de recouvrement, un tiers de confiance est une autorité reconnue par les utilisateurs. Cette autorité a la possibilité de récupérer les documents en clair. Il existe des autorités de recouvrement particulières appelées *autorités de séquestre* dont la mission est de conserver les clés privées ou les clés de session des utilisateurs.

L'absence de solutions techniques satisfaisantes durant les années 90 a ouvert la porte à la libéralisation de la cryptographie, poussée par le besoin de communiquer sur des réseaux ouverts, où la protection des informations est nécessaire. En effet, le développement du commerce électronique est fortement dépendant de la sécurisation du réseau Internet. L'économie a pris le pas sur la sécurité nationale. Ainsi, de nombreux logiciels de chiffrement existent aujourd'hui et sont libres d'utilisation. Les législations ont donc renforcé les peines encourues par les contrevenants et augmenté les pouvoirs des autorités judiciaires et administratives. Cette démarche semble être raisonnable car la majorité des personnes font un usage légal de la cryptographie. Les criminels ne sont pas les plus nombreux, et de toute façon, ils sont suffisamment puissants pour communiquer avec des moyens cryptologiques même si les gouvernements les interdisaient. Ainsi, une limitation de l'utilisation de la cryptologie aurait abouti à des effets contraignants pour attraper une minorité de personnes qui aurait échappé aux interceptions.

Par conséquent, le recouvrement de clé est utile pour d'une part, protéger une entreprise d'un employé indélicat qui utiliserait les produits mis à sa disposition dans l'entreprise pour un usage personnel non légal et d'autre part pour récupérer un document en cas de perte, de dysfonctionnement d'un disque dur, ou d'absence d'un employé.

69. Dans le cas où les utilisateurs se connaissent, ils existent d'autres solutions que les PKI pour communiquer en toute sécurité en utilisant la cryptographie asymétrique. Les PKI sont en effet des produits complexes à gérer et dans le cas de petites communautés d'utilisateurs, une telle administration n'est pas toujours justifiée.

70. Le projet de Loi sur la Société de l'Information, accepté au Conseil des Ministres du 13 juin 2001, prévoit une libéralisation totale de l'utilisation de la cryptologie en France. Parallèlement, ce projet de loi prévoit un renforcement des sanctions, si un moyen de cryptologie est utilisé pour préparer ou commettre un crime ou un délit ou pour en faciliter la préparation, et si l'auteur ou le complice de l'infraction, sur demande des autorités judiciaires ou administratives, refuse de remettre la version en clair des messages chiffrés ainsi que les conventions secrètes nécessaires au déchiffrement. Dans ce cas, l'auteur ou le complice risque de voir doubler sa peine.

8.1.2 Différentes techniques de recouvrement

Présentons tout d'abord la technique de chiffrement utilisée dans les produits de sécurité du commerce, puis les techniques de recouvrement.

Technique de chiffrement hybride

Supposons qu'Alice veuille envoyer un message m secret à Bob. Elle génère une clé de session[71], k_s, chiffre m avec un algorithme symétrique sous k_s, obtient $b = \mathbf{SymEnc}_{k_s}(m)$, et envoie k_s chiffré sous la clé publique de Bob, $a = \mathcal{E}_{\mathsf{pk}_B}(k_s)$:

$$c = (a, b) = (\mathcal{E}_{\mathsf{pk}_B}(k_s), \mathbf{SymEnc}_{k_s}(m))$$

En réception, Bob déchiffre a avec sa clé secrète, obtient la clé de session k_s, puis déchiffre b avec cette clé, ce qui lui donne le document m.

En effet, le document envoyé peut être volumineux et comme la cryptographie symétrique est plus rapide que la cryptographie asymétrique, le produit de chiffrement sera plus rapide. Ensuite, on utilise la cryptographie asymétrique car si les personnes ne se connaissent pas, cette dernière permet de résoudre le problème de l'échange de clé puisque tout utilisateur peut avoir accès à la clé publique de Bob. On rappelle que dans un système symétrique, émetteur et récepteur utilisent la même clé pour chiffrer et déchiffrer. Ainsi, l'utilisation conjointe de la *cryptographie symétrique* pour chiffrer les données et de la *cryptographie asymétrique* pour échanger la clé de session, est appelée un *chiffrement hybride*.

Techniques de recouvrement

Techniquement, il existe plusieurs formes de recouvrement. On peut vouloir retrouver le clair d'un document, m, la clé de session chiffrant un document, k_s, ou la clé privée d'un utilisateur, sk_U[72]. Il est clair que le recouvrement de la clé privée est une solution qui donne autant de pouvoir que l'utilisateur légitime, et qui permet par conséquent de retrouver la clé de session et le document en clair. Ce pouvoir est immense car l'autorité peut alors retrouver tout fichier.

Il existe deux grands systèmes de recouvrement, appelés respectivement système de séquestre et système d'encapsulation. Le premier consiste à stocker la clé secrète sk_U ou les clés de session k_s des utilisateurs dans l'autorité de recouvrement. Le stockage des clés de session n'est pas une solution pratique sauf dans le cas de systèmes centralisés comme le système Kerberos. La deuxième solution consiste à utiliser un chiffrement double. L'émetteur envoie le chiffré suivant :

$$c = (\mathcal{E}_{\mathsf{pk}_U}(k_s), \mathcal{E}_{\mathsf{pk}_A}(k_s), \mathbf{SymEnc}_{k_s}(m))$$

Ainsi, le receveur qui a la clé secrète sk_U ou l'autorité qui a la clé secrète sk_A peuvent tous deux obtenir le message m. Le problème de cette technique est qu'elle augmente la taille du chiffré c. Elle est actuellement implémentée dans EFS,[73] le système de fichiers sécurisés de Windows 2000. Cette technique a été soutenue par le Key Recovery Alliance (KRA) qui regroupait de nombreux industriels. Elle a beaucoup d'avantages, mais présente des inconvénients en cas de compromission ou de renouvellement de la clé de l'autorité. Ainsi, en cas de "crash" du système, EFS risque de ne plus pouvoir déchiffrer les fichiers !

71. Cette clé est dite de *session*, car Alice en change à chaque message.

72. Dans ce dernier cas, il s'agit par exemple d'un utilisateur qui a perdu sa clé et qui possède plusieurs fichiers à déchiffrer. On remarque qu'il n'est pas nécessaire de récupérer la clé de signature. En effet, il suffit de générer une nouvelle clé privée car on n'a pas besoin de déchiffrer d'anciens fichiers. De plus, la législation sur la signature électronique impose que la clé de signature ne soit conservée que par son détenteur. Ainsi, il est nécessaire qu'un produit de PKI sépare les clés de chiffrement et les clés de signature pour chaque utilisateur.

73. Encrypted File System

La deuxième distinction que l'on peut faire est entre un recouvrement on-line et un recouvrement off-line. Dans le premier cas, il s'agit d'une communication interactive chiffrée entre deux personnes ou entre deux machines (protocole client/serveur sécurisé utilisant le protocole SSL/TLS ou les IPSec suivant l'emplacement où a lieu le chiffrement dans la pile de communication réseau). Le recouvrement *off-line* correspond à un fichier chiffré stocké sur un disque ou envoyé par mail. Le lecteur pourra lire notre article [71] pour plus de précisions sur ce problème.

8.1.3 Limites du recouvrement

Aucune solution technique n'est satisfaisante, car deux utilisateurs qui tentent de frauder auront toujours la possibilité d'échanger une conversation secrète. En effet, on aurait souhaité construire une preuve cryptographique que le clair d'un document est recouvrable, car chiffré sous une clé symétrique. Par exemple, il est possible de faire de telles preuves pour prouver que la clé secrète sk_U associée à pk_U est chiffrée sous la clé publique pk_A d'une autorité et donc que le recouvrement de la clé privée pourra être effectué. Cependant, rien ne garantit le recouvrement du fichier. En effet, il suffit que l'émetteur et le récepteur se soient entendus sur une autre clé de session, k'_s et chiffrent le document m avec k'_s avant de le chiffrer avec k_s. Si l'émetteur envoie $c = (\mathcal{E}_{\mathsf{pk}_U}(k_s), \mathbf{SymEnc}_{k_s}(\mathbf{SymEnc}_{k'_s}(m)))$, le receveur qui connaît k'_s pourra obtenir le document en clair alors que l'autorité, qui ne connaît que sk_U associée à pk_U obtiendra $\mathbf{SymEnc}_{k'_s}(m)$, mais ne pourra pas obtenir m.

Il est clair que l'on ne peut pas empêcher deux fraudeurs d'utiliser un tel système aujourd'hui car les produits de chiffrement logiciels sont nombreux. Un mécanisme basé sur une carte à puce ou un "secure device", comme le système Clipper, aurait certes pu empêcher cette technique de surchiffrement car les fraudeurs n'auraient pu utiliser que des clés recouvrables par l'autorité, et le chiffrement n'aurait pu s'effectuer que dans la carte.

De nombreuses propositions de recouvrement ont été faites mais aucune ne parvient à éliminer le problème du surchiffrement. La seule chose que l'on peut garantir est que la clé secrète d'un utilisateur, sk_U, peut être retrouvée mais il est impossible de garantir que l'on obtiendra le fichier en clair. Pour garantir que la clé est recouvrable, on peut utiliser les solutions suivantes.

Young et Yung ont proposé à Eurocrypt '98 [185] une solution de recouvrement intégrée dans les PKI. Au moment où les utilisateurs s'enregistrent auprès d'une autorité de certification pour faire certifier une clé publique pk_U, chiffrent en même temps la clé secrète sk_U sous la clé publique de l'autorité, et obtiennent le chiffré $c = \mathcal{E}_{\mathsf{pk}_A}(\mathsf{sk}_U)$. Puis, ils font une preuve zero-knowledge prouvant que la clé chiffrée dans c correspond à la clé secrète associée à la clé publique pk_U. Ainsi, ils introduisent la notion de clé *rouge*, tant que la clé n'a pas été déchiffrée par l'autorité, et de clé *noire*, si sk_U a été déchiffrée par l'autorité. Ce système de recouvrement est appelé auto-certifiable et auto-recouvrable car l'autorité peut vérifier si le demandeur d'un certificat connaît la clé secrète associée et peut vérifier qu'elle pourra déchiffrer c pour obtenir la clé secrète si besoin est. Sinon, la clé restera *rouge*.

Enfin, Poupard et Stern ont étendu ce modèle à Eurocrypt '00 [156] en faisant des preuves zero-knowledge plus compactes de façon à ce qu'elles puissent être incluses dans le certificat de clé publique. Ainsi, tout utilisateur peut vérifier que la clé secrète est recouvrable. Cette preuve peut alors être vérifiée par le produit de chiffrement afin d'appliquer une politique de sécurité. Par exemple, une entreprise pourrait imposer que ses employés n'utilisent que des clés qu'elle puisse retrouver. Ainsi, avant de chiffrer, le produit vérifierait la preuve et ne chiffrerait le document que si cette preuve est valide. Poupard et Stern ont utilisé l'idée d'un certificat *recouvrable*, c'est-à-dire d'un certificat dont la clé secrète peut être retrouvée. Cette idée est essentielle dans les PKI d'aujourd'hui et est recommendée par le NIST dans les normes FPKI (Federal PKI)[74] mais n'empêche pas le surchiffrement.

74. `http://csrc.nist.gov/pki/twg/welcome.html`

8.2 Nouvelle solution de recouvrement

Le produit de recouvrement construit dans le projet OPPIDUM Confiance n'a pas pour objectif de contrer les fraudeurs qui tenteraient de surchiffrer. Ces derniers peuvent toujours le faire, car le produit ne peut pas garantir cette malversation. Cependant, dans ce cas, ils sont en infraction par rapport à la politique de sécurité de l'entreprise qui doit spécifier clairement cette impossibilité. En effet, même si l'outil de l'entreprise vérifie que les certificats sont recouvrables, des fraudeurs peuvent installer d'autres produits de chiffrement qui ne vérifient pas cette extension du certificat ou qui utilisent d'autres clés de chiffrement qui n'ont pas cette extension. Ainsi, le contournement est très facile et nous ne pouvons pas l'empêcher. Nous avons donc recours à des mesures organisationnelles et à des politiques de sécurité.

Nous utilisons une solution de *séquestre*, ou plus exactement de *sauvegarde* de la clé privée des utilisateurs car l'autorité est incluse dans le produit PKI de l'entreprise et n'est pas externalisée.

8.2.1 Certificats recouvrables

La solution retenue utilise des certicats recouvrables générés par le produit PKI de la société TrustyCom. Une extension X.509 est ajoutée contenant essentiellement un bit qui indique la qualité de ce certificat : recouvrable ou non.

8.2.2 Séparation des autorité de recouvrement et de certification

Un utilisateur dans un système PKI fait une demande de clés publique et secrète à une autorité de certification. Elle génère les clés, chiffre la clé secrète sk_U de l'utilisateur sous la clé publique de l'autorité de recouvrement $\mathcal{E}_{\mathsf{pk}_A}(\mathsf{sk}_U)$, et fait une demande d'archivage auprès de l'autorité de recouvrement. Cette dernière archive la clé chiffrée (rouge) et renvoie une preuve d'archivage indiquant que la clé a été correctement archivée. L'autorité de certification enregistre cette preuve signée qui lui servira de preuve, au cas où on lui reprocherait d'avoir délivrée un certificat "recouvrable" pour une clé "non recouvrable". Enfin, l'autorité retourne un certificat recouvrable à l'utilisateur.

Ainsi, toute la confiance repose sur la clé secrète de l'autorité de recouvrement sk_A. Cette clé est partagée pour instaurer plus de confiance dans le système.

8.2.3 Recouvrement de fichiers

Pour le recouvrement du document chiffré $c = (a, b) = (\mathcal{E}_{\mathsf{pk}_U}(k_s), \mathbf{SymEnc}_{k_s}(m))$, un agent de recouvrement, le logiciel client installé sur le poste de l'officier de sécurité par exemple, permet d'interroger le serveur de recouvrement en envoyant a et la clé publique de l'officier pk_O. Le serveur vérifie que cette personne O a le droit de recouvrer les documents pour l'utilisateur U. Elle déchiffre sk_U, puis déchiffre a, rechiffre k_s sous la clé publique pk_O et renvoie $a' = \mathcal{E}_{\mathsf{pk}_O}(k_s)$. L'agent de recouvrement reconstruit le nouveau message chiffré $c' = (\mathcal{E}_{\mathsf{pk}_O}(k_s), \mathbf{SymEnc}_{k_s}(m))$ et le donne au même produit de chiffrement que l'utilisateur U. Le logiciel ne s'aperçoit pas du subterfuge et déchiffre le fichier pour l'officier O. L'avantage de cette solution est que le serveur de recouvrement n'a pas accès au fichier que l'officier de recouvrement veut déchiffrer et réciproquement, l'officier de recouvrement n'a pas accès à la clé secrète de l'utilisateur U. De plus, nous n'avons pas besoin de modifier le protocole de chiffrement de fichier. Le message échangé est le même que dans n'importe quel autre produit et est conforme à la norme CMS[75]. Enfin, cette solution de certificat recouvrable permet aussi de faire du recouvrement on-line comme dans SSL par exemple.

75. Cryptographic Message Syntax, rfc 2630.

8.3 Conclusion

Cette solution de recouvrement a été développée dans le cadre du projet OPPIDUM Confiance. Le partage de la clé de l'autorité de recouvrement est nécessaire car il s'agit d'une clé très sensible de même que dans une solution d'encapsulation. Cependant, contrairement à cette solution, un adversaire n'a pas accès aux clés chiffrées $\mathcal{E}_{\mathsf{pk}_A}(\mathsf{sk}_U)$, alors que dans une solution d'encapsulation, il a accès à $\mathcal{E}_{\mathsf{pk}_A}(k_s)$.

L'autorité de recouvrement utilise des clés RSA de 2048 bits. On sait que le système RSA partagé est résistant aux attaques à clairs choisis, IND-CPA. Ceci est nécessaire dans le cadre de ce produit car un attaquant ne peut pas monter une attaque à chiffrés choisis. En effet, l'agent de recouvrement, ne fait pas des demandes de déchiffrement de clés. Pourtant, il serait utile d'avoir un système résistant à ce type d'attaque pour le système RSA.

Dans le cas où l'on utiliserait des certificats recouvrables contenant $\mathcal{E}_{\mathsf{pk}_A}(\mathsf{sk}_U)$, comme dans le système de recouvrement de Poupard et Stern, le système devrait résister à ce type d'attaque. En effet, l'agent de recouvrement fait des demandes de déchiffrement à l'autorité de recouvrement. Dans ce système, le cryptosystème utilisé est celui de Paillier que nous avons protégé contre ce type d'attaque dans le chapitre 5.

Conclusion

Dans cette thèse, nous avons étudié certains aspects théoriques et pratiques du partage de clé cryptographique. La cryptographie partagée est la composante de la cryptographie qui s'intéresse aux calculs multiparties (MPC) et qui propose des schémas *efficaces* en pratique pour distribuer des *fonctions particulières*. L'approche MPC est plus générale et fournit des solutions pour des *classes de fonctions* au sens de la théorie des calculs et de la complexité. La cryptographie à seuil permet de protéger une clé contre des attaquants plus forts que ceux considérés dans les systèmes classiques "centralisés". De plus, elle permet de répartir les décisions parmi un ensemble de personnes de telle sorte que plusieurs d'entre elles soient impliquées pour prendre une décision. Ceci permet de distribuer la confiance pour atteindre un degré de sécurité plus important, l'attaquant doit en effet corrompre plus de personnes pour frauder.

Dans la première partie de cette thèse nous avons présenté des techniques de partage de secret et nous avons rappelé les modèles de sécurité pour prouver la sécurité des schémas partagés de chiffrement et de signature.

Puis dans la deuxième partie, nous avons étudié des techniques pour *complètement* partager la signature RSA et le système de chiffrement de Paillier. Les protocoles décrits sont sûrs contre des attaques à clairs choisis et à chiffrés choisis. Nous avons aussi présenté une méthode pour générer une clé Diffie-Hellman de manière distribuée ce qui permet de *complètement partager* des protocoles comme le système de chiffrement El Gamal ou le schéma de signature DSA.

Enfin, nous avons décrit dans la troisième partie, les applications de ces partages à des systèmes sensibles comme un système de loterie, de vote électronique, ou de recouvrement de clé.

Cette thèse suggère trois problèmes ouverts essentiels :

– est-il possible de partager RSA de manière sûre contre les attaques CCA ?

– est-il possible de construire un schéma IND-CCA dans le modèle standard basé sur la factorisation ?

– est-il possible de calculer efficacement modulo un secret partagé ?

Un schéma partagé du cryptosystème RSA sûr contre des attaques à chiffrés choisis serait utile en pratique pour construire une autorité de recouvrement de clé. Même dans le modèle de l'oracle aléatoire, ce problème semble être difficile, comme le mentionnent Gennaro et Shoup dans [175]. Nous avons proposé dans cette thèse un partage du système de chiffrement de Paillier, basé sur une hypothèse reliée à la factorisation alors que jusqu'à présent, les systèmes proposés à seuil IND-CCA étaient tous reliés au problème du calcul du logarithme discret. La preuve de sécurité du schéma proposé utilise le modèle de l'oracle aléatoire.

Par conséquent, le deuxième problème semble difficile. Notre schéma de Paillier partagé sûr contre des attaques CCA utilise des preuves de validité *publiquement vérifiables*. Une piste de recherche serait d'utiliser des preuves de validité qui ne soient vérifiables que par le possesseur de la clé privée. Ceci est le cas du cryptosystème Cramer-Shoup [55], seul système de chiffrement efficace prouvé sûr dans le modèle standard sans hypothèse supplémentaire.

Enfin, un problème difficile aujourd'hui est le calcul de $b \bmod p$ où p et b sont deux secrets partagés. Ceci pourrait être un premier pas vers un algorithme de génération de nombres premiers sûr en utilisant par exemple l'algorithme de Miller-Rabin partagé. Le seul calcul que l'on sait faire aujourd'hui est le calcul d'inverse modulo un nombre secret partagé. En effet, Catalano, Gennaro et Halevi dans [40] ont construit un tel protocole sûr. On a vu dans cette thèse que cet algorithme permet de calculer le pgcd d'une quantité publique et d'une quantité secrète. Une voie de recherche serait de partager le cryptosystème d'Okamoto et Uchiyama [136] puisque le déchiffrement de ce cryptosystème permet d'obtenir le message modulo p. Enfin, le problème est peut-être plus facile en se limitant à un partage entre deux serveurs.

Annexes

9

Outils pour le partage du cryptosystème RSA

Dans cette annexe, nous décrirons différentes preuves de robustesse pour RSA et différents algorithmes utilisés dans le partage de RSA.

9.1 Différentes preuves de validité

Nous présentons ici deux preuves de validité qui peuvent être utilisée à la place de la preuve de validité de Shoup. La première preuve que nous présentons est celle de Frankel *et al.* qui n'a pas besoin de nombres premiers sûrs et qui peut donc être utilisée pour remplacer la preuve de Shoup[76]. Il s'agit d'une preuve d'égalité de logarithme discret *interactive*, prouvée sûre dans le modèle standard. Puis, nous décrivons la preuve de Gennaro *et al.* [89]. Il s'agit d'une preuve *non-interactive prouvée sûre dans le modèle standard* mais qui suppose une certaine relation entre le prouveur et le vérifieur. Cette preuve n'est pas une preuve d'égalité de logarithmes discrets, mais est une preuve d'appartenance à un certain langage que nous décrirons.

9.1.1 Preuve interactive d'égalité de logarithmes discrets dans un groupe non cyclique d'ordre inconnu

Frankel, Gemmel, Mac Kenzie, et Yung ont décrit dans [77], un protocole interactif efficace qui peut être prouvé sans faire appel à l'oracle aléatoire et qui ne fait pas d'hypothèse sur le module RSA. Le prix à payer étant l'interactivité du protocole (cf. figure 9.1).

Preuve: Cette preuve est effectuée pour plusieurs générateurs g car comme on l'a dit précédemment, $\mathbb{Z}_N^*$ n'est pas un groupe cyclique et trouver des éléments d'ordre maximal $(\lambda(N))$ n'est pas évident. Cependant, en prenant plusieurs éléments, avec forte probabilité, on génère $\mathbb{Z}_N^*$.

Le premier message du vérifieur permet de faire une preuve efficace en quatre tours et d'imposer un vérifieur honnête car son choix sera indépendant du premier message du prouveur.

Consistance. La consistance de ce protocole est immédiate.

Significatif. La preuve est significative car un prouveur malhonnête ne peut pas tricher sauf en devinant le challenge $\hat{c}$ ou en cassant la fonction de Pedersen [144] de mise en gage $Commit$, donc avec probabilité

76. Ceci nous permet donc de distribuer complètement RSA. Cependant, la preuve de Shoup présente des caractéristiques intéressantes comme le fait qu'elle soit non-interactive. De plus, des éléments dans la preuve de Frankel *et al.* n'ont pas été complètement étudiés comme le fait de savoir combien de générateurs différents faut-il prendre pour engendrer un sous-groupe d'ordre maximal de $\mathbb{Z}_N^*$. Ceci a été fait par Poupard et Stern dans [157].

Prouveur	$\mathsf{vk}_i = g^{d_i} \bmod N$	Vérifieur
$d_i \in [0, \varphi(N)[$		
$\sigma_i = x^{d_i} \bmod N$		
		Met en gage $\hat{c} = c_1 \| \ldots \| c_h$
	$\xleftarrow{Commit}$	$Commit = g^{\hat{c}} g_2^{q_0}$ où $q_0 \in_R [0, N[$
Choisit $q_j \in_R [0, X[$ où $X = 2\Delta^3 N^{3+\epsilon} t^2 n^{t+n} + nN^2$		
Calcule $t_1 = (g^{q_1}, \ldots, g^{q_h})$		
$t_2 = (x^{q_1}, \ldots, x^{q_h})$	$\xrightarrow{t_1 \| t_2}$	
	$\xleftarrow{B_1 \text{ et } B_2}$	$B_1 = \hat{c}$ et $B_2 = q_0$
Vérifie si $g^{B_1} g_2^{B_2} \stackrel{?}{=} Commit$		
$\forall j = 1$ à h, $v_j = c_j d_i + q_j$	$\xrightarrow{\sigma_i \text{ et } (v_1, \ldots, v_h)}$	
		Vérifie si $\forall j = 1$ à h, $\mathsf{vk}_i^{c_j} g^{q_j} \stackrel{?}{=} g^{v_i} \bmod N$ $\sigma_i^{c_j} x^{q_j} \stackrel{?}{=} x^{v_i} \bmod N$

FIG. 9.1: *Preuve de validité interactive dans le modèle standard de Frankel et al..*

$1/2^h$.

Statistiquement Zero-Knowledge. La simulation zero-knowledge peut facilement être effectuée en temps polynomial (en deux passes). Le simulateur après le message $Commit$ choisit $t_1 \| t_2$ au hasard et reçoit $\hat{c}$ décommité. Le simulateur rembobine juste après le premier message et comme il sait quel random choisira le combineur sauf s'il connaît $\log_g g_2$, il peut simuler la preuve en choisissant au hasard $(v_1, \ldots, v_h)$ et en calculant t_1 comme $(g^{v_1} \mathsf{vk}_i^{-c_1}, \ldots, g^{v_h} \mathsf{vk}_i^{-c_h})$ et t_2 comme $(x^{v_1} \sigma_i^{-c_1}, \ldots, x^{v_h} \sigma_i^{-c_h})$. □

9.1.2 Preuve de validité non-interactive sûre dans le modèle standard

La preuve de validité de Gennaro *et al.* est décrite dans la figure 9.2. Il s'agit d'une preuve d'appartenance à un langage. Le langage n'est pas celui des couples ayant le même logarithme discret, mais il est constitué des paires $(\sigma_i, Y) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$ telles que $\sigma_i^b x^c = Y \bmod N$. Seul un vérifieur connaissant b et c peut vérifieur ces preuves.

Description de la preuve. Considérons deux joueurs P et V. P génère une signature que V peut vérifier. Cette preuve utilisera des valeurs auxiliaires pour P et V. Le prouveur P détient la clé secrète d_i et génère une sorte de clé de vérification y. Le vérifieur V détient b et c tels que $y = bd_i + c$ dans les entiers. Les valeurs d_i, y, b et c sont partagées entre les parties durant la phase de génération des clés (et gardées secrètes par les parties). Étant donné un message M, et $x = H(M)$, le prouveur génère une part de signature $\sigma_i = x^{d_i} \bmod N$, et l'information supplémentaire de vérification $Y = x^y \bmod N$. P envoie ces valeurs à V qui vérifie la part de signature en calculant $(\sigma_i)^b x^c = (x^{d_i})^b x^c \stackrel{?}{=} x^y = Y \bmod N$.

Cette preuve suppose l'existence d'un distributeur D qui prend en entrée un module RSA N, une part d_i de la clé secrète d, et des paramètres de sécurité $0 \leq k_1, k_2, k_3 = k_1 + k_2 + \log N$. Il choisit

Prouveur		Vérifieur
$d_i \in [0, \varphi(N)[, y = bd_i + c$		$b \in [0, 2^{k_1}[, c \in [0, 2^{k_3}[$
$\sigma_i = x^{d_i} \bmod N$ et $Y = m^y \bmod N$	$\xrightarrow{(\sigma_i, Y)}$	vérifie si $\sigma_i^b x^c \stackrel{?}{=} Y \bmod N$

FIG. 9.2: *Preuve de validité non-interactive dans le modèle standard de Gennaro et al..*

$b \in [0, 2^{k_1}[$ et $c \in [0, 2^{k_3}[$ avec probabilité uniforme. Puis il calcule $y = bd_i + c$ et transmet secrètement d_i et y à P puis b et c à V.

Les preuves présentes dans cette section considèrent l'équation $y = bd_i + c$ dans les entiers. L'approche plus naturelle de générer cette équation modulo $\varphi(N)$ pourrait permettre à P et V de combiner leur information et de calculer un multiple de $\varphi(N)$, permettant de factoriser N. Une part de signature σ_i sera acceptée par le combineur soit si $\sigma_i = x^{d_i} \bmod N$, soit si $\sigma_i = -x^{d_i} \bmod N$. Dans le cas d'une signature, ce n'est pas un problème sérieux. Une part de signature incorrecte influencera seulement le signe de la signature finale, ce que l'on peut décider en utilisant l'exposant public.

Présentons avant tout les deux lemmes suivant. On représente par $\mathcal{Y}$ l'ensemble $[2^{k_1+\log N}, 2^{k_3}[$.

Lemme 7 *Étant données $d_i \in [0, \varphi(N)[$ et $y \in \mathcal{Y}$, pour chaque valeur possible de $b \in [0, 2^{k_1}[$, il existe exactement une valeur possible pour $c \in [0, 2^{k_3}[$ telle que $y = bd_i + c$.*

Preuve: Comme le calcul est effectué dans les entiers, il existe exactement une valeur de c pour chaque b, d_i et y. De plus, si $b \in [0, 2^{k_1}[$, et $y \in [2^{k_1+\log N}, 2^{k_3}[$, alors la valeur $c = y - bd_i$ est contenue dans $[0, 2^{k_3}[$. En effet :

$$\begin{aligned} y_{min} - b_{max} d_{i,max} &\leq c \leq y_{max} - b_{min} d_{i,min} \\ 1 \leq 2^{k_1+\log N} - 2^{k_1}\varphi(N) &\leq c \leq 2^{k_3} - 1 \end{aligned}$$

ce qui prouve le lemme. □

Lemme 8 *Pr* $[y \notin \mathcal{Y}] \leq \frac{1}{2^{k_2}}$.

Preuve: Le nombre total de paires (b, c) différentes est $2^{k_1}2^{k_3}$. L'intervalle $\mathcal{Y}$ est de taille $2^{k_3} - 2^{k_1+\log N}$. D'après le lemme 7, chaque valeur y dans cet intervalle peut être générée par 2^{k_1} paires (b, c). Par conséquent, comme b et c sont choisis avec probabilité uniforme, la probabilité que y tombe en dehors de cet intervalle est $1 - \frac{(2^{k_3} - 2^{k_1+\log N})2^{k_1}}{2^{k_3}2^{k_1}} = \frac{1}{2^{k_2}}$. □

Ils ont prouvé le théorème suivant :

Théorème 28. Soit $N = pq$, où $p < q$, $p = 2p' + 1$, $q = 2q' + 1$ et p, q, p', q' sont tous des nombres premiers. Supposons un prouveur $\tilde{P}$ qui triche et qui ne peut pas casser RSA pour un module N (en particulier, il ne connait pas et ne peut pas calculer la factorisation de N).
Consistance. Si P et V suivent le protocole, alors V accepte toujours une part de signature.
Significative. Un prouveur tricheur $\tilde{P}$ peut convaincre V d'accepter $\tilde{\sigma}_i \neq \pm x^{d_i} \bmod N$, avec probabilité au plus $\frac{1}{p'} + \frac{1}{2^{k_1}} + \frac{1}{2^{k_2}}$.
Zero-Knowledge. V n'apprend aucune information au-delà de la signature $\sigma_i = x^{d_i} \bmod N$, *i.e.* étant donné σ_i, b, c il existe un algorithme en temps polynomial pour calculer Y.

Preuve: Consistance. Immédiate.

Significative. Commencons par examiner le cas où $y \in \mathcal{Y} = [2^{k_1+\log N}, \ldots, 2^{k_3}]$. On remarque que le vérifieur utilise un algorithme déterministe pour accepter ou rejeter la paire publique (σ_i, Y). Par conséquent, la probabilité établie dans le théorème est prise sur l'ensemble des couples $(b, c) \in [0, 2^{k_1}[\times[0, 2^{k_3}[$ tels que $y = bd_i + c$. Afin que le prouveur P puisse convaincre V d'accepter la preuve (σ_i, Y), on doit avoir $Y = \sigma_i^b x^c \bmod N$. Or, d'après le protocole de génération $y = bd_i + c$. Par conséquent, $x^y = (x^{d_i})^b x^c \bmod N$. En divisant ces deux équations, on obtient :

$$Yx^{-y} = (\sigma_i x^{-d_i})^b \bmod N \tag{9.1}$$

Ceci signifie que Yx^{-y} appartient au sous-groupe $< \sigma_i x^{-d_i} >$. Soit k la valeur minimale telle que $Yx^{-y} = (\sigma_i x^{-d_i})^k \bmod N$. L'équation 9.1 est satisfaite si $b = k \bmod \text{ord}(\sigma_i x^{-d_i})$. Comme b est choisi au hasard avec une distribution uniforme dans $[0, 2^{k_1}[$, la probabilité qu'une paire (σ_i, Y) vérifie l'équation 9.1 est telle que :

$$\begin{aligned} \Pr\left[b = k \bmod \text{ord}(\sigma_i x^{-d_i})\right] &\leq \frac{\left\lceil \frac{2^{k_1}}{\text{ord}(\sigma_i x^{-d_i})} \right\rceil}{2^{k_1}} \\ &\leq \frac{1}{\text{ord}(\sigma_i x^{-d_i})} + \frac{1}{2^{k_1}} \end{aligned}$$

Comme N est un module RSA *sûr*, il existe seulement quatre éléments de $\mathbb{Z}_N^*$ dont l'ordre est plus petit que p'. Ce sont les racines carrées de l'unité. En effet, les éléments de $\mathbb{Z}_N^*$ ont comme ordre possible un diviseur de $\lambda(N) = 2p'q'$, c'est-à-dire $2, p', q', 2p', 2q', p'q'$, ou $2p'q'$. Si $\sigma_i x^{-d_i} = \pm 1 \bmod N$, alors $\sigma_i = \pm x^{d_i} \bmod N$. Si le prouveur pouvait trouver σ_i tel que $\sigma_i x^{-d_i}$ soit une racine non triviale de l'unité, $(\neq \pm 1)$, alors il pourrait factoriser N, ce qui est supposé infaisable. Pour tous les autres choix de σ_i, $\text{ord}(\sigma_i x^{-d_i}) \geq p'$. Ceci termine la preuve dans le cas $y \in \mathcal{Y}$. Cependant, d'après le lemme 8, on sait que la probabilité que $y \notin \mathcal{Y}$ est au plus $\frac{1}{2^{k_2}}$, en combinant ces deux probabilités, on obtient la probabilité annoncée.

Zero-Knowledge. Connaissant b, c et $\sigma_i = x^{d_i} \bmod N$, le vérifieur peut calculer $Y = x^y = \sigma_i^b x^c \bmod N$. (On utilise ici un vérifieur honnête pour V car la preuve est non-interactive.) □

9.1.3 Comparaison

L'avantage de la preuve de Shoup par rapport à la preuve de Gennaro *et al.* est d'éviter d'avoir une relation entre le vérifieur et le prouveur. L'avantage de la preuve de validité de Gennaro *et al.* est qu'elle

n'utilise pas le modèle de l'oracle aléatoire tout en étant *non-interactive*. Dans le cas où il existe un seul combineur, il vaut mieux utiliser la preuve de validité de Gennaro *et al.*. Cependant, dans le cas où tous les utilisateurs peuvent faire des requêtes de signature, le protocole de Shoup est utile. La preuve de Frankel *et al.* est utile si on veut utiliser un module RSA n'ayant pas une forme spécifique. Elle fait toutefois l'hypothèse qu'en prenant suffisamment d'éléments au hasard dans $\mathbb{Z}_N^*$ on génèrera $\mathbb{Z}_N^*$ en entier, mais l'analyse n'a pas été faite. Dans la suite de ce chapitre, nous verrons une analyse de cette hypothèse avec des modules ayant certaines particularités.

D'autre part, dans le cas de signature RSA, comme RSA-FDH [11, 51] par exemple, la preuve de sécurité utilise déjà le modèle de l'oracle aléatoire, on peut alors utiliser le protocole de Shoup. Mais, dans le cas de la signature de Gennaro, Halevi et Rabin [87] qui n'utilise pas l'oracle aléatoire, les preuves de validité de Gennaro *et al.* et celle de Frankel *et al.* permettent de distribuer ce schéma de signature sans faire appel à l'oracle aléatoire mais en utilisant le Flexible-RSA Problem qui est un problème plus facile à résoudre que le problème RSA[77]. Ceci a été fait par Catalano, Gennaro et Halevi dans [40].

9.2 Autres algorithmes distribués

9.2.1 Calcul distribué d'une multiplication

Nous rappelons brièvement le protocole de Ben-Or, Goldwasser, et Wigderson [13, 27] qui permet de calculer un partage de $c = ab(= (c_1 + \ldots + c_n))$, étant donnés deux partages $a = (a_1 + \ldots + a_n)$ et $b = (b_1 + \ldots + b_n)$. On notera ce protocole BGW dans cette thèse. Soit $P > (n2^{\log N})^2 > N$ un nombre premier. Le protocole est décrit dans la figure 9.3.

On peut prouver le théorème suivant qui montre qu'aucune coalition de $\lfloor \frac{n-1}{2} \rfloor$ serveurs apprend plus d'information sur les parts secrètes des autres serveurs. Ce résultat est au sens de la théorie de l'information et aucune hypothèse calculatoire n'est utilisée.

Théorème 29. Étant donné N, toute coalition de $\lfloor \frac{n-1}{2} \rfloor$ serveurs peut simuler la vue du protocole. Par conséquent, ce protocole est $\lfloor \frac{n-1}{2} \rfloor$ sûr.

9.2.2 Algorithme de calcul partagé d'un inverse modulo un secret partagé

Nous rappelons brièvement le protocole présenté par Catalano *et al.* [40] pour inverser une valeur publique e modulo une valeur partagée φ. L'astuce de ce schéma vient de l'observation que $\mathrm{pgcd}(e, \varphi) = \mathrm{pgcd}(e, \varphi + Re)$ où R est un grand entier utilisé pour masquer la valeur secrète et partagée $\varphi = \varphi_1 + \ldots + \varphi_n$.

Si on s'arrête au calcul de c, chaque serveur peut calculer $\mathrm{pgcd}(e, \varphi)$. Ce protocole permet aussi une fois le partage de $\varphi(N)$ de générer un partage de la clé secrète RSA d. On notera ce protocole GCD dans cette thèse.

77. Le Flexible RSA-Problem peut s'exprimer de la manière suivante : étant donné $y \in \mathbb{Z}_N^*$, trouver $x \in \mathbb{Z}_N^*$ et $e > 1$ tels que $y = x^e \mod N$. Il est clair que ce problème est plus facile que le problème RSA. Cependant, on ne sait pas si le problème RSA et le problème Flexible-RSA sont équivalents. L'hypothèse de sécurité faite lorsque l'on utilise ce problème est donc plus forte que celle que l'on fait quand on prouve la sécurité d'un système relativement au problème RSA. Cette hypothèse est appelée, Strong-RSA Assumption.

1. Soit $t = \lfloor \frac{n-1}{2} \rfloor$. Pour tout $i = 1, \ldots, n$, serveur i tire au hasard des polynômes de degré t, $f_i, g_i \in \mathbb{Z}_P[X]$ satisfaisant $f_i(0) = p_i$ et $g_i(0) = q_i$. En d'autres termes, les coefficients constants de f_i et de g_i sont fixés à p_i et q_i et tous les autres coefficients sont choisis au hasard dans $\mathbb{Z}_P$. De manière identique, serveur i choisit un polynôme de degré $2t$ $h_i \in \mathbb{Z}_P[X]$ tel que $h_i(0) = 0$.

2. Pour tout $i = 1, \ldots, n$, serveur i calcule les $3t$ valeurs :

$$\forall j = 1, \ldots, n : \; p_{i,j} = f_i(j) \; ; \; q_{i,j} = g_i(j) \; ; \; h_{i,j} = h_i(j)$$

   Serveur i envoie alors de manière secrète le triplet $(p_{i,j}, q_{i,j}, h_{i,j})$ aux serveurs j pour tout $j \neq i$. On remarque que les $p_{i,j}$ pour $j = 1, \ldots, n$ sont les parts d'un partage de secret *à la Shamir* de p_i, et de même pour q_i.

3. A ce point, chaque serveur i a toutes les parts $(p_{i,j}, q_{i,j}, h_{i,j})$ pour $j = 1, \ldots, n$. Serveur i calcule :

$$N_i = \left(\sum_{j=1}^{n} p_{i,j} \right) \cdots \left(\sum_{j=1}^{n} q_{i,j} \right) + \sum_{j=1}^{n} h_{i,j} \bmod P$$

   Serveur i broadcaste N_i à tous les autres serveurs.

4. A ce point, chaque serveur a tous les N_i pour $i = 1, \ldots, n$. Soit $\alpha(X)$ le polynôme :

$$\alpha(X) = \left(\sum_{j=1}^{n} f_j(X) \right) \cdots \left(\sum_{j=1}^{n} g_j(X) \right) + \sum_{j=1}^{n} h_j(X) \bmod P$$

   On observe que $\alpha(i) = N_i$ et par définition de f_i, g_i, h_i, nous avons $\alpha(0) = N$. De plus, $\alpha(X)$ est un polynôme de degré $2t$. On remarque que t est défini tel que $n \geq 2t + 1$. Par conséquent, comme tous les serveurs ont au moins $2t + 1$ points de $\alpha(X)$, ils peuvent l'interpoler et découvrir tous ses coefficients. Enfin, chaque serveur évalue $\alpha(0)$ et obtient $N \bmod P$. Comme $N < P$, les serveurs apprennent tous la valeur correcte de N.

FIG. 9.3: *Algorithme BGW pour calculer le produit de deux quantités partagées*

9.2.3 Algorithme du test de Fermat de biprimalité partagé

Nous rappelons brièvement le protocole présenté par Boneh et Franklin pour tester de manière partagée la biprimalité d'un nombre, c'est-à-dire, si étant donné un module N, est-il de la forme $N = pq$?

Ce test effectue les calculs modulo N qui est une quantité publique. En effet, il est très difficile de calculer modulo une quantité secrète et partagée. Par exemple, on ne sait même pas calculer de manière partagée et efficace $b \bmod p$ si b et p sont partagés. Ainsi, Boneh et Franklin effectuent un test modulo N.

De plus, d'après les résultats de Pomerance [152, 153] sur la rareté des pseudopremiers dès qu'un

1. Le serveur i choisit un entier au hasard $r_i \in_R [0..2^{L(N)+k'}]$, où k' est un paramètre de sécurité, calcule $c_i = \varphi_i + er_i$ et envoie c_i à tous les autres serveurs.
2. Chaque serveur peut alors calculer $c = \sum_i c_i = \varphi + eR$ avec $R = \sum_i r_i$. La valeur c peut être publiquement connue, R étant tenu secret.
3. Tous les serveurs peuvent alors calculer $\mathrm{pgcd}(e, c)$ et les coefficients de Bezout u et v tels que $eu + cv = \mathrm{pgcd}(e, \varphi)$. Si e et φ sont premiers entre eux, on a alors $eu + cv = 1$. Dans ce cas, il est facile de voir que si on remplace c par $\varphi + eR$, nous obtenons $e(u + Rv) + \varphi v = 1$. Ainsi, $u + Rv$ est l'inverse de e modulo φ. Notons le d.
4. Chaque serveur définit sa part de l'inverse d, $d_i = vr_i$, et le premier serveur, $d_1 = u + vr_1$.

FIG. 9.4: *Algorithme GCD pour calculer le PGCD ou l'inverse partagé d'un entier connu et d'un entier partagé*

crible est effectué, on peut estimer que dès qu'un module passe ce test, il est bien formé. Sinon, on serait obligé d'itérer ce test plusieurs fois car les nombres pseudo-premiers pour le test de Fermat représentent la moitié de l'espace.

Le test de biprimalité de Boneh-Franlin évite aussi les nombres de Carmichael en utilisant une variante du test de primalité de Solovay-Strassen modulo N. On peut remarquer que Boneh et Franklin n'ont pas tenté de partager modulo N l'algorithme de Miller-Rabin car ce dernier calcule des racines N-ième non triviales de l'unité et ceci permet de factoriser N. On voit le test de Solovay-Strassen dans le calcul de v. En effet, on a vu que l'on pouvait tester la résiduosité d'un élément g modulo N en calculant $g^{(p-1)(q-1)/4} \bmod N$ si $(g|N) = 1$. C'est ce qui est fait dans les étapes 2 et 3. Cette astuce peut être utilisée pour partager le cryptosystème de Goldwasser-Micali. L'étape 4 est un peu technique et est utile dans la preuve du théorème 30 pour obtenir la borne $1/2$.

On peut montrer les résultats suivants :

Théorème 30. Soit $N = pq$ un entier tel que $p \equiv q \equiv 3 \bmod 4$. Si N est le produit de deux nombres premiers, alors "succès" est déclaré dans toutes les invocations du protocole. Sinon, les serveurs déclarent que N n'est pas le produit de 2 nombres premiers avec probabilité au moins $\frac{1}{2}$ (parmi tous les choix des randoms g et h).

En ce qui concerne la sécurité de ce protocole dans le modèle *honnête-mais-curieux*.

Théorème 31. Supposons que p et q sont des nombres premiers distincts tels que $p \equiv q \equiv 3 \bmod 4$. Alors toute coalition de $n - 1$ serveurs peut simuler leur vue du protocole de test de biprimalité. Par conséquent, ce protocole est $n - 1$ sûr.

1. Les serveurs génèrent un random $g \in \mathbb{Z}_N^*$. La valeur de g est connue des n serveurs.
2. Le serveur 1 calcule le symbole de Jacobi de g modulo N. Si $(g|N) \neq 1$, le protocole retourne à l'étape 1 et un nouveau g est choisi.
3. Sinon, le serveur 1 calcule $v_1 = g^{(N-p_1-q_1+1)/4} \bmod N$. Tous les autres serveurs calculent $v_i = g^{(p_i+q_i)/4} \bmod N$. Les serveurs calculent ensuite $v = \prod_{i=1}^{n} v_i \bmod N$ en utilisant par exemple le protocole de Benaloh [15] et décrit dans la version Journal of Cryptology de [27]. Ils vérifient si

$$v = \prod_{i=1}^{n} v_i \stackrel{?}{=} \pm 1 \bmod N$$

   Si le test échoue, les serveurs déclarent que N n'est pas le produit de deux nombres premiers.
4. Les serveurs exécutent un test de Fermat dans le groupe $T_N = (\mathbb{Z}_N[X]/(X^2 + 1))^*/\mathbb{Z}_N^*$. Pour effectuer ce test dans T_N, les serveurs tirent un random $h \in T_N$. Serveur 1 calcule $u_1 = h^{N-p_1-q_1+1}$. Tous les autres serveurs calculent $u_i = h^{p_i+q_i}$. Les serveurs utilisent alors le même protocole que dans l'étape 3 pour calculer $u = \prod_{i=1}^{n} u_i$. Ils vérifient ensuite si

$$u = \prod_{i=1}^{n} u_i \stackrel{?}{=} 1$$

   Si le test échoue, N est rejeté. Sinon, ils déclarent "succès".

FIG. 9.5: *Algorithme de test de biprimalité partagé*

10

Chiffrement du même message sous plusieurs clés

Dans cette annexe, nous montrons qu'il faut être prudent quand on chiffre le même message sous plusieurs clés toutes différentes, en particulier lorsque l'on utilise le cryptosystème RSA sans fonction de padding. Il est évident qu'un chiffrement probabiliste permet de contrer ces attaques, mais seul un chiffrement sémantiquement sûr permet de contrer toutes les attaques possibles. Il faut ainsi utiliser RSA avec le padding OAEP par exemple pour atteindre ce niveau de sécurité.

10.1 Chiffrement RSA sous deux clés (e_1, N) et (e_2, N) avec $\mathrm{pgcd}(e_1, e_2) = 1$

Dans ce cas, on montre que l'on peut reconstruire le message m sans connaître aucune des deux clés secrètes.

Comme $\mathrm{pgcd}(e_1, e_2) = 1$, l'algorithme d'Euclide étendu permet de trouver deux entiers a et b tels que $ae_1 + be_2 = 1$. Considérons les chiffrés du même message sous les deux clés :

$$c_1 = m^{e_1} \bmod N \quad \text{et} \quad c_2 = m^{e_2} \bmod N \tag{10.1}$$

À partir de a et b, on peut calculer

$$c_1^a \times c_2^b \equiv (m^{e_1})^a (m^{e_2})^b \equiv m^{ae_1+be_2} = m \bmod N$$

Remarques: On peut aussi montrer que l'utilisateur 1 peut calculer la clé secrète de l'utilisateur 2 à partir de sa clé secrète. En effet, soient d_1 et d_2 les clés secrètes des utilisateurs 1 et 2 respectivement. On a alors

$$e_1 \times d_1 = 1 \bmod \varphi(N) \quad \text{et} \quad e_2 \times d_2 = 1 \bmod \varphi(N) \tag{10.2}$$

Ainsi, $e_1 d_1 - 1$ est un multiple de $\varphi(N)$. L'utilisateur 1 peut donc obtenir la clé secrète d_2 en calculant ${e_2}^{-1} \bmod (e_1 d_1 - 1)$ avec l'algorithme d'Euclide étendu.

Enfin, on peut aussi appliquer l'algorithme de Miller [126] pour factoriser un module RSA N lorsque l'on connaît un multiple de $\varphi(N)$.

10.2 Chiffrement du même message sous plusieurs clés RSA différentes

Un émetteur souhaite envoyer le même message à trois utilisateurs différents possédant les clés publiques (e_1, N_1), (e_2, N_2), et (e_3, N_3) respectivement avec $e_1 = e_2 = e_3 = 3$. Soit les chiffrés $c_1 = m^{e_1} \bmod N_1$, $c_2 = m^{e_2} \bmod N_2$, et $c_3 = m^{e_3} \bmod N_3$.

Premier cas. Les modules N_i sont tous relativement premiers entre eux. Dans ce cas, on peut utiliser le théorème des restes chinois pour trouver l'unique entier $z \in \{0, \ldots, N_1 N_2 N_3 - 1\}$ qui est congru à $c_i \bmod N_i$ pour $i = 1, 2$, et 3. Cependant, comme $m < N_1, N_2, N_3$, alors $m^3 \leq N_1 N_2 N_3 - 1$ et nous savons alors que m^3 est congru à $c_i \bmod N_i$ pour $i = 1, 2$, et 3.

Ceci signifie que l'on doit avoir $z = m^3$ (dans $\mathbb{Z}$). Ainsi, après avoir trouvé z, on calcule une racine cubique dans les entiers, ce qui permet d'obtenir le message m.

Deuxième cas. Dans le cas où les modules ne sont pas tous relativement premiers entre eux, on peut calculer $\mathrm{pgcd}(N_1, N_2)$, ou $\mathrm{pgcd}(N_2, N_3)$ ou $\mathrm{pgcd}(N_1, N_3)$, ce qui permet de factoriser deux tels modules. Enfin, on termine en calculant la clé secrète de l'un de ces deux utilisateurs.

10.3 Chiffrement multicast

Remarques. La différence entre le multicast et le broadcast est la suivante : dans le *multicast*, la population en réception est un groupe d'utilisateurs finis pris dans l'ensemble de tous les utilisateurs potentiels, dans le cas du *broadcast* tous les utilisateurs sont visés.

Soit $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ un système de chiffrement sémantiquement sûr. Considérons le cryptosystème $(\mathcal{K}', \mathcal{E}', \mathcal{D}')$ suivant :

Algorithme de génération des clés. L'algorithme de génération de clés $\mathcal{K}'$ sur l'entrée 1^k et n appelle n fois l'algorithme $\mathcal{K}$ sur 1^k pour obtenir $(\overline{\mathsf{pk}}, \overline{\mathsf{sk}}) = (\langle \mathsf{pk}_1, \ldots, \mathsf{pk}_n \rangle, \langle \mathsf{sk}_1, \ldots, \mathsf{sk}_n \rangle)$ où $(\mathsf{pk}_i, \mathsf{sk}_i)$ est la sortie de $\mathcal{K}(1^k)$.

Chiffrement. Pour chiffrer un message m avec la clé publique $\overline{\mathsf{pk}}$, on utilise le chiffrement $\mathcal{E}$ pour chiffrer le message sous chaque clé pk_i, soit

$$\mathcal{E}'_{\overline{\mathsf{pk}}}(m; \langle r_1, \ldots, r_n \rangle) = \langle \mathcal{E}_{\mathsf{pk}_1}(m; r_1), \ldots \mathcal{E}_{\mathsf{pk}_n}(m; r_n) \rangle$$

Déchiffrement. Pour déchiffrer le chiffré $\overline{c} = \langle c_1, \ldots, c_n \rangle$, on prend la première coordonnée et on déchiffre avec sk_1, soit :

$$\mathcal{D}'_{\overline{\mathsf{sk}}}(\overline{c}) = \mathcal{D}_{\mathsf{sk}_1}(c_1)$$

Théorème 32. Soit $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ un système de chiffrement sémantiquement sûr et n un paramètre polynomial. Alors le cryptosystème $(\mathcal{K}', \mathcal{E}', \mathcal{D}')$ est aussi sémantiquement sûr.

Preuve: Supposons que le cryptosystème $(\mathcal{K}', \mathcal{E}', \mathcal{D}')$ ne soit pas sémantiquement sûr. Il existe donc un adversaire $\mathcal{A}$ contre ce système, ce qui signifie qu'il existe $\gamma > 0$, tel que pour infiniment beaucoup de k,

$$\begin{aligned} \Big| \Pr_{\overline{\mathsf{pk}}, m_0, m_1, \omega} \Big[\mathcal{A}(1^k, \overline{\mathsf{pk}}, \mathcal{E}'_{\overline{\mathsf{pk}}}(m_b, \overline{r}), m_0, m_1) = 1 \Big] \\ - \Pr_{\overline{\mathsf{pk}}, m_0, m_1, \omega} \Big[\mathcal{A}(1^k, \overline{\mathsf{pk}}, \mathcal{E}'_{\overline{\mathsf{pk}}}(m_{1-b}, \overline{r}), m_0, m_1) = 1 \Big] \Big| > \frac{1}{k^\gamma} \end{aligned} \tag{10.3}$$

où la probabilité est prise sur le choix de $\overline{\mathsf{pk}}$ par l'algorithme $\mathcal{K}'$ sur l'entrée 1^k, le choix de m_0, m_1 par l'algorithme A_1 sur l'entrée 1^k, $\overline{\mathsf{pk}}$, et le ruban d'aléa ω pour le choix de $\overline{r}$ et sur ω pour l'algorithme A_2.

Dans la suite de la preuve, on fixe une valeur de k pour laquelle l'inégalité 10.3 est vérifiée. Considérons la probabilité de l'adversaire $\mathcal{A}$ de gagner le jeu de la sécurité sémantique sur $D_0, D_1, \ldots, D_n$ distributions de probabilité différentes. Dans le i-ième jeu, on donne à $\mathcal{A}$ le vecteur $\overline{c}$ de n chiffrés, où les i premières valeurs sont des chiffrés de m_0 et les $n-i$ dernières sont des chiffrés de m_1 en utilisant toujours l'algorithme $\mathcal{E}$. Sur chaque distribution, on calcule la probabilité de succès que l'algorithme $\mathcal{A}$ parvienne à deviner le bit b. On note p_i :

$$p_i \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l}(\overline{\mathsf{pk}}, \overline{\mathsf{sk}}) \leftarrow \mathcal{K}'(1^k)\\ m_0, m_1 \leftarrow A_1(1^k, \overline{\mathsf{pk}})\\ c_1 \leftarrow \mathcal{E}_{\mathsf{pk}_1}(m_0, r_1), \ldots, c_i \leftarrow \mathcal{E}_{\mathsf{pk}_i}(m_0, r_i)\\ c_{i+1} \leftarrow \mathcal{E}_{\mathsf{pk}_{i+1}}(m_1, r_{i+1}), \ldots, c_n \leftarrow \mathcal{E}_{\mathsf{pk}_n}(m_1, r_n)\\ A_2(1^k, \overline{\mathsf{pk}}, \overline{c}, m_0, m_1) = 1\end{array}\right] \tag{10.4}$$

L'avantage que l'algorithme $\mathcal{A}$ parvienne à deviner le bit b au i-ème jeu est donc $\leq |p_i - p_{i-1}|$.

On remarque que la distribution d'entrée D_n correspond au jeu réel de l'expérience dans laquelle on a chiffré m_1 alors que D_0 correspond au jeu réel de l'expérience où on a chiffré m_0. Ainsi, on obtient :

$$p_0 - p_n > 1/k^\gamma$$

Montrons maintenant que le système de chiffrement $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ n'est pas sémantiquement sûr en construisant un attaquant $\mathcal{B}$ avec probabilité de succès $1/(nk^\gamma)$.

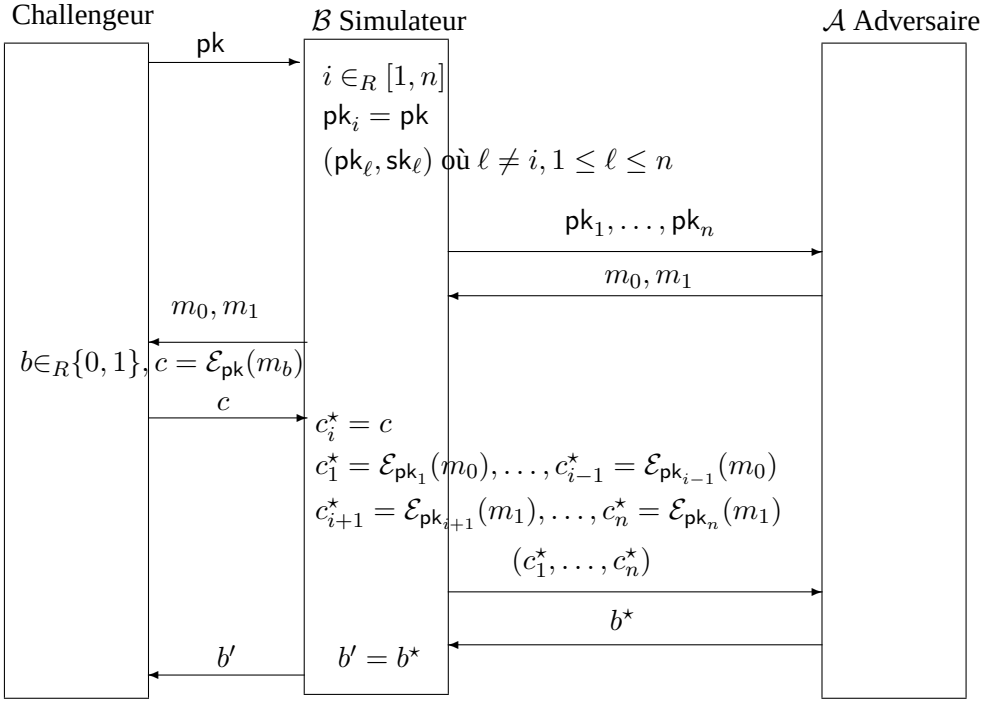


FIG. 10.1: *Preuve de sécurité du chiffrement multi-utilisateurs.*

Considérons ce qui se passe lorsque la clé secrète inconnue est la i-ième. Si le chiffré c est issu du chiffrement de m_0, alors la distribution que voit $\mathcal{A}$ est D_i et si c est issu du chiffrement de m_1, alors la

distribution est D_{i-1}. Par conséquent, pour tout i, on a :

$$\Pr\left[\mathcal{A} \text{ retourne } 1 | c = \mathcal{E}_{\mathsf{pk}}(m_1, r), i\right] = p_{i-1} \quad \text{et} \quad \Pr\left[\mathcal{A} \text{ retourne } 1 | c = \mathcal{E}_{\mathsf{pk}}(m_0, r), i\right] = p_i$$

ainsi l'avantage total de $\mathcal{A}$ est

$$\begin{aligned}
&\Pr\left[\mathcal{A} \text{ retourne } 1 | c = \mathcal{E}_{\mathsf{pk}}(m_1, r)\right] - \Pr\left[\mathcal{A} \text{ retourne } 1 | c = \mathcal{E}_{\mathsf{pk}}(m_0, r)\right] \\
&= \left(\sum_{i=1}^{n} \Pr[i] \,.\, \Pr\left[\mathcal{A} \text{ retourne } 1 | c = \mathcal{E}_{\mathsf{pk}}(m_1, r), i\right]\right) - \left(\sum_{i=1}^{n} \Pr[i] \,.\, \Pr\left[\mathcal{A} \text{ retourne } 1 | c = \mathcal{E}_{\mathsf{pk}}(m_0, r), i\right]\right) \\
&= \frac{1}{n} \sum_{i=1}^{n} (p_{i-1} - p_i) = \frac{1}{n}(p_0 - p_n) > \frac{1}{nk^{\gamma}}
\end{aligned}$$

Cet avantage est donc non-négligeable. Ce qui contredit l'hypothèse selon laquelle le schéma d'origine $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ est sémantiquement sûr. □

Remarques : La technique de preuve utilisée ici est appelée la méthode des hybrides. En effet, les différentes distributions $D_1, \ldots, D_{n-1}$ sont des vecteur mélangeant des chiffrés de m_0 et de m_1.

11

Vérification en batch

Bellare *et al.* dans [7] ont décrit des algorithmes pour exécuter des vérifications rapides en batch pour les exponentiations modulaires et les signatures numériques. Dans cet article, ils présentent des techniques pour tester si beaucoup d'instances $(x_i, y_i)_{i=1}^n$ satisfont les équations $g^{x_i} = y_i \bmod p$. La méthode naïve nécessite n exponentiations. Cependant, si nous utilisons des tests en batch probabilistes, la séquence d'exponentiations modulaires peut être calculée plus rapidement.

Ils ont aussi décrit un algorithme efficace pour calculer $y = \prod_{i=1}^n a_i^{b_i}$ où le coût de calcul est de $k + nk/2$ multiplications modulaires si nous notons k, la taille du plus grand des éléments b_i en bit : $(b_i = b_i[k] \dots b_i[1])$. Ce nombre est strictement inférieur aux n exponentiations puis $n-1$ multiplications nécessaires avec la méthode naïve. Le coût d'une seule exponentiation a^b peut être estimé à $3k/2$ multiplications où k représente la longueur en bit de b. Cet algorithme est appelé FastMult.

`Algorithm` FastMult$((a_1, b_1), \dots, (a_n, b_n))$
    $a := 1$;
    `for` $j = k$ `downto` 1 `do`
        `for` $i = 1$ `to` n `do if` $b_i[j] = 1$ `then` $a := a.a_i$;
        $a := a^2$;
`return` a

Cet algorithme utilise k multiplications dans la boucle externe et $nk/2$ multiplications en moyenne dans la boucle interne. Ainsi, pour calculer y nous avons au total $k + nk/2$ multiplications.

11.1 Random Linear Combination Test

Finalement, Bellare *et al.* présentent aussi une vérification par batch de la forme suivante : étant donné un ensemble de points, déterminer s'il existe un polynôme d'un degré donné qui passe par tous ces points. Plus formellement, soit $S = (\alpha_1, \alpha_2, \dots, \alpha_m)$ un ensemble de points. On définit la relation $\mathrm{DEG}_{\mathcal{F},t,(\beta_1,\beta_2,\dots,\beta_m)}(S) = 1$ si et seulement s'il existe un polynôme f de degré au plus t, et tel que pour tout $i \in \{1, \dots, m\}$, $f(\beta_i) = \alpha_i$, tous les calculs étant exécutés dans un corps fini $\mathcal{F}$. Soit l'instance de batch de ce problème $S_1, \dots, S_n$, où $S_i = (\alpha_{i,1}, \dots, \alpha_{i,m})$. L'instance de batch est correcte si $\mathrm{DEG}_{\mathcal{F},t,(\beta_1,\dots,\beta_m)}(S_i) = 1$ pout tout $i = 1, \dots, n$; et incorrecte sinon. Ce test est appelé RANDOM LINEAR COMBINATION TEST.

Cet algorithme prend en entrée n ensembles $S_1, \dots, S_n$ où $S_i = (\alpha_{i,1}, \dots, \alpha_{i,m})$; $\beta_1, \dots, \beta_m$, et le paramètre de sécurité k, et une valeur t et vérifie, si pour tout $i \in \{1, \dots, n\}$, il existe un polynôme $f_i(x)$ tel que $deg(f_i) \leq t$ et $f_i(\beta_1) = \alpha_{i,1}, \dots, f_i(\beta_m) = \alpha_{i,m}$.

L'algorithme fonctionne de la manière suivante :

1. Choisit $r \in_R \mathcal{F}$
2. Calcule $\gamma_i = r^n \alpha_{n,i} + \ldots + r\alpha_{1,i}$. Ceci peut être calculé de manière efficace avec l'algorithme de Horner.
3. Si $\mathrm{DEG}_{\mathcal{F},t,(\beta_1,\ldots,\beta_m)}(\gamma_1, \ldots, \gamma_m) = 1$, alors retourne "correct", "incorrect" sinon.

NOTATION Si $f_i(x) = a_m x^m + \ldots + a_0$, où $a_m \neq 0$, on dénote par $f_i(x)|^{t+1}$ le polynôme $a_m x^m + \ldots + a_{t+1} x^{t+1}$. Par conséquent, pour $m \leq t$, $f_i(x)|^{t+1}$ doit être égal à 0.

Le polynôme $F(x) = \sum_{i=1}^{n} r^i f_i(x)$ est de degré au plus t, et, par conséquent, il vérifie que $\sum_{i=1}^{n} r^i f_i(x)|^{t+1} = 0$. Ceci est une équation de degré n en l'inconnue r et donc a au plus n racines. Par conséquent, si l'instance est incorrecte, l'algorithme n'accepte que si r est l'une des racines de l'équation. Cet algorithme échoue avec probabilité au plus $\frac{n}{|\mathcal{F}|}$, qui est une quantité négligeable. Le temps d'exécution de cet algorithme est $O(nm)$ alors que la méthode naïve qui consiste à calculer le polynôme d'interpolation avec $t+1$ points et à vérifier chacun d'entre eux, demande $O(m^2 n)$ multiplications.

Bibliographie

[1] O. Baudron, P.A. Fouque, D. Pointcheval, G. Poupard, and J. Stern. Practical Multi-Candidate Election System. In *PODC '01*. ACM, 2001.

[2] O. Baudron, D. Pointcheval, and J. Stern. Extended Notions of Security for Multicast Public Key Cryptosystems. In *Proc. of the 27th ICALP*, LNCS 1853, pages 499–511. Springer-Verlag, Berlin, 2000.

[3] D. Beaver and N. So. Global, Unpredictable Bit Generation Without Broadcast. In *Eurocrypt '93*, LNCS 765, pages 424–434. Springer-Verlag, 1994.

[4] M. Bellare, A. Boldyreva, and S. Micali. Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements. In *Eurocrypt '2000*, LNCS 1807, pages 259–274. Springer-Verlag, Berlin, 2000.

[5] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *Crypto '98*, LNCS 1462, pages 26–45. Springer-Verlag, 1998.

[6] M. Bellare, M. Fischlin, S. Goldwasser, and S. Micali. Identification Protocols Secure against Reset Attacks. In *Eurocrypt '2001*, LNCS 2045, pages 495–511. Springer-Verlag, Berlin, 2001.

[7] M. Bellare, J. A. Garay, and T. Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *Eurocrypt '98*, LNCS 1403, pages 236–250. Springer-Verlag, 1998. Available at `http://www-cse.ucsd.edu/users/mihir/`.

[8] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The Power of RSA Inversion Oracles and the Security of Chaum's RSA-Based Blind Signature Scheme. In *Financial Crypto '01*, LNCS. Springer-Verlag, Berlin, 2001.

[9] M. Bellare and P. Rogaway. Random Oracles are Practical: a paradigm for designing efficient protocols. In *Proc. of the 1st CCCS*, pages 62–73. ACM press, 1993.

[10] M. Bellare and P. Rogaway. Optimal Asymetric Encryption - How to Encrypt with RSA. In *Eurocrypt '94*, LNCS 950, pages 92–111. Springer-Verlag, 1994.

[11] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures – How to Sign with RSA and Rabin. In *Eurocrypt '96*, LNCS 1070, pages 399–416. Springer-Verlag, 1996.

[12] M. Bellare and A. Sahai. Non-malleability encryption: Equivalence between two notions, and an indistinguishability-based characterization. In *Crypto '99*, LNCS 1666, pages 519–536. Springer-Verlag, 1999.

[13] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proc. of the 20th STOC*, pages 1–10. ACM Press, 1988.

[14] M. Ben-Or and T. Rabin. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. of the 21th STOC*, pages 73–85. ACM Press, 1989.

[15] J. Benaloh. Secret sharing homomorphisms : keeping shares of a secret secret. In *Crypto '86*, LNCS 263, pages 251–260. Springer-Verlag, 1987.

[16] J. Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, 1987.

[17] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Proc. 26th ACM Symposium on the Theory of Computing (STOC)*, pages 544–553. ACM, 1994.

[18] S. Blackburn, S. Blake-Wilson, S. Galbraith, and M. Burmester. Shared Generation of Shared RSA Keys. Technical report, University of Waterloo, Canada, February 1998. CORR-98-19.

[19] G. Blakley. Safeguarding Cryptographic Keys. In *proc. of AFIPS National Computer Conference*, pages 313–317, 1979.

[20] D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In *Crypto '98*, LNCS 1462, pages 1–12. Springer-Verlag, 1998.

[21] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal of Computing*, 15:364–383, 1986.

[22] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and its Applications. In *Proc. of the 20th STOC*, pages 103–112. ACM Press, New York, 1988.

[23] M. Blum, P. Feldman, and S. Micali. Proving Security Against Chosen Ciphertext Attacks. In *Crypto '88*, LNCS 403, pages 256–268. Springer-Verlag, 1989.

[24] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-Interactive Zero-Knowledge. *SIAM journal of computing*, 20(4):1084–1118, 1991.

[25] D. Boneh. The decision diffie-hellman problem. In *Proc. of the Third Algorithmic Number Theory Symposium*, LNCS 1423, pages 48–63. Springer-Verlag, 1998.

[26] D. Boneh. Twenty years of attacks on the RSA cryptosystem. In *Notices of the American Mathematical Society (AMS)*, pages 203–213, 1999.

[27] D. Boneh and M. Franklin. Efficient Generation of Shared RSA Keys. In *Crypto '97*, LNCS 1294, pages 425–439. Springer-Verlag, 1997.

[28] D. Boneh, M. Malkin, and T. Wu. Experimenting with Shared Generation of RSA keys. In *Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS)*, pages 43–56, 1999.

[29] M. Burmester and Y. Desmedt. All Language in NP Have Divertible Zero-Knowledge Proofs and Arguments under Cryptographic Assumptions. In *Eurocrypt '90*, LNCS 473, pages 1–10. Springer-Verlag, 1991.

[30] J. Camenisch and I. Damgård. Verifiable Encryption and Applications to Group Signatures and Signature Sharing. Available at `http://philby.ucsd.edu/cryptolib/1999/99-08.html`, march 1999.

[31] R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai, and T. Malkin. On Adaptive vs. Non-adaptive Security of Multiparty Protocols. In *Eurocrypt '01*, LNCS 2045, pages 262–279. Springer-Verlag, 2001.

[32] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-Resilient Functions and All-Or-Nothing Transforms. In *Eurocrypt '00*, LNCS 1807, pages 453–469. Springer-Verlag, 2000.

[33] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable Encryption. In *Crypto '97*, LNCS 1294, pages 90–104. Springer-Verlag, 1997.

[34] R. Canetti and R. Gennaro. Incoercible Multiparty Computation. In *Proc. 37th IEEE Symposium on the Foundations of Computer Science (FOCS)*. IEEE, 1996.

[35] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive Security for Threshold Cryptosystems. In *Crypt '99*, LNCS 1666, pages 98–115. Springer-Verlag, 1999.

[36] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology Revisited. In *Proc. of the 30th STOC*, pages 209–218. ACM Press, 1998.

[37] R. Canetti and S. Goldwasser. An Efficient Threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack. In *Eurorypt '99*, LNCS 1592, pages 90–106. Springer-Verlag, 1999.

[38] R. Canetti, S. Halevi, and A. Herzberg. Maintaining Authenticated Communication in the Presence of Break-ins. In *Proc. of the 16th ACM Symp. on Principles of Distributed Computation (PODC 97)*, pages 455–469. ACM, 1997.

[39] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Crypto '94*, LNCS 839, pages 425–438. Springer-Verlag, 1994.

[40] D. Catalano, R. Gennaro, and S. Halevi. Computing Inverses over a Shared Secret Modulus. In *Eurocrypt '00*, LNCS 1807, pages 190–207. Springer-Verlag, 2000.

[41] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[42] D. Chaum. Blind Signatures for Untraceable Payments. In *Crypto '82*, pages 199–203. Plenum, NY, 1983.

[43] D. Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols. In *Proc. of the 20th STOC*, pages 11–19. ACM Press, 1988.

[44] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In *Crypto '92*, LNCS 740, pages 89–105. Springer-Verlag, 1992.

[45] L. Chen. *Witness Hiding Proofs and Applications*. PhD thesis, Aarhus University, august 1994.

[46] L. Chen, I. B. Damgård, and T. P. Pedersen. Parallel Divertibility of Proofs of Knowledge. In *Eurocrypt '94*, LNCS 950, pages 140–155. Springer-Verlag, 1995.

[47] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneous Broadcast. In *Proc. of the 26th FOCS*, pages 335–344. IEEE, 1985.

[48] C. Cocks. Split Knowledge Generation of RSA Parameters. In *Cryptography and Coding : 6th IMA Conference*, LNCS 1355, pages 89–95. Springer-Verlag, 1997.

[49] C. Cocks. Split Generation of RSA Parameters with Multiple Participants. Technical report, CESG, 1998. Available at `http://www.cesg.gov.uk`.

[50] J. Cohen and M. Fisher. A robust and verifiable cryptographically secure election scheme. In *Symposium on Foundations of Computer Science*. IEEE, 1985.

[51] J.S. Coron. On The Exact Security of Full Domain Hash. In *Crypto '00*, pages 229–235. Springer-Verlag, 2000.

[52] J.S. Coron, D. Naccache, and J.P. Stern. On the Security of RSA Padding. In *Crypto '99*, pages 1–18. Springer-Verlag, 1999.

[53] R. Cramer, Y. Frankel, B. Schoenmakers, and M. Yung. Multi-Authority Secret-Ballot Elections with Linear Work. In *Eurocrypt '96*, LNCS 1070, pages 72–83. Springer-Verlag, 1996.

[54] R. Cramer, R. Gennaro, and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *Eurocrypt '97*, LNCS 1233, pages 113–118. Springer-Verlag, 1997.

[55] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptative Chosen Ciphertext Attack. In *Crypto '98*, LNCS 1462, pages 13–25. Springer-Verlag, 1998.

[56] I. Damgård and M. Jurik. Efficient Protocols based on Probabilistic Encryption using Composite Degree Residue Classes. In *PKC '01*, LNCS 1992, pages 119–136. Springer-Verlag, 2001.

[57] I. Damgård and M. Koprowski. Practical Threshold RSA Signatures Without a Trusted Dealer. In *Eurocrypt '01*, LNCS 2045, pages 152–165. Springer-Verlag, 2001.

[58] I. Damgård and J. N. Nielsen. Improved Non-Commiting Encryption Schemes Based on a General Complexity Assumption. In *Crypto '00*, LNCS 1880, pages 432–450. Springer-Verlag, 2000.

[59] B. den Boer and A. Bosselaers. Collisions for the Compression Function of MD5. In *Eurocrypt '93*, LNCS 765, pages 293–304. Springer-Verlag, 1994.

[60] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *Crypto '89*, LNCS 435, pages 307–315. Springer-Verlag, 1989.

[61] Y. Desmedt and Y. Frankel. Shared Generation of Authenticators and Signature. In *Crypto '91*, LNCS 576, pages 457–469. Springer-Verlag, 1991.

[62] W. Diffie and M. E. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, volume IT–22, no. 6, pages 644–654, november 1976.

[63] H. Dobbertin. The Status of MD5 After a Recent Attack. *CryptoBytes*, 2(2):1–6, summer 1996.

[64] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In *Proc. of the 23rd STOC*. ACM Press, New York, 1991.

[65] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[66] T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *IEEE Transactions on Information Theory*, volume IT–31, no. 4, pages 469–472, july 1985.

[67] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology*, 1:77–95, 1988.

[68] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *Proc. of the 28th FOCS*, pages 427–437. IEEE, 1987.

[69] A. Fiat and A. Shamir. How to Prove Yourself: practical solutions of identification and signature problems. In *Crypto '86*, LNCS 263, pages 186–194. Springer-Verlag, 1987.

[70] P. A. Fouque. Les technologies de l'écrit électronique. In 10-*ième annales de l'Association Rencontre Notariat Université - Vers l'authenticité électronique*. Editions quotidienne des journaux judiciaires associés, Petites affiches, 2000.

[71] P. A. Fouque, G. Poupard, and J. Stern. Recovering Keys in Open Networks. In *proc. of IEEE Information Theory Workshop*, 1999.

[72] P. A. Fouque, G. Poupard, and J. Stern. Sharing Decryption in the Context of Voting or Lotteries. In *Financial Crypto '00*, LNCS. Springer-Verlag, 2000.

[73] P. A. Fouque, J. Stern, and G. J. Wackers. Application to the rationals of the homomorphic properties of the Paillier cryptosystem. En soumission., 2001.

[74] P.A. Fouque and D. Pointcheval. Threshold Cryptosystems Secure against Chosen-Ciphertext Attacks. In *Asiacrypt '01*, LNCS. Springer-Verlag, 2001.

[75] P.A. Fouque and J. Stern. Fully Distributed Threshold RSA under Standard Assumptions. In *Asiacrypt '01*, LNCS. Springer-Verlag, 2001. Available on the eprint server `http://eprint.iacr.org`.

[76] P.A. Fouque and J. Stern. One Round Threshold Discrete-Log Key Generation without any Private Channel. In *PKC '01*, LNCS 1992, pages 300–316. Springer-Verlag, 2001.

[77] Y. Frankel, P. Gemmel, Ph. MacKenzie, and M. Yung. Optimal-Resilience Proactive Public-Key Cryptosystems. In *Proc. 38th FOCS*, pages 384–393. IEEE, 1997.

[78] Y. Frankel, P. Gemmel, Ph. MacKenzie, and M. Yung. Proactive RSA. In *Crypto '97*, LNCS 1294, pages 440–454. Springer-Verlag, 1997.

[79] Y. Frankel, P. Gemmell, and M. Yung. Witness Based Cryptographic Program Checking and Robust Function Sharing. In *Proc. 28th STOC*, pages 499–508, 1996.

[80] Y. Frankel, P. MacKenzie, and M. Yung. Robust Efficient Distributed RSA Key Generation. In *STOC '98*, pages 663–672, 1995.

[81] E. Fujisaki and T. Okamoto. How to Enhance the Security of Public-Key Encryption at Minimum Cost. In *PKC '99*, LNCS 1560, pages 53–68. Springer-Verlag, Berlin, 1999.

[82] E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Crypto '99*, LNCS 1666, pages 537–554. Springer-Verlag, Berlin, 1999.

[83] E. Fujisaki and T. Okamoto. How to Enhance the Security of Public-Key Encryption at Minimum Cost. *IEICE Transaction of Fundamentals of Electronic Communications and Computer Science*, E83-A(1):24–32, January 2000.

[84] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA–OAEP is Secure under the RSA Assumption. In *Crypto '2001*, LNCS 2139, pages 260–274. Springer-Verlag, Berlin, 2001.

[85] J. Furukawa and K. Sako. An Efficient Scheme for Proving a Shuffle. In *Crypto '01*, LNCS 2139, pages 368–387. Springer-Verlag, Berlin, 2001.

[86] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*. Freeman, New-York, 1979.

[87] R. Gennaro, S. Halevi, and T. Rabin. Secure Hash-and-Sign Signatures Without the Random Oracle. In *Eurocrypt '99*, LNCS 1592, pages 123–139. Springer-Verlag, 1999.

[88] R. Gennaro, S. Jarecki, H.Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Eurocrypt '96*, LNCS 1070, pages 354–371. Springer-Verlag, 1996.

[89] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *Crypto '96*, LNCS 1109, pages 157–172. Springer-Verlag, 1996.

[90] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Eurocrypt '99*, LNCS 1592, pages 295–310. Springer-Verlag, 1999.

[91] R. Gennaro, D. Micciancio, and T. Rabin. An Efficient Non-Interactive Statistical Zero-Knowledge Proof System for Quasi-Safe Prime Products. In *Proc. of the 5th CCCS*, pages 67–72. ACM press, 1998.

[92] N. Gilboa. Two Party RSA Key Generation. In *Crypto '99*, LNCS 1666, pages 116–129. Springer-Verlag, 1999.

[93] M. Girault and J. Stern. On the Length of Cryptographic Hash-Values used in Identification Schemes. In *Crypto '94*, LNCS 839, pages 202–215. Springer-Verlag, 1994.

[94] O. Goldreich. Foundations of Cryptography. 1989.

[95] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game. In *Proc. of the 19th STOC*, pages 218–229. ACM Press, 1987.

[96] O. Goldreich, S. Micali, and A. Wigderson. How to Prove All $\mathcal{NP}$ Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design. In *Crypto '86*, LNCS 263, pages 171–185. Springer-Verlag, 1987.

[97] D.M. Goldschlag and S.G. Stubblebine. Publicly Verifiable Lotterie : Applications of Delaying Functions. In *Financial Crypto '98*, LNCS 1465, pages 214–226. Springer-Verlag, 1998.

[98] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28, 1984.

[99] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. In *Proc. of the 17th STOC*, pages 291–304. ACM Press, 1985.

[100] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM journal of computing*, 18(1):186–208, february 1989.

[101] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptative Chosen-Message Attacks. *SIAM journal of computing*, 17(2):281–308, april 1988.

[102] L. C. Guillou and J. J. Quisquater. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In *Eurocrypt '88*, LNCS 330, pages 123–128. Springer-Verlag, 1988.

[103] V. Guruswani and M. Sudan. Improved decoding for Reed-Solomon and algebraic geometric codes. *IEEE Tran. on Info. Theory*, 45(6):1757–1767, 1999.

[104] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Public Key and Signature Systems. In *ACM CCS '97*. ACM, 1997.

[105] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing, or : how to cope with perpetual leakage. In *Crypto '95*, LNCS 963, pages 339–352. Springer-Verlag, 1995.

[106] M. Hirt and K. Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *Eurocrypt '00*, LNCS 1807, pages 539–556. Springer-Verlag, 2000.

[107] T. Itoh and K. Sakurai. On the Complexity of Constant Round ZKIP for Possession of Knowledge. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E76-A(1), 1993.

[108] T. Itoh, K. Sakurai, and H. Shizuya. Any Language in IP has a divertible ZKIP. In *Asiacrypt '91*, LNCS 739, pages 382–397. Springer-Verlag, 1993.

[109] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated Verifier Proofs and Their Applications. In *Eurocrypt '96*, LNCS 1070, pages 143–154. Springer-Verlag, Berlin, 1996.

[110] S. Jarecki and A. Lysyanskaya. Adaptively Secure Threshold Cryptography : Introducing Concurrency, Removing Erasures. In *Eurocrypt' 00*, LNCS 1807, pages 221–242. Springer-Verlag, 2000.

[111] M. Joye, J. J. Quisquater, and M. Yung. On the Power of Misbehaving Adversaries and Security Analysis of the Original EPOC. In *CT – RSA '2001*, LNCS 2020, pages 208–222. Springer-Verlag, Berlin, 2001.

[112] D. Kahn. *La Guerre des Codes Secrets : des hiéroglyphes à l'ordinateur*. InterEditions, 1980.

[113] J. Katz, S. Myers, and R. Ostrovsky. Cryptographic counters and applications to electronic voting. In *Eurocrypt '01*, LNCS 2045. Springer-Verlag, 2001.

[114] J. Kilian and K. Sako. Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth. In *Eurocrypto '95*, LNCS 921, pages 393–403. Springer-Verlag, 1995.

[115] B. King. Improved Methods to Perform Threshold RSA. In *Asiacrypt '00*, LNCS 1976, pages 359–372. Springer-Verlag, 2000.

[116] P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Others Systems. In *Crypto '96*, LNCS 1109, pages 104–113. Springer-Verlag, 1996.

[117] P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Crypto '99*, LNCS 1666, pages 388–397. Springer-Verlag, 1999.

[118] S. K. Langford. Threshold DSS Signatures without a Trusted Party. In *Crypto '95*, LNCS 963, pages 395–409. Springer-Verlag, 1995.

[119] A. Lenstra and H. Lenstra. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.

[120] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes.

[121] C.H. Lim and P.J. Lee. Another Method for Attaining Security Against Adaptively Chosen Ciphertext Attacks. In *Crypto '93*, LNCS 773, pages 287–296. Springer-Verlag, Berlin, 1994.

[122] U. M. Maurer. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete logarithms. In *Crypto '94*, LNCS 839, pages 271–281. Springer-Verlag, 1994.

[123] U. M. Maurer and S. Wolf. Diffie Hellman Oracles. In *Crypto '96*, LNCS 1109, pages 268–282. Springer-Verlag, 1996.

[124] S. Micali and P. Rogaway. Secure computation. In *Crypto '91*, LNCS 576, pages 392–404. Springer-Verlag, 1991.

[125] S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems. In *Crypto '95*, LNCS 963, pages 185–196. Springer-Verlag, 1995.

[126] G. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

[127] S. Miyazaki, K. Sakurai, and M. Yung. On Threshold RSA-signing with no dealer. In *ICICS '99*, LNCS 1787. Springer-Verlag, 1999.

[128] D. Naccache and J. Stern. A New Public Key Cryptosystem Based on Higher Residues. In *Proc. of the 5th CCCS*, pages 59–66. ACM press, 1998.

[129] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure agains Chosen Ciphertext Attacks. In *Proc. of the 22nd STOC*, pages 427–437. ACM Press, 1990.

[130] NIST. Digital Signature Standard (DSS). Federal Information Processing Standards PUBlication 186, november 1994.

[131] K. Ohta and T. Okamoto. Divertible Zero-Knowledge Interactive Proofs and Commutative Random Self-Reducibility. In *Eurocrypt '89*, LNCS 434, pages 134–149. Springer-Verlag, 1990.

[132] K. Ohta and T. Okamoto. On Concrete Security Treatment of Signatures Derived from Identification. In *Crypto '98*, LNCS 1462, pages 354–369. Springer-Verlag, 1998.

[133] T. Okamoto. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In *Crypto '92*, LNCS 740, pages 31–53. Springer-Verlag, 1992.

[134] T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In *CT – RSA '2001*, LNCS 2020, pages 159–175. Springer-Verlag, Berlin, 2001.

[135] T. Okamoto and D. Pointcheval. The Gap-Problems: a New Class of Problems for the Security of Cryptographic Schemes. In *PKC '2001*, LNCS 1992. Springer-Verlag, Berlin, 2001.

[136] T. Okamoto and S. Uchiyama. A New Public-Key Cryptosystem as Secure as Factoring. In *Eurocrypt '98*, LNCS 1403, pages 308–318. Springer-Verlag, 1998.

[137] R. Ostrovski and M. Yung. How To Withstand Mobile Virus Attacks. In *Proc. 10th ACM Conf. on Principles of Distributed Systems*, 1991.

[138] P. Paillier. *Cryptographie à Clé Publique Basée sur la Résiduosité de Degré Composite*. PhD thesis, École Nationale Supérieure des Télécommunications, september 1999.

[139] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Eurocrypt '99*, LNCS 1592, pages 223–238. Springer-Verlag, 1999.

[140] P. Paillier and D. Pointcheval. Efficient Public-Key Cryptosystems Provably Secure against Active Adversaries. In *Asiacrypt '99*, LNCS 1716, pages 165–179. Springer-Verlag, Berlin, 1999.

[141] V. Pan. Faster solution of the key equation for decoding BCH error-correcting codes. In *STOC '97*, pages 168–175. ACM, 1997.

[142] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1995.

[143] T.P. Pedersen. A Threshold Cryptosystem without a Trusted Party. In *Eurocrypt'91*, LNCS 547, pages 522–526. Springer-Verlag, 1991.

[144] T.P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Crypto'91*, LNCS 576, pages 129–140. Springer-Verlag, 1991.

[145] S. C. Pohlig and M. E. Hellman. An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance. *IEEE Transactions on Information Theory*, IT–24(1):106–110, january 1978.

[146] D. Pointcheval. *Les Preuves de Connaissance et leurs Preuves de Sécurité*. PhD thesis, Université de Caen, december 1996.

[147] D. Pointcheval. Chosen-Ciphertext Security for any One-Way Cryptosystem. In *PKC '2000*, LNCS 1751, pages 129–146. Springer-Verlag, Berlin, 2000.

[148] D. Pointcheval. Self-Scrambling Anonymizers. In *Financial Cryptography '2000*, LNCS. Springer-Verlag, Berlin, 2000.

[149] D. Pointcheval and J. Stern. Provably Secure Blind Signature Schemes. In *Asiacrypt '96*, LNCS 1163, pages 252–265. Springer-Verlag, 1996.

[150] D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In *Eurocrypt '96*, LNCS 1070, pages 387–398. Springer-Verlag, 1996.

[151] D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[152] C. Pomerance. On the distribution of pseudoprimes. In *Mathematics of Computation*, 37(156), pages 587–593, 1981.

[153] C. Pomerance. Two methods in elementary analytic number theory. In *Number theory and applications*, pages 135–161. Kluwer Academic Publishers, 1989.

[154] G. Poupard. *Authentification d'Entité, de Messages et de Clés Cryptographiques : Théorie et Pratique*. PhD thesis, École Polytechnique, may 2000.

[155] G. Poupard and J. Stern. Generation of Shared RSA Keys by Two Parties. In K .Ohta and D. Pei, editors, *Asiacrypt '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 11–24. Springer-Verlag, 1998.

[156] G. Poupard and J. Stern. Fair Encryption of RSA Keys. In *Proceedings of Eurocrypt 2000*, Lecture Notes in Computer Science, pages 172–189. Springer-Verlag, 2000.

[157] G. Poupard and J. Stern. Short Proofs of Knowledge for Factoring. In *PKC 2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 2000.

[158] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[159] T. Rabin. A Simplified Approach to Threshold and Proactive RSA. In *Crypto '98*, LNCS 1462, pages 89–104. Springer-Verlag, 1998.

[160] C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *Crypto '91*, LNCS 576, pages 433–444. Springer-Verlag, 1992.

[161] R. Rivest. Finding Four Million Large Random Primes. In *Crypto '90*, LNCS 537, pages 625–626. Springer-Verlag, 1991.

[162] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, february 1978.

[163] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystem. *Communications of the ACM*, 21(2):120–126, 1978.

[164] A. Sahai. Non-Malleable Non-Interactive Zero-Knowledge and Chosen-Ciphertext Security. In *FOCS '99*, LNCS 2139. IEEE, 1999.

[165] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pages 522–523. ACM, 1994.

[166] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Crypto '89*, LNCS 435, pages 235–251. Springer-Verlag, 1990.

[167] C. P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[168] C. P. Schnorr and M. Jakobsson. Security of Signed ElGamal Encryption. In *Asiacrypt '2000*, LNCS 1976, pages 458–469. Springer-Verlag, Berlin, 2000.

[169] B. Schoenmakers. Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *Crypto '99*, LNCS 1666, pages 148–164. Springer-Verlag, 1999.

[170] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, Nov. 1979.

[171] C. E. Shannon. A Mathematical Theory of Communication. *Bell system technical journal*, 27(4):379–423, 623–656, 1948.

[172] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell system technical journal*, 28(4):656–715, 1949.

[173] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Eurocrypt '97*, LNCS 1233, pages 256–266. Springer-Verlag, 1997.

[174] V. Shoup. Practical Threshold Signatures. In *Eurocrypt '00*, LNCS 1807, pages 207–220. Springer-Verlag, 2000.

[175] V. Shoup and R. Gennaro. Securing Threshold Cryptosystems against Chosen Ciphertext Attack. In *Eurocrypt '98*, LNCS 1403, pages 1–16. Springer-Verlag, 1998.

[176] R.D. Silverman. Fast Generation of Random, Strong RSA Primes. RSA Laboratories, May 1997.

[177] S. Singh. *L'histoire des codes secrets*. J.C. Lattès, 2000.

[178] M. Stadler. Publicly verifiable secret sharing. In *Eurocrypt '96*, LNCS 1070, pages 190–199. Springer-Verlag, 1996.

[179] J. Stern. *La Science du Secret*. Odile Jacob, 1998.

[180] J. Stern and S. Vaudenay. CS-Cipher. In *Fast Software Encryption '98*, LNCS 1318, pages 189–205. Springer-Verlag, 1998.

[181] D. R. Stinson. *Cryptography, Theory and Practice*. CRC Press, 1995.

[182] S. Vanstone and R. Zuccherato. Elliptic Curve Cryptosystem Using Curves of Smooth Order Over the Ring Z_n. *IEEE Transaction on Information Theory*, IT-43, 1997.

[183] S. Vaudenay. On the Security of CS-Cipher. In *Fast Software Encryption '99*, LNCS 1636, pages 260–274. Springer-Verlag, 1999.

[184] A. C. Yao. Protocols for secure computations. In *Proc. of the 23nd FOCS*, pages 160–164. IEEE, 1982.

[185] A. Young and M. Yung. Auto-Recoverable Auto-Certifiable Cryptosystems. In *Eurocrypt '98*, LNCS 1403, pages 17–31. Springer-Verlag, 1998.

[186] A. Young and M. Yung. A PVSS as Hard as Discrete Log and Shareholder Separability. In *PKC '01*, LNCS 1992, pages 287–299. Springer-Verlag, 2001.

Résumé

La cryptographie a pour but de garantir la protection des communications contre différents types d'adversaires. La cryptographie partagée permet d'une part, de partager le pouvoir de déchiffrer ou de signer entre plusieurs serveurs, et d'autre part, de considérer des adversaires qui peuvent attaquer différents serveurs. Ces types d'attaque sont réalistes dans les applications pratiques exigeant un fort niveau de sécurité. La cryptographie partagée construit des protocoles sûrs et efficaces en pratique contre ces adversaires. Elle est à mi-chemin entre la cryptographie et le calcul multipartie qui permet de réaliser le calcul de certaines fonctions grâce à un protocole distribué.

Dans cette thèse, nous avons abordé le problème du partage complet d'un protocole sans faire appel à aucun moment à un distributeur de confiance. Après des rappels de cryptographie et de cryptographie partagée, nous avons présenté un tel protocole pour le schéma RSA. Puis, nous avons proposé le partage complet de l'algorithme de chiffrement de Paillier par rapport à différents types d'attaquants. Ensuite, nous avons proposé une variante à la génération partagée d'une clé Diffie-Hellman. Nous avons enfin montré dans la troisième partie que les différents partages du schéma homomorphe de Paillier étaient utiles pour prendre en compte les exigences de protocoles de plus haut-niveau comme un schéma de loterie, de vote électronique ou de recouvrement de clé.

Mots-clés: Cryptographie, Cryptographie à clé publique, Cryptographie à seuil, Signature et chiffrement partagés

Abstract

The cryptography aims at protecting communications against different kinds of adversaries. Threshold cryptography allows on the one hand, to share the decryption or signature power amongst several servers, and on the other hand, to consider adversaries that may attack many servers. These kinds of attacks are realistic in practical applications requiring a high security level. Threshold cryptography designs secure and efficient protocols against these adversaries. It is between the cryptography area and multiparty computation which allows to distribute the calculation of some functions.

In this thesis, we have dealt with the problem of complete distributed protocols without requiring a trusted dealer at any moment. After some results and facts of cryptography and threshold cryptography, we have presented such a protocol for the RSA scheme. Then, we have proposed the complete sharing of Paillier cryptosystem against different kinds of attackers. Next, we have proposed a variant of a distributed key generation for Diffie-Hellman keys. Finally, we have showed in the third party that the different kinds of sharing of the homomorphic cryptosystem of Paillier are needed in order to take into account the requirements of high-level schemes such as lottery, electronic voting or key recovery systems.

Keywords: Cryptography, Public-key cryptography, Threshold cryptogaphy, Shared signature schemes and cryptosystems