École Normale Supérieure École Polytechnique

Cryptanalyses et preuves de sécurité de schémas à clé publique

THÈSE

présentée et soutenue publiquement le 16 Mai 2001

pour l'obtention du

Doctorat de l'École Polytechnique

par

Jean-Sébastien Coron

Composition du jury :

Rapporteurs :	Mihir Bellare
	Marc Girault
Examinateurs :	David Naccache
	Ronald Rivest
	Adi Shamir
	Jacques Stern (directeur)
	Jean-Marc Steyaert
	Serge Vaudenay

Gemplus, Laboratoire de Cryptographie, 34 rue Guynemer, 92447 Issy-les-Moulineaux

Remerciements

Je remercie tout d'abord Jacques Stern pour l'excellente qualité de son encadrement et la liberté donnée dans l'orientation de mes travaux.

Je remercie David Naccache qui m'a accueilli il y a trois ans dans son équipe de recherche en cryptographie de la société Gemplus. Il est à l'origine d'un grand nombre d'idées développées dans cette thèse; son enthousiasme et sa générosité me paraissent sans limite. Je remercie aussi mes collègues de Gemplus avec qui j'ai le plaisir de travailler quotidiennement, Pascal Paillier, Héléna Handschuh et Christophe Tymen.

Je remercie également les membres du Groupe de Recherche en Complexité et Cryptographie de l'ENS, notamment Phong Nguyen et David Pointcheval, avec qui j'ai eu d'intéressantes discussions.

Enfin, je remercie Mihir Bellare et Marc Girault pour avoir accepté d'être rapporteurs, ainsi que Ronald Rivest, Adi Shamir, Serge Vaudenay et Jean-Marc Steyaert qui ont bien voulu participer à mon jury de thèse.

Avant-Propos

Cette thèse a pour but d'étudier la sécurité de certains schémas de chiffrement et de signature basés sur l'algorithme RSA, qui sont couramment utilisés dans la pratique. Cette thèse présente l'intégralité de mes travaux de doctorat [16, 25, 28, 33, 34], à l'exception de [18, 23, 24, 26, 27, 29, 30, 31, 32], et peut se diviser en deux parties: une partie "cryptanalyse" et une partie "preuve de sécurité".

Ainsi, après une introduction générale à la cryptographie au chapitre un, on présente dans le chapitre deux l'algorithme RSA ainsi que les attaques dont il a fait l'objet depuis son invention. Ensuite, dans le chapitre trois, on décrit plus en détail les principales attaques connues sur le chiffrement RSA. Dans le chapitre quatre, on montre que le standard de chiffrement PKCS#1 v1.5 est vulnérable à certaines attaques. Au chapitre cinq, on rappelle les principales attaques connues sur les signatures RSA. On montre aussi comment étendre l'attaque de Girault et Misarsky sur les signatures RSA à redondance affine, pour des tailles de redondance plus grande. Au chapitre six, on montre que les normes de signature ISO 9796-1 et ISO 9796-2 sont vulnérables à certaines attaques.

Dans la seconde partie de la thèse, on étudie les moyens de prouver la sécurité de schémas de signature à clef publique. On rappelle au chapitre sept comment formaliser les notions de sécurité que l'on souhaite atteindre. Ensuite, on donne au chapitre huit une preuve de sécurité modifiée du schéma de signature de Gennaro-Halevi-Rabin, permettant de prendre en compte la taille de la sortie de la fonction de hachage. Au chapitre neuf, on donne une preuve de sécurité améliorée du schéma de signature Full-Domain-Hash. Enfin, au chapitre dix, on montre que la preuve précédente est optimale, et plus généralement, que si dans un schéma de signature chaque message possède une signature unique, on ne peut pas atteindre le même niveau de sécurité que l'on aurait obtenu si chaque message avait plusieurs signatures possibles.

Table des matières

Partie I. Généralités sur la cryptologie

1. Introduction à la cryptologie 13
1.1 Introduction 13
1.2 Objectifs de la cryptographie 14
1.2.1 Confidentialité 14
1.2.2 Intégrité 15
1.2.3 Authenticité 15
1.3 Théorie de la complexité 16
1.3.1 Algorithme et machine de Turing 16
1.3.2 Complexité d'un algorithme 17
1.3.3 Problèmes de décision 17
1.3.4 Classes de complexité 18
1.3.5 Machines de Turing étendues 18
1.3.6 Réduction d'un problème à un autre 19
1.4 Conclusion 19

2. Le cryptosystème RSA 21
2.1 Introduction 21
2.2 Attaque par factorisation du module N 23
2.2.1 Méthodes dont la complexité est fonction de la taille du facteur premier 23
2.2.2 Méthodes dont la complexité est fonction de la taille de l'entier 24
2.3 Attaques élémentaires 26
2.3.1 Forge existentielle 26
2.3.2 Attaque par masquage 26
2.4 Attaques multiplicatives 27
2.4.1 Attaque de Desmedt-Odlyzko 27
2.4.2 Attaque de Kaliski-Robshaw 29
2.5 Attaque avec exposant privé petit 29
2.6 Conclusion 30

Partie II. Cryptanalyse de schémas de chiffrement basés sur RSA

3. Attaques sur le chiffrement RSA 33
3.1 Modèle d'attaque 33
3.2 Attaques avec exposant public e petit 34
3.2.1 Le théorème de Coppersmith 34
3.2.2 L'attaque de Håstad 35
3.2.3 L'attaque de Franklin-Reiter sur les messages liés 36
3.2.4 L'attaque de Coppersmith sur les redondances courtes 37
3.3 Conclusion 37

4. Attaques contre PKCS#1 v1.5 chiffrement 39
4.1 Introduction 39
4.2 Le standard de chiffrement PKCS#1 v1.5 40
4.3 L'attaque de Bleichenbacher 40
4.4 Nouvelle attaque à clair choisi pour e petit 41
4.4.1 Isoler les facteurs de Δ inférieurs à une borne B 42
4.4.2 Identification des candidats pour l'entier ω 44
4.4.3 Retrouver m en utilisant l'attaque sur RSA avec petit exposant avec messages liés 45
4.5 Comparaison avec l'attaque de Coppersmith sur RSA avec petit exposant 45
4.6 Expérimentation de l'attaque sur un module de 1024-bits 46
4.7 Attaque à clair choisi pour un exposant de chiffrement quelconque 48
4.7.1 Description de l'attaque 48
4.7.2 Analyse 49
4.8 Conclusion 50

Partie III. Cryptanalyse de schémas de signature basés sur RSA

5. Attaques sur les signatures RSA 53
5.1 Les différents modèles d'attaque 53
5.2 Attaque de Desmedt et Odlyzko 54
5.3 Attaques sur les signatures RSA avec redondance 54
5.3.1 L'attaque de De Jonge et Chaum 55
5.3.2 L'attaque de Girault et Misarksy 56
5.3.3 L'attaque de Misarsky 57
5.4 Extension de l'attaque de Girault et Misarsky 57
5.4.1 Description de l'attaque 57
5.4.2 Application pratique 59
5.5 Conclusion 60

6. Attaques sur les standards ISO 9796-1 et ISO 9796-2 61
6.1 Introduction 61
6.2 Le schéma de signature de Rabin-Williams 61
6.2.1 Rappels en théorie des nombres 61
6.2.2 Le schéma de signature de Rabin-Williams 62
6.3 Extension de l'attaque de Desmedt et Odlyzko 63
6.4 Complexité de l'attaque 65
6.5 Cryptanalyse du schéma de signature ISO/IEC-9796-1 67
6.5.1 Le schéma ISO/IEC-9796-1 67
6.5.2 Attaque sur une version modifiée de ISO/IEC-9796-1 68
6.5.3 Application pratique: attaque sur la version modifiée 69
6.5.4 Attaque sur le vrai standard ISO 9796-1 70
6.5.5 Application pratique: attaque sur ISO 9796-1 71
6.5.6 L'attaque de Grieu sur ISO 9796-1 73
6.6 Sécurité des signatures ISO 9796-2 74
6.6.1 Inclusion partielle du message 74
6.6.2 Inclusion totale du message 75
6.7 Conclusion 75

Partie IV. Preuves de sécurité pour schémas de signature

7. Introduction aux preuves de sécurité pour les schémas de signature numérique 79
7.1 Introduction 79
7.2 Schéma de signature 80
7.3 Sécurité d'un schéma de signature 81
7.4 Preuve de sécurité d'un schéma de signature 81
7.5 Conclusion 82

8. Etude de la sécurité du schéma de signature de Gennaro-Halevi-Rabin 83
8.1 Introduction 83
8.2 Le schéma de signature Gennaro-Halevi-Rabin 84
8.2.1 Définition 84
8.2.2 Preuve de sécurité 84
8.3 Attaque du schéma de Gennaro-Halevi-Rabin 85
8.3.1 Méthode d'attaque 85
8.3.2 Quelques propriétés des nombres lisses 86
8.3.3 Attaque générique du schéma de signature 87
8.3.4 Temps de calcul de l'attaque en pratique 88
8.4 Nouvelle preuve de sécurité et attaque générique optimale 89
8.4.1 Nouvelle preuve de sécurité 89

8.4.2 Attaque générique optimale 92
8.5 Conclusion 92

9. Sécurité du schéma de signature Full Domain Hash 93
9.1 Introduction 93
9.2 Le schéma de signature FDH 94
9.3 Nouvelle preuve de sécurité 95
9.4 Conclusion 97

10. Preuves de sécurité pour schémas de signature à signature unique 99
10.1 Introduction 99
10.2 Sécurité des schémas de signature à signature unique dans le modèle de l'oracle aléatoire 100
10.2.1 Preuve de sécurité pour FDH 100
10.2.2 Sécurité des schémas à signature unique dans le modèle de l'oracle aléatoire 101
10.2.3 Extension aux réductions exécutant le forgeur plus d'une fois . . 110
10.2.4 Application au schéma de signature Full Domain Hash 113
10.3 Preuves de sécurité dans le modèle standard 114
10.4 Conclusion 117

Partie I

Généralités sur la cryptologie

1. Introduction à la cryptologie

1.1 Introduction

Longtemps, la cryptologie est restée un art réservé aux militaires et aux espions. C'est au cours de ces vingt-cinq dernières années que la cryptologie est devenue une science basée sur les mathématiques et l'informatique. Le développement des moyens de communication à l'échelle planétaire constitue aujourd'hui le moteur de la recherche publique en cryptologie. Les applications principales de la cryptologie sont bien sûr le chiffrement des messages pour en garantir la confidentialité, mais aussi l'authentification à distance, la signature de documents numériques, et le contrôle de l'intégrité des messages transmis sur un réseau informatique.

Il existe deux modèles distincts en matière de chiffrement des données. Le premier est le chiffrement symétrique, dans lequel une même clé secrète est partagée entre les correspondants. Pour chiffrer un message, on lui applique un algorithme qui utilise cette clé secrète, et on utilise la même clé secrète pour déchiffrer le message. L'algorithme de chiffrement symétrique le plus connu et le plus utilisé est l'algorithme DES [2] adopté en 1976. Cet algorithme s'est révélé raisonnablement résistant aux attaques par cryptanalyse différentielle [10] et linéaire [63] apparues par la suite. La taille de la clé secrète est cependant limitée à 56 bits, ce qui le rend aujourd'hui vulnérable aux attaques par recherche exhaustive de la clé. Le bureau des standards Américain a lancé en 1997 un appel à proposition pour définir un remplaçant du DES dénommé AES, et utilisant une clé de taille allant de 128 bits à 256 bits, ce qui rend la recherche exhaustive impossible. C'est l'algorithme Rijndael qui a finalement été sélectionné au mois d'octobre 2000 pour remplacer le DES. Nous référons le lecteur aux nombreux ouvrages généraux sur la cryptographie ([78], [64] et [80]) qui offrent un tour d'horizon relativement complet des techniques de chiffrement symétrique.

Le deuxième modèle est le chiffrement à clé publique ou chiffrement asymétrique, un concept inventé en 1976 par Whitfield Diffie et Martin Hellman [40], marquant ainsi la naissance de la cryptologie moderne. Chaque utilisateur possède deux clés: une clé publique et une clé privée. Pour chiffrer un message, on utilise la clé publique du destinataire, qui seul peut le déchiffrer en utilisant sa clé privée correspondante. On résout ainsi le problème de la distribution des clés: deux personnes peuvent communiquer de façon confidentielle sans jamais s'être rencontrées auparavant.

La première réalisation d'un algorithme à clé publique fut l'algorithme RSA [75], proposé en 1978 par Ronald Rivest, Adi Shamir et Leonard Adleman, dont la sécurité repose sur la difficulté de factoriser des grands nombres. On ne sait pas si ce problème mathématique est réellement difficile, mais on ne connaît pas aujourd'hui de méthode efficace pour le résoudre.

Dans ce chapitre, nous revenons plus en détail sur les objectifs que permet d'atteindre la cryptographie. Nous rappelons ensuite certains résultats élémentaires de la théorie de la complexité, fréquemment utilisée en cryptographie.

1.2 Objectifs de la cryptographie

Les objectifs fondamentaux de la cryptographie sont la confidentialité, l'intégrité et l'authenticité.

1.2.1 Confidentialité

La confidentialité permet de protéger le contenu des informations sauvegardées ou transmises sur un réseau. Seules les personnes autorisées doivent pouvoir accéder aux informations ainsi protégées. Les protections utilisées peuvent être physiques ou mathématiques. Le *chiffrement de l'information* permet de résoudre le problème de la confidentialité: une personne souhaitant transmettre un messages lui applique au préalable une fonction dite de chiffrement, et transmet le résultat au destinataire. Ce dernier retrouve le message original en utilisant une fonction de déchiffrement.

Dans le modèle de la cryptographie à clé secrète (aussi appelé chiffrement symétrique), les deux parties partagent la même clé de chiffrement et de déchiffrement, qui doit être gardée secrète. Les deux personnes jouent ainsi un rôle symétrique.

Dans le modèle de la cryptographie à clé publique, le chiffrement est publique tandis que le déchiffrement est confidentiel. Pour envoyer un message chiffré, on applique une fonction de chiffrement utilisant la clé publique du destinataire. Seul le destinataire peut retrouver le message original à l'aide de sa clé privée. Les deux clés sont liées mathématiquement, mais il doit être impossible dans la pratique de retrouver la clé privée à partir de la clé publique.

Les algorithmes à clé secrète et à clé publique ont chacun leur domaine d'utilisation, leurs avantages et leurs inconvénients. Les algorithmes à clé secrète sont en général beaucoup plus rapides, et ont des clés plus courtes. L'avantage principal des algorithmes à clé publique réside dans la gestion des clés: seule la clé privée est maintenue secrète, les clés ont une durée de vie plus longue, et le nombre de clés requises pour que des personnes communiquent entre elles sur un réseau est considérablement réduit.

Dans la pratique, on peut combiner les avantages de la clé publique et de la clé secrète. Par exemple, le chiffrement à clé publique permet d'établir une clé secrète permettant à deux entités A et B de communiquer de façon sécurisée. Dans ce

scénario, l'entité A choisit aléatoirement une clé secrète, dite clé de session, et la chiffre en utilisant la clé publique de B. Ce dernier retrouve la clé de session en utilisant sa clé privée, et ensuite A et B peuvent communiquer de façon sécurisée en utilisant la clé de session en chiffrement symétrique. Des applications très populaires telles que SSL [57] mettent en oeuvre ce principe plusieurs millions de fois par jour.

1.2.2 Intégrité

Le problème de l'intégrité est le contrôle du contenu : on veut pouvoir détecter toute modification, accidentelle ou intentionnelle, des données sauvegardées ou transmises. On résout ce problème à l'aide des *fonctions de hachage*, qui à une donnée de longueur arbitraire associe une donnée de taille fixe appelée *empreinte*. Une fonction de hachage doit être à *sens unique*, c'est à dire qu'il doit être impossible étant donné une empreinte de trouver un message qui se hache en cette valeur. La fonction de hachage doit être également *sans collisions*, c'est à dire qu'il doit être impossible en pratique de trouver deux messages distincts qui se hachent en la même valeur. Ainsi, personne ne peut modifier le message en conservant la même empreinte.

1.2.3 Authenticité

La notion d'authenticité s'applique à la fois aux personnes, on parle dans ce cas d'identification, et aux documents, ce qui correspond à l'authentification.

Pour s'identifier, un individu prouve qu'il connaît une information secrète. Une notion particulièrement élégante est celle de preuve interactive à divulgation nulle de connaissance, dans laquelle un individu (le prouveur) convainc une autorité (le vérifieur) qu'il possède une information secrète, sans révéler aucune information sur la donnée secrète elle-même. Ainsi, il n'est pas possible qu'un espion écoutant la ligne se fasse ensuite passer pour l'individu, l'espion n'ayant appris aucune information sur la donnée secrète.

L'authentification d'un document permet de prouver son caractère authentique, c'est à dire son lien avec une entité précise. On garantit ainsi l'origine de l'information contenue dans le document. Les techniques utilisées sont les codes d'authentification de message (notés de façon usuelle MAC pour *Message Authentification Code*) en clé secrète, et la signature électronique en clé publique. Ces techniques assurent implicitement l'intégrité du document.

Dans un code d'authentification de message, l'entité à l'origine du document partage la même clé secrète que l'entité qui en vérifie l'origine. La même clé secrète permet donc de prouver l'authenticité du document et d'en vérifier l'authenticité.

La signature numérique d'un document électronique permet de lier le contenu du document à son signataire, en assurant à la fois l'intégrité du document (le document n'a pas été modifié) et l'authenticité du signataire (c'est bien lui qui a signé). La vérification de la signature est publique: n'importe qui peut la vérifier en utilisant la clé publique du signataire. On obtient ainsi la propriété de non-répudiation de

la signature: le signataire ne peut pas nier par la suite avoir signé le document. Le lecteur peut se référer à [64] pour une étude plus précise des codes d'authentification de message et des algorithmes de signature électronique.

1.3 Théorie de la complexité

La théorie de la complexité permet de rendre compte des ressources nécessaires à la résolution d'un problème donné et de classer les problèmes suivant leur difficulté. Les ressources généralement prises en compte sont le temps de calcul, et parfois l'espace mémoire. La cryptologie fait appel à la théorie de la complexité pour évaluer la sécurité des schémas cryptographiques, et savoir quelles ressources sont nécessaires pour en mettre en défaut leur sécurité.

1.3.1 Algorithme et machine de Turing

On appelle *algorithme* une procédure qui, à partir de données en entrée, se termine et fournit des données en sortie. Un algorithme peut être modélisé par une *machine de Turing*, dont nous reprenons ici la définition figurant dans la traduction française [49] de l'article d'Alan Turing [81]

Définition 1.3.1 (Machine de Turing). *Une machine de Turing consiste en:*

- un ruban, avec une extrémité à gauche, infini à droite, divisé en cases de même taille. On peut penser par exemple à une bande magnétique suffisamment longue;

- un ensemble fini de symboles, par exemple $\{0, 1, s, d, f\}$*, avec* 0 *et* 1 *pour la numérotation binaire,* s *pour séparer deux expressions,* d *pour le début du ruban et* f *pour signifier que ce qui est à droite n'a plus d'importance. Ils seront inscrits et lus sur le ruban à raison d'un symbole par case;*

- une tête de lecture/écriture qui se déplace sur le ruban. A chaque impulsion, la tête peut soit rester sur place, soit reculer (si possible) ou avancer, dans les deux cas d'une case. La tête a le droit de lire ou d'écrire dans la case qu'elle est en train de sonder. Pour rester concret, on peut penser que c'est la bande et non la tête qui bouge;

- un ensemble fini d'états. Ces états permettent de distinguer plusieurs comportements possibles. Notamment l'état D *représente l'état de départ et l'état* F *symbolise la fin du "programme" et donc la lecture du résultat par un observateur extérieur;*

- un ensemble fini d'instructions: à chaque impulsion, en fonction du symbole c *lu par la tête, en fonction de l'état courant* S*, la tête écrit un nouveau symbole* c'*, effectue un déplacement noté* -1 *(gauche),* $+1$ *(droite) ou* 0 *(immobile) et passe à l'état* S'*.*

Pour utiliser la machine de Turing, on inscrit les données d'entrée de l'algorithme sur le ruban, puis la machine est initialisée dans l'état de départ D. Si la machine s'arrête, la sortie est la donnée inscrite sur le ruban. La machine de Turing permet ainsi de décrire un algorithme sous la forme d'une machine abstraite sur laquelle se déroule un programme.

1.3.2 Complexité d'un algorithme

L'identification entre algorithmes et machines de Turing permet de définir rigoureusement le coût d'un algorithme. Etant donné une machine de Turing M dont le ruban contient initialement la donnée d'entrée x, on définit la *complexité* $T_M(x)$ d'un calcul comme étant le nombre de transitions nécessaires à la machine pour passer de l'état initial D à l'état final F. On définit également la *complexité spatiale* $S_M(x)$ comme étant le nombre de cases du ruban utilisées pendant le calcul. En notant $|x|$ la taille d'une donnée x, la *complexité dans le pire cas* d'une machine M est le temps maximal de calcul de M en fonction de la taille de x:

$$T_M(n) = \sup\{T_M(x), |x| = n\}$$

On définit également la *complexité en moyenne* d'une machine M comme le temps moyen d'exécution de M sur l'ensemble des entrées d'une taille fixée $|x|$, avec p_n une distribution de probabilités sur l'ensemble des mots de taille n:

$$\bar{T_M}(n) = \sum_{|x|=n} p_n(x) \cdot T_M(x)$$

On dit qu'une machine de Turing M (ou un algorithme) est *polynomiale* si sa complexité dans le pire cas est inférieure à un polynôme en la taille des entrées. Les algorithmes polynomiaux s'identifient généralement aux algorithmes efficaces dans la pratique, permettant d'obtenir rapidement un résultat.

1.3.3 Problèmes de décision

Un algorithme permet de résoudre un problème, qui pose une question relative à une donnée. Une *instance* du problème est une valeur de cette donnée. En représentant les données en binaire, on formalise un problème comme un sous-ensemble Π de $\{0,1\}^* \times \{0,1\}^*$, $\{0,1\}^*$ étant l'ensemble des mots formés de 0 et de 1. On dit qu'un mot $\tau \in \{0,1\}^*$ est une *solution* de l'instance $\sigma \in \{0,1\}^*$ si (σ, τ) appartient à Π. On suppose qu'à toute instance correspond au moins une solution, ce qui revient à considérer la réponse "pas de solution" comme une solution. On dit qu'une machine de Turing résout un problème Π si, recevant en entrée une instance quelconque de Π, elle finit par s'arrêter et renvoyer une solution.

Un domaine important de la théorie de la complexité concerne l'évaluation des problèmes de décision, soit l'ensemble des problèmes qui ont pour réponse `Oui` ou `Non`. Beaucoup de problèmes dont l'énoncé n'est pas décisionnel peuvent se ramener à des problèmes décisionnels. Les problèmes décisionnels sont en bijection avec les langages de $\{0,1\}^*$: à tout problème décisionnel Π, on associe le langage formé des mots σ tels que la solution de σ est `oui`.

1.3.4 Classes de complexité

On définit les classes de complexité suivantes:

- La classe $\mathcal{P}$ est l'ensemble des problèmes de décision que l'on peut résoudre en temps polynomial.

- La classe $\mathcal{NP}$ est l'ensemble des problèmes de décision pour laquelle la réponse `oui` est vérifiable en temps polynomial, en utilisant une information supplémentaire appelée *certificat*.

- La classe $co-\mathcal{NP}$ est de manière analogue l'ensemble des problèmes de décision dont la réponse négative est vérifiable en temps polynomial.

On a évidemment $\mathcal{P} \subseteq \mathcal{NP}$, mais on ne sait pas si l'inclusion est stricte ou pas. La conjecture la plus célèbre de l'informatique théorique est que $\mathcal{P} \neq \mathcal{NP}$.

1.3.5 Machines de Turing étendues

Pour accroître les capacités de calcul et donc réduire la complexité, on peut être tenté d'ajouter plusieurs rubans à la machine de Turing initiale. En fait, il est toujours possible de simuler une telle machine à l'aide de la machine de Turing initiale, la simulation étant de plus polynomiale. Ainsi, on ne change pas les classes de complexité $\mathcal{P}$ et $\mathcal{NP}$ en remplaçant la machine de Turing à un seul ruban par une machine à plusieurs rubans. Le fait que les fonctions de complexité de ces machines soient polynomialement liées permet de parler de temps de calcul polynomial indépendamment du modèle utilisé.

Une *machine de Turing probabiliste* est une machine de Turing disposant d'une source de bits aléatoires, uniformément distribués dans $\{0,1\}$. Cette source est modélisée par un ruban supplémentaire, appelé *ruban aléatoire*, contenant une suite infinie de bits aléatoires. Comme on suppose que la machine de Turing finit par s'arrêter quelle que soit l'entrée, le nombre de bits du ruban aléatoire lus est borné par une fonction de la taille de l'entrée, ce qui permet de définir une distribution de probabilité uniforme sur le ruban aléatoire. On définit la *complexité espérée* d'une machine de Turing probabiliste comme l'espérance, prise sur le ruban aléatoire, de la complexité de la machine. On dit que la machine est *polynomiale* lorsque cette complexité est polynomiale. Dans la pratique, la machine de Turing probabiliste est un modèle réaliste d'ordinateur disposant d'un générateur de bits aléatoires.

On définit la classe $\mathcal{BPP}$ comme l'ensemble des problèmes décisionnels tels qu'il existe un machine de Turing polynomiale qui renvoie `oui` avec probabilité supérieure à $2/3$ si la solution est `oui`, et qui renvoie `non` avec probabilité supérieure à $2/3$ si la solution est `non`. Les problèmes décisionnels de la classe $\mathcal{BPP}$ sont généralement des problèmes pouvant être résolus efficacement dans la pratique.

Une *machine de Turing avec oracle* est une machine de Turing ayant accès à un oracle lui permettant d'obtenir la réponse à une question. Formellement, l'oracle est une fonction f de $\{0,1\}^*$ dans $\{0,1\}^*$. La machine de Turing avec oracle possède

un ruban supplémentaire, le ruban à oracle, ainsi que deux états spéciaux: "invocation" et "apparition". Lorsque la machine est dans l'état "invocation", la valeur x présente sur le ruban à oracle est remplacée par $f(x)$, et la machine passe dans l'état "apparition". La machine de Turing à oracle obtient donc la réponse en temps unité.

1.3.6 Réduction d'un problème à un autre

La machine de Turing avec oracle permet de relier la difficulté d'un problème à un autre. On dit qu'un problème L_1 *se réduit polynomialement* à un problème L_2, s'il existe une machine de Turing polynomiale à oracle qui résout L_1, à l'aide d'un oracle résolvant L_2. Si L_1 se réduit polynomialement à L_2, cela signifie que L_2 est au moins aussi difficile que L_1, ou de manière équivalente que L_1 n'est pas plus difficile que L_2.

On définit également la notion de réduction randomisée entre deux problèmes. On dit qu'un problème décisionnel L_1 *se réduit aléatoirement* à un problème L_2, s'il existe une machine de Turing probabiliste polynomiale à oracle, l'oracle résolvant L_2, telle que la machine répond "oui" avec probabilité supérieure à $2/3$ si la solution de L_1 est oui, et sinon la machine répond "non" avec probabilité supérieure à $2/3$.

Un problème décisionnel L est dit *$\mathcal{NP}$-complet* s'il appartient à la classe $\mathcal{NP}$ et si tout problème L_1 de la classe $\mathcal{NP}$ se réduit polynomialement à L. Les problèmes $\mathcal{NP}$-complets sont les problèmes les plus difficiles de la classe $\mathcal{NP}$ au sens où ils sont au moins aussi difficiles que tout autre problème dans $\mathcal{NP}$. Il existe des centaines de problèmes dont on sait qu'ils sont $\mathcal{NP}$-complets [47]. Un exemple de problème $\mathcal{NP}$-complet est le problème de la somme du sac-à-dos: étant donné un ensemble d'entiers positifs $\{a_1, \ldots, a_n\}$ et un entier positif s, déterminer si oui ou non il existe un sous-ensemble des a_i dont la somme est s.

Les meilleurs algorithmes connus pour résoudre les problèmes $\mathcal{NP}$-complet ont tous une complexité exponentielle. La découverte d'un algorithme polynomial permettant de résoudre un problème $\mathcal{NP}$-complet entraînerait que $\mathcal{P} = \mathcal{NP}$, un résultat surprenant qui se révélerait désastreux pour la cryptologie moderne puisque la plupart des problèmes supposés difficiles en cryptologie seraient dans ce cas faciles à résoudre.

1.4 Conclusion

Nous avons présenté les principaux objectifs que permet d'atteindre la cryptologie, ainsi que les définitions élémentaires de la théorie de la complexité, qui permettent d'évaluer la difficulté à résoudre un problème donné.

Pendant longtemps, on a considéré qu'un système cryptographique était sûr si personne n'avait encore mis en défaut sa sécurité. Un axe de recherche important aujourd'hui est celui des preuves de sécurité: on montre que si l'on est capable

de mettre en défaut la sécurité d'un schéma, on peut alors résoudre un problème mathématique réputé difficile. On obtient ainsi une garantie sur la sécurité du schéma lui-même.

2. Le cryptosystème RSA

2.1 Introduction

Le cryptosystème RSA a été inventé en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman [75]. Premier algorithme à clé publique découvert, c'est aussi le plus utilisé de nos jours. Il permet d'assurer la confidentialité et l'authenticité des données. L'algorithme RSA se retrouve dans de nombreuses applications commerciales, qu'il s'agisse d'assurer la confidentialité des transactions électroniques sur Internet, de permettre la confidentialité et l'authenticité du courrier électronique, ou d'assurer une connection sécurisée avec un ordinateur distant.

Depuis sa parution, le cryptosystème RSA a fait l'objet de nombreuses études mettant en évidence certaines vulnérabilités, mais aucune n'a permis un cassage total. Les attaques mises en oeuvre ont surtout montré le danger d'une utilisation incorrecte de RSA. Dans ce chapitre, nous commençons par rappeler la définition de RSA et nous décrivons ensuite les principales attaques connues. On verra dans les chapitres suivants certaines attaques plus élaborées s'appliquant soit au chiffrement, soit à la signature.

Un *module* RSA est le produit $N = p \cdot q$ de deux grands nombres premiers p et q de taille voisine. La taille typique d'un module RSA est de 1024 bits, chaque facteur premier ayant une taille de 512 bits. Soient e et d deux entiers tels que $e \cdot d = 1 \bmod \phi(N)$, où $\phi(N) = (p-1) \cdot (q-1)$ est la fonction d'Euler. Rappelons que $\phi(N)$ est l'ordre du groupe multiplicatif $\mathbb{Z}_N^*$. On appelle e l'*exposant de chiffrement* et d l'*exposant de déchiffrement*. La paire (N, e) est la clé publique, qui permet à chacun de chiffrer un message, et la paire (N, d) est la clé privée, permettant à celui qui la possède de déchiffrer un message chiffré.

Un *message* est un entier $M \in \mathbb{Z}_N^*$. Pour chiffrer le message M, on calcule

$$C = M^e \bmod N \tag{2.1}$$

Ainsi, comme le module N et l'exposant e sont publiques, tout le monde peut chiffrer un message. Pour déchiffrer le cryptogramme C, on calcule:

$$M = C^d \bmod N$$

Seule la personne possédant l'exposant privé de déchiffrement peut donc déchiffrer le message. On retrouve effectivement le message M car:

$$C^d = M^{e \cdot d} = M^{1+k \cdot \phi(N)} = M \bmod N$$

En réalité, avant de chiffrer un message avec RSA, il faut pour des raisons de sécurité que nous verrons dans les prochains chapitres, appliquer au message un format particulier, comme par exemple lui ajouter une partie variable. En effet, sans connaître la clé privée, il doit être non seulement impossible de retrouver le message M à partir du chiffré C, mais aussi d'obtenir une information quelconque sur le message M.

Le cryptosystème RSA s'utilise aussi pour générer des signatures électroniques. Une signature électronique garantit l'authenticité, l'intégrité et la non-répudiation d'un document électronique. Pour signer un message $M \in \mathbb{Z}_N^*$ avec RSA, le signataire utilise sa clé privée (N, d) pour obtenir la signature:

$$S = M^d \bmod N$$

Seul le signataire possédant la clé privée (N, d) est donc capable de signer le message M. On vérifie la validité de la signature S en utilisant la clé publique (N, e), en s'assurant que:

$$M = S^e \bmod N$$

Ainsi, n'importe qui peut vérifier la signature S de M. En réalité, comme pour le chiffrement, il faut pour des raisons de sécurité appliquer au message M à signer un format particulier.

Pour obtenir une paire de clés RSA, on génère aléatoirement deux nombres premiers p et q de taille $n/2$ bits et on les multiplie pour obtenir le module $N = p \cdot q$. Ensuite, étant donné un exposant de chiffrement $e < \phi(N)$, on calcule $d = e^{-1} \bmod \phi(N)$ en utilisant l'algorithme d'Euclide étendu. On génère habituellement un nombre premier aléatoire p de taille $n/2$-bits en générant aléatoirement des entiers de $n/2$-bits et en testant leur primalité à l'aide d'un test probabiliste de primalité [64].

L'algorithme de chiffrement RSA conduit à définir la fonction de chiffrement RSA:

$$f : \mathbb{Z}_N^* \to \mathbb{Z}_N^*, x \to x^e \bmod N$$

Si l'entier d est connu la fonction peut être facilement inversée:

$$f^{-1}(y) = y^d \bmod N$$

Dans ce chapitre et les suivants, nous étudierons le problème consistant à inverser la fonction f sans connaître l'entier d, que nous appelons le problème du cassage de RSA. Plus précisément, étant donné (N, e, y), la question est de savoir quelle est la difficulté de calculer la racine e-ème de y modulo N lorsque la factorisation de N demeure inconnue. On pourrait imaginer d'énumérer tous les éléments $x \in \mathbb{Z}_N^*$ jusqu'à trouver un x tel que $x^e = y \bmod N$, mais le temps de calcul d'une telle méthode serait de l'ordre de N, donc *exponentiel* en la taille $\log_2 N$ du module.

Cela rend la méthode inutilisable pour des modules de taille supérieure à quelque dizaines de bits. On s'intéresse au contraire à des méthodes de calcul dont le temps de calcul est *polynomial* en la taille $\log_2 N$ du module, car dans la pratique de telles méthodes conduisent rapidement au résultat.

2.2 Attaque par factorisation du module N

La première attaque possible contre le cryptosystème RSA consiste à tenter de factoriser le module N pour retrouver p et q, pour être ensuite capable de calculer l'exposant de déchiffrement d. Même si les algorithmes de factorisation d'entiers ont beaucoup progressé durant ces vingt dernières années, on ne connaît pas d'algorithme de factorisation suffisamment efficace pour menacer la sécurité de RSA. En particulier, on ne connaît pas d'algorithme de factorisation dont la complexité soit polynomiale en fonction de la taille du module. L'algorithme de factorisation le plus rapide actuellement est l'algorithme du *crible algébrique*, dont le temps de calcul pour factoriser un module RSA de n bits est donné par:

$$\exp\left((1.923 + \circ(1))n^{1/3}\log^{2/3} n\right)$$

Le record actuel en la matière est la factorisation d'un module RSA de 512 bits [46]. Les auteurs de [46] estiment que la factorisation d'un module de 1024 bits pourrait être réalisée d'ici 2018, compte tenu de l'augmentation des capacités de calcul et de l'amélioration des algorithmes de factorisation.

Dans ce paragraphe, nous décrivons les méthodes de factorisation les plus couramment utilisées. Nous décrivons d'abord les méthodes de factorisation dont le temps de calcul dépend essentiellement de la taille du facteur premier à extraire. Ensuite nous verrons la méthode du crible quadratique et la méthode du crible algébrique, dont les complexités dépendent de la taille de l'entier à factoriser. Dans la pratique, lorsqu'on ne sait rien sur la factorisation d'un entier, on commence par utiliser les méthodes de factorisation dont le temps de calcul dépend de la taille du facteur premier à extraire. Lorsque tous les petits facteurs premiers ont été trouvés, on factorise le reste en utilisant la méthode du crible quadratique ou la méthode du crible algébrique.

2.2.1 Méthodes dont la complexité est fonction de la taille du facteur premier

Soit N l'entier à factoriser. On décrit la méthode par division successive, la méthode ρ de Pollard, la méthode $p-1$ et la méthode des courbes elliptiques de Lenstra.

Division successive : Cette méthode consiste à diviser N par tous les entiers premiers inférieurs à une borne donnée. Pour extraire un facteur premier p, le temps de calcul est $\mathcal{O}(p)$.

Méthode ρ de Pollard [15]: Soit p un facteur premier de N. On définit un polynôme f à coefficients entiers, par exemple $f(x) = x^2 + 1$. On sélectionne aléatoirement un entier $x_0 \in \mathbb{Z}_N$ et on itère la fonction f modulo N (*i.e.* on calcule $x_1 = f(x) \bmod N$, $x_2 = f(f(x)) \bmod N, \ldots, x_i = f^i(x) \bmod N$) jusqu'à trouver une collision modulo p, c'est à dire deux entiers $i \neq j$ tels que $x_i = x_j \bmod p$. Une telle collision est détectée en calculant $\mathrm{PGCD}(x_i - x_j, N)$. Si $\mathrm{PGCD}(x_i - x_j) \neq 1$, alors avec forte probabilité, $p = \mathrm{PGCD}(x_i - x_j, N)$.

La complexité moyenne de la méthode de Pollard pour extraire un facteur p est en $\mathcal{O}(\sqrt{p})$. Dans la pratique, un nombre premier de 60 bits peut être extrait en temps raisonnable (moins de quelques heures sur un PC).

La méthode $p-1$: Si l'entier $p-1$ est B-lisse (*i.e.* si tous les facteurs premiers de $p-1$ sont inférieurs à B), alors $p-1$ divise le produit $\ell(B)$ de tous les entiers inférieurs à B et premiers avec B. Comme $a^{p-1} \bmod p = 1$, on a $a^{\ell(B)} \bmod p = 1$, ce qui permet d'obtenir p avec forte probabilité en calculant:

$$p = \mathrm{PGCD}(a^{\ell(B)} - 1 \bmod N, N)$$

La méthode des courbes elliptiques de Lenstra (ECM) [62]: L'algorithme ECM est une généralisation de la méthode $p-1$. On commence par générer un point P d'une courbe elliptique $\mathcal{E}$ prise aléatoirement modulo N. On calcule ensuite $(B!).P$ pour un petit entier B. Si l'ordre de $\mathcal{E}$ modulo p est B-lisse on a $(B!).P = \mathcal{O}$ sur la courbe elliptique $\mathcal{E}$ modulo p. En calculant $(B!).P$ sur la courbe elliptique modulo N, on obtient alors une inversion impossible à réaliser modulo N, qui permet d'extraire le facteur premier p.

L'algorithme ECM extrait un facteur p de N en temps moyen

$$\exp\left((\sqrt{2} + \circ(1))\sqrt{\log p \log\log p}\right)$$

Dans la pratique, on peut extraire par cette méthode des facteurs premiers de 90 bits en un temps raisonnable (moins de quelques heures sur un PC).

2.2.2 Méthodes dont la complexité est fonction de la taille de l'entier

On décrit dans ce paragraphe la méthode du crible quadratique ainsi que la méthode plus performante du crible algébrique.

Crible quadratique: la méthode du crible quadratique a été développée par Pomerance [74]. Soit N l'entier à factoriser, soit $m = \lfloor\sqrt{N}\rfloor$, et soit le polynôme $q(x) = (x+m)^2 - N$. On a:

$$q(x) = x^2 + 2 \cdot m \cdot x + m^2 - N \simeq x^2 + 2 \cdot m \cdot x$$

donc $q(x)$ est de l'ordre de $\sqrt{N}$ si x est très inférieur à $\sqrt{N}$.

On définit un ensemble de petits facteurs premiers $\mathcal{S} = \{p_1, p_2, \ldots, p_t\}$ appelé base de factorisation. On sélectionne des entiers $a_i = x_i + m$ et on teste si $b_i = q(x_i) = (a_i)^2 - N$ a tous ses facteurs dans $\mathcal{S}$, dans ce cas on a:

$$a_i^2 = b_i = \prod_{j=1}^{t} p_j^{e_{i,j}} \bmod N$$

Aux exposants $e_{i,j}$ on associe les vecteurs binaires

$$v_i = (e_{i,1} \bmod 2, \ldots, e_{i,t} \bmod 2)$$

Si on obtient $t+1$ vecteurs v_i, chaque vecteur ayant t composantes, ils sont linéairement dépendant dans $\mathbb{Z}_2$, et on peut trouver un sous-ensemble non vide $T \subseteq \{1, 2, \ldots, t+1\}$ tel que

$$\sum_{i \in T} v_i = 0 \quad \text{dans } \mathbb{Z}_2$$

Le produit $\prod_{i \in T} b_i$ est alors un carré dans $\mathbb{Z}$. Soit y la racine carrée dans $\mathbb{Z}$ de $\prod_{i \in T} b_i$ et soit $z = \prod_{i \in T} a_i$, on a

$$z^2 = y^2 \bmod N$$

Si $z \neq \pm y \bmod N$, alors $\mathrm{PGCD}(z - y, N)$ donne un facteur premier de N.

L'idée essentielle de la méthode du crible quadratique consiste, au lieu de tester séparément pour chacun des b_i s'il se factorise dans $\mathcal{S}$, à prendre chacun des premiers $p \in \mathcal{S}$ et à examiner la divisibilité par p des entiers $q(x)$ pour un grand nombre de valeurs de x. On remarque pour cela que si $q(x) = 0 \bmod p$, alors $q(x + l \cdot p) = 0 \bmod p$ pour tout entier l. En résolvant l'équation $q(x) = 0 \bmod p$, on obtient une ou deux (en fonction du nombre de solutions) suites de valeurs y telles que p divise $q(y)$.

La méthode du crible quadratique fonctionne alors de la manière suivante. On définit un tableau $Q[]$ indexé par x, dont la x-ème entrée est initialisée à $\lfloor \log |q(x)| \rfloor$. Soient x_1, x_2 les solutions de $q(x) = 0 \bmod p$ pour $p \in \mathcal{S}$. On soustrait alors $\lfloor \log p \rfloor$ aux entrées $Q[x]$ telles que $x = x_1$ ou $x_2 \bmod p$. On répète cela pour tous les premiers $p \in \mathcal{S}$. A la fin, les entrées $Q[x]$ dont la valeur est proche de 0 correspondent à un entier $q(x)$ qui est probablement factorisable dans $\mathcal{S}$, ce qu'on peut vérifier par division successive.

La complexité moyenne en temps de la méthode du crible quadratique, en se basant sur une conjecture sur la distribution des entiers B-lisses, est donnée par:

$$\exp\left((1 + \circ(1))\sqrt{\log N \log\log N}\right)$$

Dans la pratique, on utilise une variante dans laquelle plusieurs polynômes $q(x)$ sont utilisés au lieu d'un seul. Cette variante à polynôme multiple possède la même complexité asymptotique, mais elle a l'avantage d'être facilement parallélisable, chaque processeur utilisant un polynôme distinct.

Crible algébrique: la méthode du crible algébrique a été développée par Pollard [73] en 1988. Elle est assez similaire à la méthode du crible quadratique: on cherche des entiers x et y tels que $x^2 = y^2 \bmod N$ et $x \neq \pm y \bmod N$. Pour cela, deux bases de factorisation sont utilisées, la première constituée de tous les nombres premiers inférieurs à une certaine borne, la deuxième constituée de tous les idéaux premiers de norme inférieure à une certaine borne dans l'anneau des entiers d'un corps de nombre bien choisi. La complexité heuristique de la méthode du crible algébrique est donnée par:

$$\exp\left((C + \circ(1)) \cdot (\log N)^{1/3} (\log\log N)^{2/3}\right)$$

où $C = (64/9)^{1/3} \simeq 1.923$.

2.3 Attaques élémentaires

Dans ce paragraphe, on décrit deux attaques élémentaires qui illustrent le danger d'une utilisation incorrecte de RSA.

2.3.1 Forge existentielle

Etant donné un module RSA N, si la signature d'un message M est donnée par

$$S = M^d \bmod N$$

alors on voit que pour tout entier $r \in \mathbb{Z}_N$, la signature du message $M = r^e \bmod N$ est donnée par r. Il est donc possible de produire une signature valide d'un message sans connaître l'exposant privé d : c'est une forge existentielle. Dans la pratique, pour éviter cette attaque, on ajoute une redondance au message M pour qu'il soit difficile de trouver un entier r tel que $r^e \bmod N$ respecte cette redondance. On peut aussi utiliser une fonction de hachage. On applique alors la signature RSA au haché du message.

2.3.2 Attaque par masquage

Soit (N, d) la clé privée d'Alice correspondant à la clé publique (N, e). Supposons qu'un attaquant dénommé Marvin désire faire signer le message M à Alice. En lisant le message M, Alice refuse de le signer. Marvin génère alors un entier aléatoire $r \in \mathbb{Z}_N$ et calcule $M' = M \cdot r^e \bmod N$. Il demande ensuite à Alice de signer le message aléatoire M'. Si Alice accepte, Marvin obtient:

$$S' = M'^d \bmod N = r \cdot M^d \bmod N$$

et peut donc calculer la signature $S = M^d = S'/r \bmod N$ du message M.

On a ainsi décrit une *attaque à message choisi* sur l'algorithme de signature RSA: l'attaquant Marvin a demandé à Alice de signer un message M' qu'il a choisi, ce

qui lui permet d'obtenir la signature d'un autre message M. Dans la pratique, on évite cette attaque en appliquant une redondance au message M, de sorte qu'il soit difficile de trouver un entier r tel que $M' = M \cdot r^e \bmod N$ respecte cette redondance.

Il est facile de transposer l'attaque précédente dans le cas du chiffrement RSA. Un attaquant voulant obtenir le déchiffrement $M = C^d \bmod N$ d'un chiffré C peut choisir un entier aléatoire s et demander le déchiffrement de $C' = s^e \cdot C \bmod N$. Avec la réponse $M' = (C')^d \bmod N$, il obtient facilement le message original $M = M' \cdot s^{-1} \bmod N$. Pour pallier cette attaque, on ajoute de la redondance au message M à chiffrer, pour qu'il soit difficile à un attaquant de trouver un entier s tel que le chiffré C' corresponde à un clair M' qui respecte cette redondance.

L'attaque précédente peut s'étendre au cas où l'attaquant n'obtient que le bit de poids faible du message clair [53]. Il existe en effet un algorithme permettant de déchiffrer tout message, en utilisant un oracle qui donne le bit le moins significatif du message clair correspondant au chiffré.

2.4 Attaques multiplicatives

2.4.1 Attaque de Desmedt-Odlyzko

L'attaque de Desmedt-Odlyzko [38] est une généralisation de l'attaque par masquage décrite précédemment. Il s'agit aussi d'une attaque à chiffré choisi sur le chiffrement RSA, ou de manière équivalente une attaque à message choisi sur la signature RSA. Dans l'attaque par masquage, l'attaquant possède d'abord un chiffré C et demande ensuite le déchiffrement d'un autre message pour retrouver le clair correspondant à C. Au contraire, dans l'attaque de Desmedt-Odlyzko, l'attaquant collecte d'abord le déchiffrement d'un grand nombre de messages chiffrés, et ensuite il peut décrypter n'importe quel chiffré.

Dans un premier temps, l'attaquant obtient le déchiffrement de $L_N[\alpha]$ chiffrés choisis, pour un réel α donné, avec

$$L_N[\alpha] = \exp(\alpha\sqrt{\log N \log\log N})$$

Ensuite, étant donné un chiffré C, l'attaquant retrouve le message clair en

$$L_N[1/(4 \cdot \alpha) + \circ(1)]$$

opérations si $\alpha \geq 1/2$, et en

$$L_N[1/(2 \cdot \alpha) - \alpha + \circ(1)]$$

opérations si $0 < \alpha \leq 1/2$. En prenant $\alpha = 1/2$, la complexité totale de l'attaque est donc

$$L_N[1/2 + \circ(1)]$$

Lors de la publication de l'article en 1985, le meilleur algorithme de factorisation connu, l'algorithme de crible quadratique, nécessitait $L_N[1 + \circ(1)]$ opérations. La complexité asymptotique de l'attaque de Desmedt-Odlyzko était donc meilleure. Toutefois, l'attaque n'est pas vraiment pratique car pour des modules de taille raisonnable elle nécessite le déchiffrement d'un très grand nombre de messages.

L'attaque de Desmedt-Odlyzko comprend les étapes suivantes:

1. Soient $\alpha > 0$ et $k = \lfloor \sqrt{N} \rfloor$.
2. Soient $S_1 = \{p | p \leq L_N[\alpha] \text{ et } p \text{ premier}\}$, $S_2 = \{k+1, k+2, \ldots, k + \lfloor L_N[\alpha] \rfloor\}$ et $S = S_1 \cup S_2$.
3. Obtenir le déchiffrement $x^d \bmod N$ de tous les éléments x de S.
4. Soit C le chiffré dont on veut obtenir le clair $M = C^d \bmod N$.
5. Choisir un entier y aléatoirement modulo N et calculer $b = C \cdot y^{-e} \bmod N$.
6. S'il existe un facteur premier q_i de b supérieur à $L_N[2\alpha]$, revenir à l'étape précédente.
7. Pour chacun des facteurs premiers $q = q_i$ de b, calculer

$$m = \left\lfloor \frac{\sqrt{N}}{q} \right\rfloor$$

   et pour $\delta > 0$ et $\beta = 1/(4\alpha) + \delta$ si $\alpha > 1/2$ et $\beta = 1/(2 \cdot \alpha) - \alpha + \delta$ si $\alpha \leq 1/2$, déterminer les entiers parmi

$$m + 1, m + 2, \ldots, m + \lfloor L_N[\beta] \rfloor$$

   dont tous les facteurs premiers sont inférieurs à $L_N[\alpha]$.
8. Pour ces entiers et pour les entiers i positifs tels que $i \leq L_N[1/(4\alpha)]$ si $\alpha > 1/2$ et $i \leq L_N[\alpha]$ si $\alpha \leq 1/2$, chercher un entier t

$$t = q \cdot (m + j) \cdot (k + i) \bmod N$$

   tel que les facteurs premiers de t soient inférieurs à $L_N[\alpha]$.
9. On obtient ainsi pour tous les facteurs premiers $q = q_i$ de b:

$$q = \prod_{x \in S} x^{u_x} \bmod N$$

10. L'entier b peut donc se factoriser dans S et on peut écrire:

$$C = y^e \cdot \prod_{x \in S} x^{a_x} \bmod N$$

11. L'attaquant retrouve donc le message M à partir des $x^d \bmod N$:

$$M = C^d = y \cdot \prod_{x \in S} (x^d)^{a_x} \bmod N$$

2.4.2 Attaque de Kaliski-Robshaw

L'attaque de Kaliski-Robshaw [59] est une variante de l'attaque précédente. Elle s'applique au cas où le message à déchiffrer est le produit de petits facteurs premiers pour le chiffrement RSA, et au cas où le message à signer est le produit de petits facteurs premiers pour la signature RSA.

1. Soit B un entier positif et $E = \{p : p \text{ premier tel que } p \leq B\}$.
2. Obtenir le déchiffrement (ou de manière équivalente la signature) de plusieurs messages dont tous les facteurs premiers sont inférieurs à B.
3. Par combinaison multiplicative, obtenir le déchiffrement (ou la signature) de tous les éléments de E.
4. L'attaquant peut ensuite déchiffrer (ou signer) tout message dont les facteurs premiers sont tous inférieurs à B.

Nous reviendrons en détail sur cette attaque dans le chapitre consacré aux cryptanalyses de schémas de signature RSA.

2.5 Attaque avec exposant privé petit

Pour réduire le temps nécessaire au déchiffrement d'un message ou à la génération d'une signature, on peut être tenté d'utiliser un exposant d petit. En effet, la durée d'une exponentiation étant proportionnelle à $\log_2 d$, si on prend un exposant d de 100 bits pour un module RSA de 1024 bits, on gagnera pratiquement un facteur 10 en temps de calcul. Malheureusement, on ne peut pas utiliser d'exposant trop petit, comme le montre le théorème suivant dû à Wiener [82].

Théorème 2.5.1. *Soit $N = p \cdot q$ avec $q < p < 2 \cdot q$. Soit $d \leq \frac{1}{3}N^{1/4}$. Etant donné (N, e) avec $e \cdot d = 1 \bmod \phi(N)$, il existe un algorithme polynomial permettant de retrouver d.*

Il faut donc que la taille de d soit supérieure au quart de la taille de N. Pour un module de 1024 bits, d doit donc être de taille supérieure à 256 bits. Le théorème précédent a été récemment amélioré par Boneh et Durfee [13]: si $d < N^{0.292}$, il est possible de retrouver d de manière efficace à partir de (N, e).

Dans la pratique, pour obtenir de meilleures performances, le déchiffrement RSA est réalisé en utilisant le théorème du reste Chinois (CRT). Pour calculer $M = C^d \bmod N$, on calcule séparément le message M modulo p et modulo q, en calculant $M_p = C^{d_p} \bmod p$ et $M_q = C^{d_q} \bmod q$, avec $d_p = d \bmod p - 1$ et $d_q = d \bmod q - 1$. En utilisant le théorème du reste Chinois, on obtient l'unique message $M \in \mathbb{Z}_N$ tel que $M = M_p \bmod p$ et $M = M_q \bmod q$. Une exponentiation modulaire ayant une complexité cubique en la taille du module, chacune des deux exponentiations est 8 fois plus rapide que l'exponentiation $M = C^d \bmod N$. Il en résulte un gain en temps de calcul d'un facteur 4.

Une autre approche pour réduire le temps d'exécution en utilisant la méthode du CRT consiste à choisir d de sorte que $d_p = d \bmod p - 1$ et $d_q = d \bmod q - 1$ soient petits, avec un exposant d de la même taille que N. Dans ce cas, les attaques de Wiener et de Boneh et Durfee ne s'appliquent plus. La meilleure attaque connue [45] a une complexité en temps de $\mathcal{O}(\min(\sqrt{d_p}, \sqrt{d_q}))$. On peut donc prendre par exemple d_p et d_q de taille 128 bits. Dans le cas de RSA avec un module de 1024 bits, on obtient ainsi un gain en temps d'un facteur 4 par rapport à un déchiffrement CRT classique.

2.6 Conclusion

Nous avons décrit le cryptosystème RSA ainsi que les principales attaques connues. Ces attaques ne mettent pas en cause la sécurité de RSA mais illustrent les dangers d'une utilisation incorrecte de RSA. Pour pallier les attaques multiplicatives sur RSA, on applique de la redondance au message à chiffrer ou à signer. Nous verrons cependant dans les chapitres suivants que des attaques plus sophistiquées permettent de passer outre certaines protections.

Partie II

Cryptanalyse de schémas de chiffrement basés sur RSA

3. Attaques sur le chiffrement RSA

Dans ce chapitre, on étudie les principales attaques connues sur le chiffrement RSA. On commence par rappeler les différents modèles d'attaque, qui rendent compte à la fois de l'objectif de l'attaquant et des ressources dont il dispose.

3.1 Modèle d'attaque

L'objectif de l'attaquant n'est pas nécessairement de retrouver la clef privée associée à la clef publique; il s'agit de l'objectif le plus ambitieux, et on parle dans ce cas de *cassage total*. Le plus souvent, l'objectif de l'attaquant est de pouvoir retrouver le message clair associé à un message chiffré donné. L'attaquant peut aussi poursuivre un objectif plus modeste: obtenir seulement la moitié du texte clair, ou même seulement un bit du message. Le rôle d'un schéma de chiffrement étant de préserver la confidentialité d'un message, un attaquant ne devrait pas pouvoir obtenir une quelconque information sur le message clair à partir du chiffré (en dehors de la taille du message clair). Cette propriété est formalisée par la notion de sécurité sémantique [51] et de non-malléabilité [41].

On dit qu'un schéma de chiffrement à clé publique est *sémantiquement sûr* s'il est impossible de distinguer le chiffrement de deux messages. Plus précisément, on considère un attaquant qui, après avoir reçu la clef publique, renvoie deux messages m_0 et m_1 de même taille. On chiffre l'un des deux messages et on envoie le chiffré y à l'attaquant. Ce dernier doit déterminer de quel message (m_0 ou m_1) y est le chiffré, avec probabilité significativement supérieure à $1/2$. Le schéma de chiffrement est sémantiquement sûr s'il n'existe pas de tel attaquant (voir [6] pour une définition précise).

On dit qu'un schéma de chiffrement à clé publique est *non-malléable* s'il est impossible, étant donné un chiffré y, d'obtenir des chiffrés distincts de y dont les clairs correspondants soient reliés à celui de y de façon significative (voir [6] pour une définition précise).

Les ressources de l'attaquant dépendent des informations auxquelles il a accès. En cryptographie à clef publique, l'attaquant peut toujours chiffrer n'importe quel message. On parle alors d'*attaque à clair choisi*, qui correspond aux ressources les plus faibles. Dans une *attaque à chiffré choisi non-adaptative*, l'attaquant peut obtenir le déchiffrement de tout message, mais seulement avant d'avoir reçu le chiffré y.

Finalement, dans une *attaque à chiffré choisi adaptative*, l'attaquant peut obtenir tout au long de l'attaque le déchiffrement de tout message excepté y.

Le modèle de sécurité le plus fort est celui dans lequel l'objectif de l'attaquant est le plus modeste (obtenir une quelconque information sur le clair) alors qu'il dispose du maximum de ressources, c'est à dire qu'il peut réaliser une attaque à chiffré choisi adaptative. Les relations d'implication entre les diverses notions de sécurité possibles sont étudiées dans [6]. On montre en particulier que dans le cadre d'une attaque à chiffré choisi adaptative, la sécurité sémantique est équivalente à la non-malléabilité.

3.2 Attaques avec exposant public e petit

L'utilisation d'un exposant de chiffrement petit permet d'accélérer le chiffrement ou la vérification de signature. Il est possible de prendre $e = 3$, mais il est recommandé pour éviter certaines attaques de prendre $e = 2^{16} + 1 = 65537$. Dans ce cas, avec un module RSA de 1024 bits, on obtient un gain en temps proche d'un facteur 50 par rapport à un exposant e de pleine taille. Contrairement aux attaques contre les exposants de déchiffrement petits décrites dans le chapitre précédent, les attaques avec e petit ne conduisent pas à un cassage total de RSA.

3.2.1 Le théorème de Coppersmith

Les attaques sur RSA avec exposant petit sont pour la plupart basées sur un théorème très important dû à Coppersmith [19].

Théorème 3.2.1 (Coppersmith). *Soit $p(x)$ un polynôme de degré δ à une seule variable. Soit N un entier dont la factorisation est inconnue. Soit X une borne sur la solution désirée. Si $X < \frac{1}{2} \cdot N^{1/\delta - \epsilon}$, alors en temps polynomial en $(\log N, \delta, 1/\epsilon)$, on peut trouver tous les entiers x_0 tels que $p(x_0) = 0 \bmod N$ et $|x_0| < X$.*

Corollaire 3.2.1 (Coppersmith). *Sous les mêmes hypothèses, excepté que $X \leq N^{1/\delta}$, en temps polynomial en $(\log N, 2^\delta)$, on peut trouver tous les entiers x_0 tels que $p(x_0) = 0 \bmod N$ et $|x_0| \leq X$.*

Le théorème précédent fournit un algorithme permettant de trouver efficacement toutes les racines d'un polynôme f modulo N de degré δ qui sont inférieures à $N^{1/\delta}$. La méthode est basée sur l'utilisation de l'algorithme de réduction de réseau LLL (du nom de ses inventeurs L. Lovász, A. Lenstra, et H. Lenstra Jr.) [61]. Cet algorithme comprend de nombreuses applications en théorie des nombres et en cryptanalyse (voir [44]).

Le théorème de Coppersmith s'applique en particulier au cas où le message clair M consiste en une partie connue $B = 2^k \cdot b$ et d'une partie inconnue x. Le chiffré est alors $C = M^e = (B + x)^e \bmod n$. En utilisant le théorème précédent avec le polynôme $p(x) = (B + x)^e - C$, on retrouve x à partir de C si $|x| < N^{1/e}$. Dans le cas où $e = 3$, on peut donc retrouver le message clair si on en connaît déjà les deux-tiers.

3.2.2 L'attaque de Håstad

L'attaque de Håstad [56] est une attaque qui s'applique au cas où un même message, ou plus généralement des messages liés entre eux par une relation connue, sont chiffrés et envoyés à différentes personnes.

Pour illustrer l'attaque sur un cas très simple, supposons d'abord que Bob utilise le même exposant de chiffrement pour envoyer à k personnes différentes le même message m. Chaque personne P_i possède sa propre clef publique (N_i, e). On suppose que le message M est inférieur à tous les N_i. Pour envoyer M à la i-ème personne P_i, Bob chiffre naïvement le message M avec $C_i = M^e \bmod N_i$. L'attaquant Marvin peut espionner la communication et obtenir chacun des k messages chiffrés.

Pour simplifier, on peut supposer que l'exposant public e est égal à 3. On voit facilement que dans ce cas, Marvin peut obtenir M si le nombre de chiffrés k est supérieur ou égal à 3. En effet, supposons que Marvin obtienne C_1, C_2, C_3, où:

$$C_1 = M^3 \bmod N_1 \quad C_2 = M^3 \bmod N_2 \quad C_3 = M^3 \bmod N_3$$

On peut supposer que $\mathrm{PGCD}(N_i, N_j) = 1$ pour tout $i \neq j$, car sinon, Marvin pourrait factoriser au moins deux modules N_i. En appliquant le théorème du reste Chinois (CRT) à C_1, C_2, C_3, on obtient un entier C' compris entre 0 et $N_1 \cdot N_2 \cdot N_3$ tel que $C' = M^3 \bmod N_1 \cdot N_2 \cdot N_3$. Comme le message M est inférieur à chacun des modules N_i, on a $M^3 < N_1 \cdot N_2 \cdot N_3$. L'égalité $C' = M^3$ se vérifie par conséquent sur les entiers, et Marvin retrouve M en calculant (dans $\mathbb{Z}$) la racine cubique de C'. On voit que l'attaque se généralise à tout exposant de chiffrement e, à condition que le nombre de chiffrés k soit supérieur ou égal à e. L'attaque n'est donc réalisable que pour e petit.

Pour se prémunir contre ce type d'attaque, on pourrait imaginer qu'avant de chiffrer M, Bob ajoute une information de temps au message M. Par exemple, si le message M a une taille de m bits, Bob pourrait envoyer le chiffré de $M_i = i \cdot 2^m + M$ à la personne P_i. L'attaque précédente ne s'applique plus puisque dans ce cas les messages sont différents. Malheureusement, Håstad a montré dans [56] que ce type de protection n'est pas sûre. L'attaque de Håstad est une généralisation de l'attaque précédente, et montre que même en appliquant un polynôme donné au message avant de le chiffrer, on peut retrouver le message.

Supposons que pour chacune des personnes $P_1, \ldots, P_k$, Bob ait déterminé un polynôme constant et public $f_i \in \mathbb{Z}_{N_i}[x]$. Pour diffuser le message M, Bob envoie le chiffré de $f_i(M)$ à la personne P_i. Marvin, en espionnant la communication, obtient les chiffrés $C_i = f_i(M)^{e_i} \bmod N_i$ pour $i = 1, \ldots, k$. L'attaque de Håstad permet de retrouver M à partir des chiffrés C_i s'il y en a suffisamment. Le théorème suivant est une version améliorée par D. Boneh [12] du théorème de Håstad:

Théorème 3.2.2. *Soient $N_1, \ldots, N_k$ des entiers premiers entre eux deux à deux. Soit $N_{min} = \min(N_i)$. Soient les k polynômes $g_i \in \mathbb{Z}_{N_i}[x]$ de degré maximum δ. On suppose qu'il existe un unique $M < N_{min}$ satisfaisant:*

$$g_i(M) = 0 \bmod N_i \quad \textit{pour tout } i = 1, \ldots, k$$

Alors si $k \geq \delta$, on peut retrouver efficacement M à partir des N_i et des polynômes g_i.

Preuve. Soit $\bar{N} = N_1 \cdot N_2 \cdots N_k$. On peut supposer que chacun des polynômes g_i est unitaire. En effet, si le coefficient de plus haut degré de g_i n'est pas inversible dans $\mathbb{Z}_{N_i}^*$, on peut factoriser N_i. En multipliant chacun des g_i par une puissance de x, on peut supposer aussi qu'ils sont tous de degré δ. On construit le polynôme:

$$g(x) = \sum_{i=1}^{k} T_i \cdot g_i(x), \quad \text{où } T_i = \begin{cases} 1 \bmod N_j & \text{si } i = j \\ 0 \bmod N_j & \text{si } i \neq j \end{cases}$$

Les entiers T_i sont les coefficients du reste Chinois. Le polynôme g est unitaire car les polynômes g_i le sont. Le polynôme g est de degré δ, et on a $g(M) = 0 \bmod \bar{N}$. Comme $M < N_{min} \leq \bar{N}^{1/k} \leq \bar{N}^{1/\delta}$, en appliquant le théorème 3.2.1, on retrouve M. □

Le théorème précédent montre donc que l'on sait résoudre un système d'équations univariées modulo un entier composite dont la factorisation est inconnue, à condition que le nombre d'équations fournies soit supérieur ou égal au degré maximal des polynômes. En prenant $g_i = (f_i)^{e_i} - C_i \bmod N_i$, on voit que Marvin peut retrouver M à partir des chiffrés si le nombre de personnes est supérieur au maximum des $e_i \cdot \deg(f_i)$. En particulier, si tous les e_i sont égaux à e et si Bob envoie des messages liés linéairement entre eux, Marvin retrouve le message clair M dès que le nombre de destinataires k est supérieur ou égal à e.

3.2.3 L'attaque de Franklin-Reiter sur les messages liés

L'attaque de Franklin-Reiter sur les messages liés avec un exposant RSA petit consiste en la méthode suivante [20]: supposons que deux messages m_1, m_2 vérifient une relation polynomiale connue p de la forme

$$m_2 = p(m_1), \qquad \deg(p) = \delta \ ,$$

et que les deux chiffrés correspondant c_1 et c_2 sont connus. Dans ce cas $z = m_1 \bmod N$ est une racine commune des deux équations polynomiales:

$$\begin{aligned} z^e - c_1 &= 0 \bmod N \\ (p(z))^e - c_2 &= 0 \bmod N \ , \end{aligned}$$

de sorte qu'avec forte probabilité on retrouve m_1 avec :

$$\mathrm{PGCD}(z^e - c_1, (p(z))^e - c_2) = z - m_1 \bmod N \ .$$

La complexité de l'attaque est quadratique en l'exposant e. Elle ne s'applique donc que lorsqu'un petit exposant e est utilisé.

3.2.4 L'attaque de Coppersmith sur les redondances courtes

Cette attaque s'applique au cas suivant: supposons qu'avant de chiffrer un message m on lui concatène une chaîne de bits aléatoires, de sorte que $M = m + r$ avec r un entier aléatoire. Supposons que Bob utilise cette méthode pour chiffrer le message m à Alice. Il calcule donc $M_1 = m + r_1$ et envoie $C_1 = (M_1)^e \bmod N$ à Alice. Supposons que Marvin intercepte le message chiffré C_1 pour l'empêcher d'atteindre sa destination. Alice n'ayant pas reçu C_1, elle demande à Bob de lui chiffrer une nouvelle fois m. Bob calcule donc $M_2 = m + r_2$ et lui envoie le chiffré C_2 correspondant. Marvin obtient donc deux chiffrés C_1 et C_2 du même message m. Avec la méthode suivante, Marvin retrouve m, à condition que la taille des aléas r_1 et r_2 soit inférieure à la taille de N divisée par e^2.

On définit les deux polynômes:

$$g_1(x, y) = x^e - C_1 \quad \text{et} \quad g_2(x, y) = (x + y)^e - c_2$$

On sait que lorsque $y = r_2 - r_1$, les polynômes g_1 et g_2 modulo N ont M_1 pour racine commune. L'entier $\Delta = r_2 - r_1$ est donc une racine du résultant $h(y) = \mathrm{res}_x(g_1, g_2)$ des polynômes g_1 et g_2. Le degré du polynôme h est au plus e^2. Donc si $r_1, r_2 < N^{1/e^2}$, on a $|\Delta| < N^{1/e^2}$, et Δ est une petite racine du polynôme h modulo N, que l'on peut retrouver efficacement en utilisant le théorème précédent de Coppersmith. Lorsque Δ est connu, l'attaque précédente de Franklin-Reiter permet de retrouver M_1 et donc m.

3.3 Conclusion

Nous avons décrit les principales attaques connues sur le chiffrement RSA. En pratique, ces attaques permettent à un attaquant de déchiffrer un texte chiffré sans factoriser le module et avec un coût calculatoire modéré. L'outil cryptanalytique le plus performant dans le cas d'exposants de chiffrement petits est celui basé sur le théorème de Coppersmith.

Pour rendre le chiffrement RSA résistant contre ces attaques, on peut imposer un format spécial au message. Nous verrons cependant dans le chapitre suivant que même avec un format spécial comme celui du standard PKCS#1, le chiffrement RSA peut se révéler vulnérable.

4. Attaques contre PKCS#1 v1.5 chiffrement

Lors de la conférence Crypto '98, Daniel Bleichenbacher a décrit dans [11] une attaque à chiffré choisi sur le schéma de chiffrement RSA PKCS#1 v1.5 [79]. L'attaque a ceci de particulier qu'elle ne requiert pas l'obtention des déchiffrés des messages, mais seulement de savoir si le déchiffré est conforme ou non au format PKCS#1 v1.5. Comme certaines applications vérifiaient qu'un chiffré était PKCS#1 v1.5-conforme et renvoyaient un message d'erreur dans le cas contraire, l'attaque était applicable, et l'attaquant pouvait alors déchiffrer les messages de son choix.

Lors de la conférence Eurocrypt 2000, nous avons présenté dans [28] deux nouvelles attaques contre le standard de chiffrement PKCS#1 v1.5. Contrairement à l'attaque de Bleichenbacher, ces attaques sont à clair choisi seulement, c'est à dire qu'elles ne nécessitent pas l'accès à un oracle de déchiffrement. La première attaque s'applique au cas d'un petit exposant de chiffrement. On montre que si un texte clair se termine avec suffisamment de zéros, il existe une méthode relativement efficace permettant de le retrouver à partir de plusieurs chiffrés de ce même texte clair. La seconde attaque s'applique quel que soit l'exposant public, à condition que la plupart des bits du message clair soient égaux à 0.

4.1 Introduction

PKCS, qui signifie *Public-Key Cryptography Standards* est un *corpus* de specifications couvrant le chiffrement RSA, l'échange de clef à la Diffie-Hellman, le chiffrement basé sur l'utilisation de mots de passe, etc. Historiquement, PKCS fut développé par les Laboratoires RSA, Apple, Digital, Lotus, Microsoft, MIT, Northern Telecom, Novell et Sun.

Dans la collection PKCS, PKCS#1 v1.5 décrit une méthode particulière de chiffrement RSA appelée `rsaEncryption`. Les données traitées par `rsaEncryption` sont d'abord chiffrées en chiffrement symétrique avec une clé K choisie aléatoirement, puis K est chiffrée avec RSA en utilisant la clef publique du destinataire.

L'attaque de Bleichenbacher est une attaque à chiffré choisi adaptative contre PKCS#1 v1.5. Elle permet de retrouver un texte clair arbitraire à partir du déchiffrement d'une centaine de milliers de textes chiffrés. Bien que les modèles d'attaque active soient généralement considérés comme d'un intérêt théoriques (les attaques à chiffré choisi présupposent que l'attaquant a accès à un oracle de

déchiffrement), l'attaque de Bleichenbacher utilise un oracle qui détecte seulement la conformité du chiffré avec le schéma de padding PKCS#1 v1.5. Autrement dit, l'oracle répond "oui" si au chiffré c correspond un clair m dont le format est conforme à PKCS#1 v1.5, et "non" dans le cas contraire. C'est une hypothèse réaliste dans la pratique: de nombreux serveurs vérifiaient en effet qu'un chiffré était PKCS#1 v1.5-conforme, et renvoyaient un message d'erreur dans le cas contraire. En conséquence, le standard PKCS#1 v1.5 a été remplacé dans la version 2.0 [77]. Dans le nouveau standard, la méthode de chiffrement utilisée est OAEP, développé par Bellare et Rogaway dans [8]. Le schéma OAEP est sémantiquement sûr dans le modèle de l'oracle aléatoire contre les attaques à chiffré choisi adaptatives.

Dans ce chapitre, nous montrons qu'une attaque à message clair choisi suffit en fait à casser le standard PKCS#1 v1.5. La technique mise en oeuvre permet à un attaquant de retrouver efficacement le texte clair à condition qu'il se termine par un nombre suffisant de zéros. Ces attaques ne requièrent qu'un nombre limité de chiffrés (typiquement moins de dix) de ce même texte clair.

4.2 Le standard de chiffrement PKCS#1 v1.5

Soient N un module RSA, e un exposant de chiffrement et d l'exposant de déchiffrement correspondant. On note par la suite par k la longueur en octets du module N, et on a donc:

$$2^{8(k-1)} \leq N < 2^{8k}$$

Un message m de taille $|m|$ octets avec $|m| \leq k - 11$ est chiffré de la manière suivante [79]: un entier r' constitué de $k-3-|m| \geq 8$ octets tous non nuls est généré aléatoirement. A partir du message m et de l'entier r', on définit:

$$\text{PKCS}(m, r') = 0002_{16} \| r' \| 00_{16} \| m \ ,$$

où $||$ note la concaténation et on obtient le chiffré c en calculant:

$$c = \text{PKCS}(m, r')^e \bmod N \ .$$

4.3 L'attaque de Bleichenbacher

L'attaque de Bleichenbacher est une attaque à chiffré choisi adaptative, permettant de déchiffrer tout message chiffré c. Lors d'une attaque à chiffré choisi, l'attaquant peut obtenir le déchiffrement de tout message chiffré, excepté celui de c. L'attaque est dite adaptative si l'attaquant peut choisir les chiffrés en fonction des réponses qu'il a obtenues précédemment. Nous avons vu dans le chapitre consacré à RSA que le chiffrement simple avec RSA était vulnérable contre les attaques à chiffré choisi, même si l'attaquant n'obtient qu'un seul bit du message clair correspondant

au chiffré qu'il soumet. L'attaque de Bleichenbacher utilise un résultat similaire: on suppose que l'attaquant a accès à un oracle qui, pour tout chiffré, répond si le clair correspondant est conforme ou non au format PKCS#1 v1.5. On montre alors que l'on peut utiliser cet oracle pour calculer $c^d \bmod N$ pour tout entier c déterminé à l'avance. Autrement dit, l'attaquant peut déchiffrer n'importe quel message. De plus, si la même clé publique est utilisée pour signer, l'attaquant peut par cette méthode obtenir la signature de tout message.

Soit c le chiffré dont l'attaquant veut obtenir le clair $m = c^d \bmod N$. L'attaquant choisit un entier s, calcule

$$c' = c \cdot s^e \bmod N$$

et envoie c' à l'oracle. Si l'oracle répond que c' est un chiffré PKCS#1 v1.5-conforme, alors l'attaquant sait que les deux premiers octets de $m \cdot s$ sont 00 et 02. Si on note

$$B = 2^{8(k-2)}$$

alors le fait que $m \cdot s$ soit PKCS#1 v1.5-conforme implique que:

$$2B \leq m \cdot s \bmod N < 3B$$

En utilisant cette information pour de nombreux entiers s distincts, l'attaque décrite par Bleichenbacher permet de retrouver m, au bout d'environ un million d'appels à l'oracle.

4.4 Nouvelle attaque à clair choisi pour *e* petit

Dans ce paragraphe, nous décrivons une nouvelle attaque contre le standard de chiffrement RSA PKCS#1 v1.5. Contrairement à l'attaque de Bleichenbacher précédente, elle ne nécessite pas l'accès à un oracle de déchiffrement. En revanche, l'attaquant a besoin de deux chiffrés (ou plus) du même message. C'est une hypothèse réaliste: par exemple, si Bob souhaite envoyer un message chiffré à Alice, l'attaquant Marvin peut intercepter le message, et Alice qui n'a rien reçu demande à Bob de lui rechiffrer le message. L'attaquant intercepte une nouvelle fois le chiffré et obtient ainsi deux chiffrés du même message.

En notant comme précédemment $|m|$ la taille du message en octets, et en notant:

$$r = 0002_{16} || r'$$

on peut écrire:

$$\text{PKCS}(m, r') = r \cdot 2^{\beta} + m \quad \text{avec } \beta = 8|m| + 8$$

Par la suite on suppose que les Z derniers bits les moins significatifs de m sont égaux à zéro. On peut donc écrire le message m sous la forme:

$$m = \bar{m} \cdot 2^Z$$

En notant M la taille du message $\bar{m}$ en bits, on a $8|m| = M + Z$ et ainsi:

$$\text{PKCS}(m, r') = 2^Z(r2^{\beta-Z} + \bar{m}) \ .$$

A partir de deux chiffrés du même message m:

$$\begin{aligned} c_1 &= [2^Z(r_1 2^{\beta-Z} + \bar{m})]^e \bmod N \\ c_2 &= [2^Z(r_2 2^{\beta-Z} + \bar{m})]^e \bmod N \end{aligned}$$

l'attaquant calcule :

$$\Delta = \frac{c_1 - c_2}{2^{eZ}2^{\beta-Z}} \bmod N \tag{4.1}$$

$$= \underbrace{(r_1 - r_2)}_{=\omega} \underbrace{\Big[\sum_{j=0}^{e-1}(r_1 2^{\beta-Z} + \bar{m})^{e-1-j}(r_2 2^{\beta-Z} + \bar{m})^j\Big]}_{=v} \bmod N \tag{4.2}$$

L'attaque se déroule alors de la façon suivante : supposons que $r_1 > r_2$ et que le nombre de zéros Z soit suffisamment grand pour que $0 < \omega \cdot v < N$, alors la relation (4.2) se vérifie sur les entiers, et donc $\omega = r_1 - r_2$ doit diviser Δ.

Par conséquent, en extrayant les petits facteurs premiers de Δ, on peut espérer pouvoir reconstruire un candidat possible pour ω. La connaissance de ω permet ensuite de retrouver le message m en utilisant l'attaque à petit exposant de chiffrement avec messages reliés sur RSA décrite dans le chapitre précédent (attaque de Franklin-Reiter).

En notant R la taille de l'aléa en bits (le standard impose $R \geq 64$), et $|N|$ la taille du module en bits, la condition $\omega \cdot v < N$ est satisfaite si :

$$e \cdot R + (e-1) \cdot (M + 10) < |N| \ . \tag{4.3}$$

4.4.1 Isoler les facteurs de Δ inférieurs à une borne B

La première étape de l'attaque consiste à obtenir un ensemble $\mathcal{D}$ de diviseurs de Δ, en espérant que ω figure dans cet ensemble. Pour cela, on extrait les facteurs premiers $\mathcal{P} = \{p_1, \ldots, p_i\}$ de Δ inférieurs à une borne B. L'ensemble $\mathcal{D}$ est obtenu en prenant toutes les combinaisons multiplicatives possibles des facteurs premiers de $\mathcal{P}$, dont la taille en bits soit inférieure à R (la taille de ω). Si tous les facteurs premiers de ω sont inférieurs à B (dans ce cas, ω est dit B-lisse), alors on sait que $\omega \in \mathcal{D}$.

Comme seule une factorisation partielle de Δ est requise, on utilisera seulement des méthodes de factorisation dont la complexité $C(p)$ ne dépend que de la taille des facteurs premiers. Ces méthodes ont été décrites dans un chapitre précédent et

comprennent la méthode par divisions successives, qui donne une complexité en $\mathcal{O}(p)$ pour extraire un facteur premier p de Δ, la méthode de Pollard dont la complexité pour extraire p est $\mathcal{O}(\sqrt{p})$, et la méthode des courbes elliptiques de Lenstra (ECM), qui extrait un facteur p en temps moyen

$$\exp\left((\sqrt{2}+\circ(1))\sqrt{\log p \log\log p}\right)$$

On note traditionnellement par $\psi(x,y)$ le nombre d'entiers positifs $z \leq x$ tels que z est y-lisse, c'est à dire que tous les facteurs premiers de z sont inférieurs à y. Le théorème suivant [39] permet d'obtenir le comportement asymptotique de $\psi(x,y)$:

Théorème 4.4.1. *Pour tout réel strictement positif u, on a:*

$$\lim_{x\to\infty} \psi(x, x^{1/u})/x = \rho(u) \ ,$$

où la fonction $\rho(t)$, appelée fonction de Dickman, est définie par:

$$\rho(t) = \begin{cases} 1 & si \ \ 0 \leq t \leq 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & si \ \ n \leq t \leq n+1 \ . \end{cases}$$

Le théorème précédent montre que pour un entier z *uniformément distribué* entre un et x, la probabilité que z soit lisse par rapport à $x^{1/u}$ est asymptotiquement $\rho(u)$. Cependant, les entiers que nous considérons dans la suite ne sont pas uniformément distribués. Par conséquent, les probabilités et les complexités doivent êtres considérées comme heuristiques.

D'après le théorème 4.4.1, la probabilité que w soit B-lisse est approximativement $\rho(R/\log_2 B)$. Ainsi, en utilisant deux messages chiffrés, la probabilité de trouver tous les facteurs de w est $\rho(R/\log_2 B)$. En utilisant k chiffrés, $k\cdot(k-1)/2$ couples peuvent être obtenus. Si on suppose une indépendance statistique entre la factorisation des w correspondants, approximativement

$$\sqrt{2/\rho(R/\log_2 B)}$$

chiffrés sont nécessaires pour espérer obtenir la factorisation d'au moins un ω avec une complexité moyenne :

$$C(B)/\rho(R/\log_2 B) \tag{4.4}$$

où $C(B)$ est la complexité moyenne nécessaire pour extraire tous les facteurs premiers inférieurs à B.

Dans la pratique, un algorithme de factorisation commence par effectuer des divisions successives jusqu'à une certaine borne B' (nous avons pris $B' = 15000$), puis utilise la méthode de Pollard, puis finalement l'algorithme ECM. Dans la table 4.1, nous donnons les temps de calculs obtenus sur un Pentium 233-MHz pour extraire

L	32	40	48	56	64	72
temps en secondes	6	15	50	90	291	730

Tableau 4.1. Temps de calcul nécessaires pour extraire un premier de L bits d'un nombre de 1024 bits.

un facteur premier de taille L bits d'un nombre de 1024 bits, en utilisant la librairie de calcul MIRACL [65].

Le tableau 4.1 montre clairement que pour $R \leq 72$, les facteurs de l'entier ω pourront être retrouvés efficacement. Pour $R \geq 72$, nous estimons avec l'équation (4.4) dans la table 4.2 le temps d'exécution nécessaire et le nombre de chiffrés requis, lorsqu'on se limite à l'extraction de facteurs de taille inférieure à 72 bits.

L	128	160	192	224	256
temps en secondes	1719	3440	7654	19010	51127
nombre de chiffrés	3	4	5	8	12

Tableau 4.2. Temps d'exécution et nombre approximatif de messages chiffrés nécessaires pour retrouver les facteurs d'au moins un entier ω.

4.4.2 Identification des candidats pour l'entier ω

Nous obtenons de la partie précédente un ensemble de premiers $\mathcal{P} = \{p_1, \ldots, p_i\}$ diviseurs de Δ, et nous supposons que $\mathcal{P}$ contient tous les facteurs de l'entier ω. De l'ensemble $\mathcal{P}$ nous dérivons un ensemble $\mathcal{D}$ de diviseurs de Δ, qui contient ω. L'ensemble $\mathcal{D}$ représente l'ensemble des candidats à tester pour retrouver ω.

Notons $d(k)$ le nombre de diviseurs d'un entier k. Le théorème suivant [55] donne une estimation du nombre de diviseurs d'un entier aléatoire. On dit qu'une fonction arithmétique $f(k)$ est d'*ordre moyen* $g(k)$ si

$$f(1) + f(2) + \ldots + f(k) \sim g(1) + \ldots + g(k)$$

Théorème 4.4.2. *L'ordre moyen de* $d(k)$ *est* $\log k$. *Plus précisément, on a :*

$$d(1) + d(2) + \ldots + d(k) = k \log k + (2\gamma - 1)k + \mathcal{O}(\sqrt{k})$$

où γ *est la constante d'Euler.*

Le théorème 4.4.2 montre que si Δ était uniformément distribué entre un et N, le nombre moyen de diviseurs de Δ serait approximativement $\log N$, et donc le nombre moyen de candidats pour ω serait inférieur à $\log N$. Mais comme Δ n'est pas uniformément distribué entre un et N, cela ne fournit qu'un argument heuristique justifiant que le nombre moyen de candidats est inférieur à $\log N$, donc polynomial en la taille de N. Dans la pratique, il n'est pas nécessaire de tester tous les candidats parce que seuls les candidats de taille inférieure à R et proche de R ont des chances d'être égaux à ω.

4.4.3 Retrouver m en utilisant l'attaque sur RSA avec petit exposant avec messages liés

On obtient ainsi une liste de diviseurs de Δ, l'un des éléments de la liste étant ω. Il suffit alors d'appliquer l'attaque de Franklin-Reiter décrite dans le chapitre précédent. Pour le candidat égal à ω, l'attaque permet de retrouver le message m.

Plus précisément, soient $m_1 = \text{PKCS}(m, r_1)$ et $m_2 = \text{PKCS}(m, r_2)$, on a :

$$\begin{aligned} c_1 &= (m_1)^e \bmod N \\ c_2 &= (m_2)^e \bmod N \\ m_2 &= m_1 - 2^\beta \cdot \omega \ . \end{aligned}$$

Pour chaque diviseur Δ_j de Δ, l'attaquant calcule modulo N:

$$\mathcal{R}_j(z) = \text{PGCD}(z^e - c_1, (z - 2^\beta \Delta_j)^e - c_2) \ .$$

Si $\Delta_j \neq \omega$, alors $\mathcal{R}_j(z) = 1$, tandis que si $\Delta_j = \omega$, alors $\mathcal{R}_j(z) = z - m_1$ avec forte probabilité, ce qui permet de retrouver le message m.

4.5 Comparaison avec l'attaque de Coppersmith sur RSA avec petit exposant

L'attaque de Coppersmith est décrite dans le chapitre précédent. On a vu que le théorème de Coppersmith permettait de retrouver les racines inférieures à $N^{1/\delta}$ d'un polynôme modulo N de degré δ. On a vu que le théorème de Coppersmith s'appliquait dans les cas suivants:

Messages stéréotypés: Supposons que le clair m soit composé d'une partie connue $B = 2^k \cdot b$ et d'une partie inconnue x. Le message chiffré est $c = m^e = (B + x)^e \bmod N$. En utilisant le théorème de Coppersmith avec le polynôme $p(x) = (B + x)^e - c$, on peut obtenir x à partir de c si $|x| < N^{1/e}$.

Redondance aléatoire: Supposons que deux messages m et m' satisfassent une relation affine du type $m' = m + r$ pour un entier petit et inconnu r. A partir du chiffré par RSA des deux messages :

$$\begin{aligned} c &= m^e \bmod N \\ c' &= (m')^e = (m + r)^e \bmod N \ , \end{aligned}$$

on élimine m des deux équations ci-dessus en prenant leur résultant, ce qui donne un polynôme univarié en r modulo N de degré e^2. En conséquence, si $|r| < N^{1/e^2}$, il est possible d'obtenir r, ce qui permet de retrouver m.

Dans le cas qui nous intéresse, à savoir lorsque le message se termine par Z zéros, l'attaque sur message stéréotypés fonctionne pour

$$e \cdot (M + R) < |N|$$

et l'attaque sur padding aléatoire fonctionne pour :

$$e^2 R < |N| \ .$$

En négligeant les termes constants, la méthode de la section 4.4 fonctionne pour

$$eR + (e-1)M < |N|$$

En conséquence, comme cela est illustré sur la figure 4.1 pour $e = 3$, notre méthode étend la méthode de Coppersmith aux cas où :

$$\begin{cases} \frac{N}{e^2} < R < \frac{N}{e} \\ \frac{N}{e} - R < M < \frac{N}{e-1} - \frac{e}{e-1}R \end{cases}$$

Cependant, la méthode de Coppersmith est polynomiale tandis que notre méthode est sub-exponentielle en la taille de N.

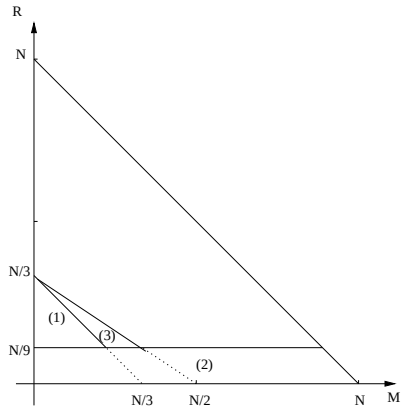

Fig. 4.1. Domaines de validité pour $e = 3$ de l'attaque de Coppermsith sur les messages stéréotypés (1), l'attaque de Coppersmith sur les messages avec padding aléatoire (2), et notre attaque (3)

4.6 Expérimentation de l'attaque sur un module de 1024-bits

Pour confirmer la validité de l'attaque, nous l'avons expérimentée sur le challenge officiel de 1024-bits RSA-309 fourni par les laboratoires RSA, avec un exposant public de chiffrement $e = 3$. Pour les aléas r'_1 et r'_2, nous avons choisi RSA-100 mod 2^{128} et RSA-110 mod 2^{128} respectivement. Les paramètres sont $|N| = 1024$, $M = 280$, $R = 128$, $Z = 592$ et $\beta = 880$. Du fait que $R > |N|/9$ et $R + M > |N|/3$, l'attaque de Coppersmith sur RSA avec petit exposant ne s'applique pas ici.

$N =$	RSA-309							
$=$	bdd14965	645e9e42	e7f658c6	fc3e4c73	c69dc246	451c714e	b182305b	0fd6ed47
	d84bc9a6	10172fb5	6dae2f89	fa40e7c9	521ec3f9	7ea12ff7	c3248181	ceba33b5
	5212378b	579ae662	7bcc0821	30955234	e5b26a3e	425bc125	4326173d	5f4e25a6
	d2e172fe	62d81ced	2c9f362b	982f3065	0881ce46	b7d52f14	885eecf9	03076ca5
$\eta_1 =$	RSA-100							
$=$	000002c8	d59af47c	81ab3725	b472be41	7e3bf7ab	85439af7	26ed3dfd	f66489d1
	55dc0b77	1c7a50ef	7c5e58fb					
$r_1' =$	$\eta_1 \bmod 2^{128}$							
$=$	f66489d1	55dc0b77	1c7a50ef	7c5e58fb				
$\eta_2 =$	RSA-110							
$=$	00000f3d	b4dfacd9	ca1d1c77	cda2c23e	8826c929	130886e2	fffa6e21	271ca6f3
	e2a5a57d	e621eec5	b14ff581	a6368e9b				
$r_2' =$	$\eta_2 \bmod 2^{128}$							
$=$	e2a5a57d	e621eec5	b14ff581	a6368e9b				
$\bar{m} =$	..I.'.m.	..a...c.	i.p.h.e.	r.t.e.x.	t.,...p.	l.e.a.s.	e...b.r.	e.a.k...
	0049276d	20612063	69706865	72746578	742c2070	6c656173	65206272	65616b20
	m.e...!.							
	6d652021							
$m =$	$\bar{m}2^Z$							
$\mu_1 =$	$\text{PKCS}(m, r_1')$							
$=$	0002f664	89d155dc	0b771c7a	50ef7c5e	58fb0049	276d2061	20636970	68657274
	6578742c	20706c65	61736520	62726561	6b206d65	20210000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
$\mu_2 =$	$\text{PKCS}(m, r_2')$							
$=$	0002e2a5	a57de621	eec5b14f	f581a636	8e9b0049	276d2061	20636970	68657274
	6578742c	20706c65	61736520	62726561	6b206d65	20210000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
$c_1 =$	$\mu_1^3 \bmod n$							
$=$	2c488b6f	cf2e3d4c	01b82776	64790af0	d78f82fd	4605fda2	76b9356d	80e82cfb
	8737340f	5a7091b0	38c4bb41	ae6462d9	f751766c	c343c87b	54397ca2	647d6a81
	3609d876	f29554e0	9efcbf2d	b49d8300	5fce9ea8	80fd9cf2	476fbab0	257f1462
	d295a4cb	5468bb86	b3151a49	14e51ed1	7cbc083c	9ae0b4da	9c2a7de0	079df4a0
$c_2 =$	$\mu_2^3 \bmod n$							
$=$	829da9a7	af2c61ed	7bb16f94	7cb90aa7	df8b99df	c06017d7	3afc80fd	64494abb
	3c1cb8db	1167eccd	d1b6d09e	8ca5a98c	c5e19620	b6313eef	495169d7	9ed9a2b1
	cb393e7d	45bea586	49e20986	9a2399f7	f70dd819	90183e1a	3c6a971a	33497e57
	f0ad9fb9	0c7d331e	7108d661	4c487a85	36cf7750	060811d8	70b8a040	e0c39999

En utilisant l'algorithme ECM, on obtient rapidement la factorisation de Δ:

$$\Delta = p_1^5 \times \prod_{i=2}^{10} p_i$$

où les premiers p_i sont donnés ci-après. Parmi les $3072 = 6 \cdot 2^9$ diviseurs possibles, seulement 663 correspondent à des candidats de taille inférieure à 128 bits. On note les candidats par ordre décroissant $\{\Delta_1, \Delta_2, \ldots, \Delta_{663}\}$.

On calcule ensuite:

$$\mathcal{R}_j(z) = \text{GCD}(z^e - c_1, (z - 2^\beta \Delta_j)^e - c_2) \quad \text{pour } 1 \leq j \leq 663$$

Pour $j \neq 25$, on a $\mathcal{R}_j(z) = 1$, et pour $j = 25$, on obtient:

$$\mathcal{R}_{25}(z) = z - m_1$$

On peut vérifier que

$$\Delta_{25} = w = p_1^5 \times p_2 \times p_3 \times p_4 \times p_5 \times p_8 \ ,$$

et

$$m_1 = \mu_1 = \text{PKCS}(m, r_1') \ .$$

$\Delta =$ 00000001 fa75bf4e 390bdf4b 7a0524e0 b9ebed20 5758be2e f1685067 1de199af
0f8714f7 077a6c47 6870ea6d 2de9e7fb 3c40b8d2 017c0197 f9533ed1 f4fe3eab
836b6242 aa03181a 56a78001 7c164f7a c54ecfa7 73583ad8 ffeb3a78 eb8bcbe2
8869da15 60be7922 699dc29a 52038f7b 83e73d4e 7082700d 85d3a720

$p_1 =$ 00000002

$p_2 =$ 00000007

$p_3 =$ 00000035

$p_4 =$ 000000c5

$p_5 =$ 4330e379

$p_6 =$ 548063d7

$p_7 =$ 001ebf96 ff071021

$p_8 =$ 0000021b ac4d83ae 7dedba55

$p_9 =$ 0000128a ec52c6ec 096996bf

$p_{10} =$ 00000022 e3b1a6b0 13829b67 f604074a 5a1135b3 45be0835 ea407ed7 8138a27a
112e78c8 131f3bc3 b6d17dc0 e8a905f1 ca4b6aff 680bc58c 4962309d c7aaccad
2116235c b0d6803e e0a58ca7 55cbea23 e936f189 a76dfbeb

$\Delta_{25} =$ 13bee453 6fba1cb1 6b2a5b6d d627ca60

$\mathcal{R}_{25}(z) = z - m_1$

$m_1/2^Z =$ ..I.'.m. ..a...c. i.p.h.e. r.t.e.x. t.,...p. l.e.a.s. e...b.r. e.a.k...
0049276d 20612063 69706865 72746578 742c2070 6c656173 65206272 65616b20
m.e...!.
6d652021

4.7 Attaque à clair choisi pour un exposant de chiffrement quelconque

4.7.1 Description de l'attaque

Dans cette partie nous décrivons une attaque à clair choisi sur le schéma de chiffrement PKCS#1 v1.5 qui fonctionne pour un exposant de chiffrement e quelconque. Cette attaque est une variante d'une attaque décrite dans [14] sur le chiffrement

RSA simple. Elle s'applique seulement au cas de messages très courts. Comme dans la partie 4.4 nous considérons seulement des messages finissant par Z zéros :

$$m = m'||0\ldots0$$

Pour un aléa r' constitué d'octets différents de zéro, le standard PKCS#1 v1.5 transforme le message m en :

$$\text{PKCS}(m, r') = \mathtt{0002}_{16}||r'||\mathtt{00}_{16}||m'||0\ldots0$$

et le chiffre en :

$$c = \text{PKCS}(m, r')^e \bmod N$$

En notant $x = \mathtt{0002}_{16}||r'||\mathtt{00}_{16}||m'$, on peut écrire

$$\text{PKCS}(m, r') = x \cdot 2^Z \ .$$

On note

$$y = c/2^{eZ} = x^e \bmod N$$

ainsi que M la taille de m' en bits et X la taille de x en bits. On a donc $X = M + R + 10$.

Supposons que $x = x_1 \cdot x_2$ où x_1 et x_2 sont deux entiers inférieurs à une borne B. On construit la table:

$$\frac{y}{i^e} \bmod N \quad \text{pour} \quad i = 1, \ldots, B$$

et pour chaque $j = 0, \ldots, B$ on détermine si $j^e \bmod N$ appartient à la table. Si c'est le cas, on a $y/i^e = j^e \bmod N$ et à partir du couple (i, j) on retrouve $x = i \cdot j$, ce qui donne le message m.

4.7.2 Analyse

L'attaque requiert un temps moyen $\mathcal{O}(B((\log N)^3 + \log B))$. On note $\phi(x, y)$ le nombre d'entiers $v < x$ tels que v peut s'écrire $v = v_1 \cdot v_2$ avec $v_1 < y$ et $v_2 < y$. Le théorème suivant donne une borne inférieure pour $\phi(x, y)$.

Théorème 4.7.1. *Pour $x \to \infty$ et $1/2 < \alpha < 1$,*

$$\liminf \phi(x, x^\alpha)/x \geq \log(2\alpha) \ . \tag{4.5}$$

Preuve. Pour $y > \lceil\sqrt{x}\rceil$, nous notons :

$$\mathcal{T}(x, y) = \{v < x, \text{ tel que } v \text{ est } y\text{-lisse et pas } \lceil x/y\rceil\text{-lisse}\} \ .$$

Tout entier $v \in \mathcal{T}(x, y)$ possède un facteur premier p compris entre $\lceil x/y\rceil$ et y, et donc $v = p \cdot r$ où $p < y$ et $r < y$. En conséquence,

$$\phi(x, y) \geq \#\mathcal{T}(x, y) \ . \tag{4.6}$$

Le théorème 4.4.1 et $\rho(t) = 1 - \log t$ pour $1 \leq t \leq 2$ permettent d'obtenir:

$$\lim_{x \to \infty} \#\mathcal{T}(x, x^\alpha)/x = \log(2\alpha)$$

ce qui donne (4.5) en utilisant (4.6) □

Comme x n'est pas uniformément distribué entre zéro et 2^X, le théorème 4.7.1 ne donne qu'un argument heuristique justifiant qu'en prenant $B = 2^{\alpha X}$ avec $\alpha > 1/2$, l'attaque permet de retrouver x avec probabilité supérieure à

$$\log(2\alpha)$$

et une complexité proche de

$$2^{\alpha \cdot X} .$$

Ainsi, un message de huit bits chiffré à l'aide de PKCS#1 v1.5 avec un aléa de 64 bits peut être retrouvé avec probabilité voisine de 0.08 en complexité 2^{44} en temps et en espace (en prenant $\alpha = 0.54$).

4.8 Conclusion

Nous avons proposé deux attaques à clair choisi contre le standard de chiffrement PKCS#1 v1.5. La première attaque s'applique pour des petits exposants de chiffrement et montre qu'un message se terminant par suffisamment de zéros peut être retrouvé à partir de plusieurs chiffrés de ce même message. Cette attaque permet d'étendre le domaine sur lequel s'applique l'attaque de Coppersmith. Notre seconde attaque montre comment étendre l'attaque précédente sur PKCS#1 v1.5 à n'importe quel exposant public de chiffrement.

Ces attaques montrent le risque encouru lorsqu'on choisit des formats de message dont la sécurité n'est pas clairement justifiée, et montrent la nécessité d'utiliser des schémas de chiffrement à sécurité prouvée, comme par exemple OAEP (PKCS#1 v2) [8].

Partie III

Cryptanalyse de schémas de signature basés sur RSA

5. Attaques sur les signatures RSA

Comme nous l'avons vu au chapitre 2, l'algorithme RSA permet de réaliser un schéma de signature électronique. Nous avons vu aussi que de nombreuses attaques sont apparues sur les algorithmes de signature basés sur RSA. Ces attaques ne mettent pas en cause l'algorithme RSA lui-même, mais plutôt son utilisation incorrecte. On a vu que si on n'applique aucun format particulier au message M, on était directement exposé à une attaque par forge existentielle ou par masquage.

Deux méthodes permettent couramment de se prémunir contre de telles attaques. La première consiste à hacher le message avant de le signer et la deuxième consiste à ajouter de la redondance au message. Ces deux méthodes peuvent se révéler également vulnérables. Ainsi, De Jonge et Chaum ont décrit dans [37] la première attaque sur un schéma de signature avec redondance. Desmedt et Odlyzko décrivent dans [38] une attaque contre le cryptosystème RSA qui peut s'appliquer au cas des signatures avec fonction de hachage. Les résultats de De Jonge et Chaum ont été étendus par Girault et Misarsky dans [50] au cas des redondances modulaires puis par Misarsky dans [66] au cas où les redondances sont séparées en plusieurs parties, en utilisant l'algorithme de réduction de réseau LLL [61]. Nous référons le lecteur à la thèse de Misarsky [68] qui décrit ces attaques en détail.

Dans ce chapitre, après avoir rappelé les différents modèles d'attaque contre un schéma de signature, on étudie les principales attaques connues s'appliquant spécifiquement aux signatures RSA. Nous montrons également qu'il est possible d'étendre l'attaque de Girault et Misarsky au cas de redondances de taille plus grande; cette attaque sera présentée à la conférence Crypto 2001 [16].

5.1 Les différents modèles d'attaque

Comme dans le cas du chiffrement, le modèle d'attaque dépend à la fois de l'objectif de l'attaquant et des ressources dont il dispose.

Lorsque l'objectif de l'attaquant est de forger la signature d'un message, sans pouvoir nécessairement le choisir à l'avance, on parle de *forge existentielle*. C'est le niveau de succès le plus faible. Lorsque l'attaquant veut forger la signature d'un message choisi à l'avance, il s'agit d'une *forge sélective*. Lorsque l'attaquant veut de plus être capable de signer n'importe quel message, on parlera de *forge universelle*. Enfin, le *cassage total* correspond au cas où l'attaquant retrouve la clef privée du signeur. C'est le niveau de succès le plus fort.

Les ressources les plus faibles de l'attaquant correspondent au cas où il ne connaît que la clef publique du signeur; on parle dans ce cas d'*attaque à clé publique seule*. Lors d'une *attaque à signature connue*, l'attaquant connaît la signature de certains messages choisis par le signeur. Enfin, lors d'une *attaque à message choisi adaptative*, l'attaquant peut obtenir la signature de messages de son choix. C'est l'attaque la plus puissante.

Le modèle de sécurité le plus fort correspond donc au cas où il est impossible pour un attaquant de réaliser une forge existentielle lors d'une attaque à message choisie adaptative. Autrement dit, l'attaquant ne peut pas produire lui-même une signature valide d'un message, même inintelligible, et ceci même après avoir obtenu la signature d'autres messages de son choix.

5.2 Attaque de Desmedt et Odlyzko

Il est possible d'adapter l'attaque de Desmedt et Odlyzko décrite dans le chapitre consacré à RSA au cas des schémas de signature avec redondance ou utilisant une fonction de hachage. Cette variante est décrite par Misarsky dans [67]. L'attaque s'applique au cas où le message à signer est relativement petit.

Soit m le message à signer et $\mu(m)$ la fonction de hachage ou de redondance associée. On veut obtenir $\mu(m)^d \bmod N$ sans connaître la clef privée d.

1. On commence par factoriser l'entier $\mu(m)$ en produit de petits premiers.

2. On obtient les racines e-èmes des petits facteurs premiers en combinant les signatures de messages dont les facteurs premiers sont aussi petits.

3. On obtient finalement la signature de m en multipliant les racines e-èmes des petits facteurs premiers.

La complexité de l'attaque dépend de la taille de $\mu(m)$, et pas de la taille du module N. L'attaque ne s'applique que dans le cas où l'entier $\mu(m)$ est de petite taille, car dans le cas contraire, la probabilité qu'il se décompose en produit de petits facteurs premiers est trop faible. On évite cette attaque en choisissant $\mu(m)$ de la taille du module N.

5.3 Attaques sur les signatures RSA avec redondance

Une *fonction de redondance* R est une fonction inversible prenant en entrée un message m et renvoyant en sortie une chaîne de bits, telle qu'il est facile de calculer R^{-1}. On définit la taille de la redondance comme la taille en bits de $R(m)$ moins la taille du message m.

Pour signer un message m, on calcule:

$$s = R(m)^d \bmod N$$

Pour vérifier la signature s, le vérifieur obtient

$$R(m) = s^e \bmod N$$

retrouve le message m et vérifie que $R(m)$ respecte la redondance.

Dans la suite de ce chapitre, on considère plus particulièrement les signatures RSA avec redondance fixée. Pour signer un message m, le signeur concatène à m un block fixé P:

$$R(m) = P \| m$$

et la signature s'obtient en calculant:

$$s = (P \| m)^d \bmod N$$

où d est l'exposant privé et N le module.

Plus généralement, on considère les signatures RSA à redondance affine, donnée par:

$$R(m) = \omega \cdot m + a \quad \text{où} \quad \begin{cases} w \text{ est la redondance multiplicative} \\ a \text{ est la redondance additive} \end{cases} \tag{5.1}$$

Une redondance fixée à gauche $P \| m$ correspond à $\omega = 1$ et $a = P \cdot 2^\ell$, tandis qu'une redondance fixée à droite $m \| P$ s'obtient avec $\omega = 2^\ell$ et $a = P$.

5.3.1 L'attaque de De Jonge et Chaum

L'attaque de De Jonge et Chaum [37] s'applique lorsqu'une fonction de redondance affine est utilisée:

$$R(m) = \omega \cdot m + a$$

L'attaque est basée sur l'utilisation de l'algorithme d'Euclide.

Si la redondance additive a est nulle, alors l'attaque s'applique quelle que soit la valeur de ω, à condition que la taille de la redondance soit inférieure à la moitié de la taille du module N:

$$|\text{redondance}| \prec \frac{1}{2}|N|$$

Si la redondance multiplicative est égale à 1, alors l'attaque s'applique quelle que soit la valeur de a, à condition que la taille de la redondance soit inférieure au tiers de la taille du module:

$$|\text{redondance}| \prec \frac{1}{3}|N|$$

La figure 5.1 montre un exemple de signature RSA à redondance affine pouvant être forgée par cette méthode.

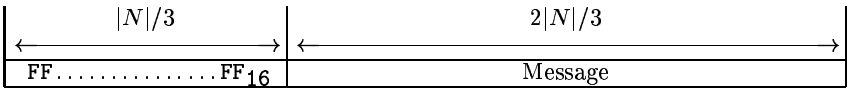

Fig. 5.1. Exemple de redondance fixée pouvant être attaquée par la méthode de De Jonge et Chaum, avec $\omega = 1$ et $a = \texttt{FF}\ldots\texttt{FF}\ \texttt{00}\ldots\texttt{00}_{16}$

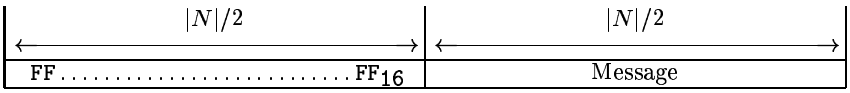

Fig. 5.2. Exemple de redondance fixée pouvant être attaquée par la méthode de Girault et Misarsky, avec $\omega = 1$ et $a = \texttt{FF}\ldots\texttt{FF}\ \texttt{00}\ldots\texttt{00}_{16}$

5.3.2 L'attaque de Girault et Misarksy

L'attaque précédente a été étendue par Girault et Misarsky [50], en utilisant un algorithme dû à Okamoto et Shiraishi [70], qui est une extension de l'algorithme d'Euclide. L'attaque de Girault et Misarsky améliore l'efficacité de l'attaque de De Jonge et Chaum sur une fonction de redondance affine. De plus, l'attaque s'applique aussi lorsqu'une redondance modulaire est utilisée:

$$R(m) = \omega_1 \cdot m + \omega_2 \cdot \phi(m) + a \tag{5.2}$$

où ω_1, ω_2 sont les redondances multiplicatives, a est la redondance additive et $\phi(m)$ est la redondance modulaire, avec

$$\phi(m) = m \bmod m_r$$

où m_r est une constante.

Lorsque $R(m)$ est une fonction affine de m, c'est à dire lorsque $\omega_2 = 0$, l'attaque s'applique quels que soient ω_1 et a, à condition que la taille de la redondance soit inférieure à la moitié de la taille du module N:

$$|\text{redondance}| \prec \frac{1}{2}|N|$$

Cette attaque est illustrée à la figure 5.2.

Dans le cas général, pour une fonction $R(m)$ donnée par (5.2), l'attaque s'applique à condition que la taille de la redondance soit inférieure à la moitié de la taille du module moins la taille de la redondance modulaire:

$$|\text{redondance}| \prec \frac{1}{2}|N| - |\text{redondance modulaire}|$$

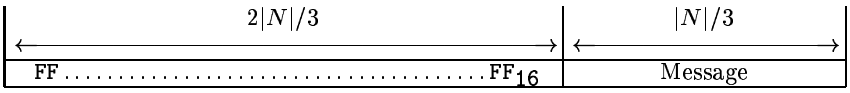

Fig. 5.3. Exemple de redondance fixée pouvant être attaquée par notre méthode avec $\omega = 1$ et $a =$ FF...FF 00...00$_{16}$

5.3.3 L'attaque de Misarsky

L'attaque de Misarsky [66] étend l'attaque précédente au cas où le message m et la redondance modulaire sont séparés en plusieurs parties, qui n'ont pas nécessairement la même taille:

$$R(m) = \sum_{i=1}^{k_1} m_i \cdot \omega_i + \sum_{j=1}^{k_2} \phi(m)_j \omega_{k_1+j} + a$$

où $w_1, w_2, \ldots$ sont les différentes redondances multiplicatives, m_i sont les différentes parties du messages m, $\phi(m)_j$ les différentes parties de la redondance modulaire $\phi(m)$, et a la redondance additive. L'attaque de Misarsky utilise l'algorithme de réduction de réseau LLL [61].

5.4 Extension de l'attaque de Girault et Misarsky

5.4.1 Description de l'attaque

Dans ce paragraphe, on étend l'attaque de Girault et Misarsky contre les signatures RSA à redondance affine,

$$R(m) = \omega \cdot m + a$$

pour des tailles de redondance allant jusqu'aux deux-tiers de la taille du module, comme cela est illustré à la figure 5.3.

$$|\text{redondance}| \prec \frac{2}{3}|N|$$

Comme l'attaque de Girault et Misarsky, notre attaque est une attaque multiplicative s'appliquant quels que soient ω et a et de complexité polynomiale.

Une attaque multiplicative est une attaque dans laquelle la fonction de redondance d'un message peut s'exprimer comme combinaison multiplicative des fonctions de redondance d'autres messages. Par exemple, l'attaque de Girault et Misarsky permet de trouver trois messages m_1, m_2 et m_3 tels que:

$$R(m_1) \cdot R(m_2) = R(m_3) \bmod N$$

Ensuite, à partir des signatures de m_2 et m_3 on peut obtenir la signature de m_1 avec:

$$R(m_1)^d = \frac{R(m_3)^d}{R(m_2)^d} \bmod N$$

De plus, l'attaque de Girault et Misarsky est sélective, c'est à dire que le message m_1 peut être entièrement déterminé par l'attaquant.

Dans notre attaque, on cherche quatre messages distincts m_1, m_2, m_3 et m_4, tous de taille inférieure à un tiers de la taille du module N, tels que

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \bmod N$$

Notre attaque sera seulement existentielle: on ne sera pas capable de déterminer à l'avance l'un des messages m_1, m_2, m_3 ou m_4. On obtient:

$$(\omega \cdot m_1 + a) \cdot (\omega \cdot m_2 + a) = (\omega \cdot m_3 + a) \cdot (\omega \cdot m_4 + a) \bmod N$$

En notant $P = a/\omega \bmod N$, on obtient:

$$(P + m_1) \cdot (P + m_2) = (P + m_3) \cdot (P + m_4) \bmod N$$

et en notant:

$$\begin{array}{lcllcl} t & = & m_3 & \qquad y & = & m_2 - m_3 \\ x & = & m_1 - m_3 & \qquad z & = & m_4 - m_1 - m_2 + m_3 \end{array} \tag{5.3}$$

on obtient

$$((P + t) + x) \cdot ((P + t) + y) = (P + t) \cdot ((P + t) + x + y + z) \bmod N$$

ce qui se simplifie en:

$$x \cdot y = (P + t) \cdot z \bmod N \tag{5.4}$$

Il faut donc trouver quatre entiers x, y, z et t, tous de taille inférieure au tiers de la taille de N, qui doivent satisfaire l'équation (5.4).

D'abord, on obtient deux entiers strictement positifs z et u tels que:

$$P \cdot z = u \bmod N \quad \text{avec} \quad \begin{cases} -N^{\frac{1}{3}} < z < N^{\frac{1}{3}} \\ 0 < u < 2 \cdot N^{\frac{2}{3}} \end{cases}$$

Cela revient à trouver une bonne approximation de la fraction P/N, et peut se faire en développant P/N en fraction continue, c'est à dire en appliquant l'algorithme d'Euclide étendu à P et N. On obtient une solution telle que $|z| < Z$ et $0 < u < U$ si $Z \cdot U > N$, ce qui est le cas ici avec $Z = N^{\frac{1}{3}}$ et $U = 2 \cdot N^{\frac{2}{3}}$.

Ensuite on sélectionne un entier y tel que $N^{\frac{1}{3}} \leq y \leq 2 \cdot N^{\frac{1}{3}}$ et $\mathrm{PGCD}(y, z) = 1$. On calcule ensuite l'entier positif $t < y$ tel que:

$$t \cdot z = -u \bmod y$$

ce qui est possible car $\mathrm{PGCD}(y, z) = 1$. Ensuite on définit l'entier x tel que

$$x = \frac{u + t \cdot z}{y} \leq 4N^{\frac{1}{3}}$$

et on obtient

$$P \cdot z = u = x \cdot y - t \cdot z \bmod N$$

ce qui donne l'équation (5.4), avec x, y, z et t qui sont des entiers positifs tous inférieurs à $4 \cdot N^{\frac{1}{3}}$. A partir de x, y, z et t, on obtient en utilisant (5.3) quatre messages m_1, m_2, m_3 et m_4 de taille un tiers de la taille du module N, tels que:

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \bmod N$$

et on peut exprimer la signature de m_1 en fonction de la signature de m_2, m_3 et m_4.

On a donc réalisé une attaque à message choisi: l'attaquant demande la signature des messages m_2, m_3 et m_4, pour être ensuite capable d'exhiber la signature de m_1, qui n'a jamais été signé par le signeur légitime. La complexité de notre attaque est polynomiale en la taille du module N.

5.4.2 Application pratique

Dans ce paragraphe, on donne un exemple de signature forgée en utilisant un module de 1024 bits proposé par les laboratoires RSA, dont la factorisation est encore inconnue. On prend $\omega = 1$ et $a = 2^{1023} - 2^{352}$.

$N =$	RSA-309							
$=$	bdd14965	645e9e42	e7f658c6	fc3e4c73	c69dc246	451c714e	b182305b	0fd6ed47
	d84bc9a6	10172fb5	6dae2f89	fa40e7c9	521ec3f9	7ea12ff7	c3248181	ceba33b5
	5212378b	579ae662	7bcc0821	30955234	e5b26a3e	425bc125	4326173d	5f4e25a6
	d2e172fe	62d81ced	2c9f362b	982f3065	0881ce46	b7d52f14	885eecf9	03076ca5
$R(m_1) =$	7fffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	00415df4	ca4219b6	ea5fa8e4
	e2eabcfc	61348b80	e7ccbac7	3d1f5cc7	249e1519	9412886a	f76220c6	d1409cd6
$R(m_2) =$	7fffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	00127f44	f753253a	a0348be7
	826e893f	693032db	c2194dbb	3b81e1c2	630b66d3	1448a3f4	7fd2d34f	b28aefd6
$R(m_3) =$	7fffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	00781bd4	e0c918a7	308fcff7
	8f64044c	a35b4937	36cd37d7	93f281b5	fdd0a951	52a0479b	57dd73b2	25b6df85
$R(m_4) =$	7fffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff	000919fd	86e5afce	7fc11c94
	0e0827c8	03be05bb	71f8de48	c61d6d5f	0feb036d	a1ff2f8b	5f596108	3d142538

On vérifie que

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \bmod N$$

où les messages m_1, m_2, m_3 et m_4 sont de taille un tiers de la taille du module.

5.5 Conclusion

Nous avons rappelé les principales attaques connues sur les signatures RSA, et nous avons montré comment étendre l'attaque de Girault et Misarsky sur les signatures RSA à redondance affine: nous avons décrit une attaque à message choisi pour des tailles de redondance allant jusqu'aux deux-tiers de la taille du module. Un problème intéressant consiste à essayer d'étendre cette attaque à des tailles de redondance encore plus grande, par exemple pour une redondance de taille les trois-quarts de la taille du module.

Parmi les attaques que nous avons rappelées, la plus puissante est sans doute l'attaque de Desmedt-Odlyzko, puisqu'elle ne suppose rien sur la fonction de redondance utilisée. La seule contrainte est que la taille de la fonction $\mu(m)$ soit suffisamment petite, bien inférieure à la taille de N, pour que $\mu(m)$ puisse se décomposer en produit de petits facteurs premiers avec forte probabilité. Nous verrons dans le prochain chapitre comment étendre l'attaque de Desmedt-Odlyzko à certains schémas de signature RSA pour lesquels la fonction $\mu(m)$ est de la taille de N.

6. Attaques sur les standards ISO 9796-1 et ISO 9796-2

6.1 Introduction

Dans ce chapitre, on montre que l'attaque de Desmedt et Odlyzko décrite au chapitre précédent s'étend aux standards de signature digitale ISO 9796-1 et ISO 9796-2. Cette nouvelle attaque a été présentée à la conférence Crypto '99 [34]. Comme l'attaque de Desmedt et Odlyzko, c'est une attaque à message choisi: l'attaquant obtient la signature des messages $m_1, \ldots, m_{\tau-1}$ et exhibe la signature d'un message m_τ qui n'a jamais été signé par le signataire. En notant $\mu(m)$ la fonction de redondance du schéma de signature, la méthode consiste à dériver de chacun des $\mu(m_i)$ un entier qui soit le produit de petits facteurs premiers, pour pouvoir ensuite exprimer $\mu(m_\tau)$ comme une combinaison multiplicative des $\mu(m_1), \ldots, \mu(m_{\tau-1})$. On en déduit alors la signature de $\mu(m_\tau)$ en fonction de la signature de $\mu(m_1), \ldots, \mu(m_{\tau-1})$.

Notre méthode consiste à tout d'abord étendre l'attaque de Desmedt et Odlyzko au cas où il existe une combinaison linéaire entière petite de $\mu(m)$ et du module N, et au cas où $\mu(m)$ est le produit d'une constante c par un entier petit. Ensuite, on montre qu'il est effectivement possible dans le standard ISO 9796-2 d'obtenir une combinaison linéaire entière petite de $\mu(m)$ et de N, et qu'il est possible dans le standard ISO 9796-1 d'exprimer $\mu(m)$ comme le produit d'une constante c par un entier petit. Dans le cas où un exposant de chiffrement pair est utilisé, l'attaque permet de factoriser le module N: il s'agit alors d'un cassage total.

6.2 Le schéma de signature de Rabin-Williams

6.2.1 Rappels en théorie des nombres

Si p est un nombre premier impair et si $x \neq 0 \bmod p$, on dit que x est un *résidu quadratique* modulo p s'il existe un entier z tel que $x = z^2 \bmod p$. On note QR_p l'ensemble des résidus quadratiques modulo p. On dit que x est un non-résidu quadratique modulo p si $x \neq 0 \bmod p$ et si $x \notin QR_p$. On note NQR_p l'ensemble des non-résidus quadratiques modulo p.

Si p est un nombre premier impair, on définit le symbole de Legendre $\left(\frac{x}{p}\right)$ pour tout entier $x \geq 0$ par:

$$\left(\frac{x}{p}\right) = \begin{cases} 0, & \text{si } x = 0 \bmod p \\ 1, & \text{si } x \in QR_p \\ -1, & \text{si } x \in QNR_p \end{cases}$$

On montre que si p est un nombre premier impair, alors:

$$\left(\frac{x}{p}\right) = x^{(p-1)/2} \bmod p$$

Soit n un entier impair composite dont la décomposition en facteurs premiers s'écrit $n = \prod_{i=1}^{k} p_i^{e_i}$. Soit un entier $x \geq 0$, on définit le symbole de Jacobi par:

$$\left(\frac{x}{n}\right) = \prod_{i=1}^{k} \left(\frac{x}{p_i}\right)^{e_i}$$

Il n'est pas nécessaire de connaître la factorisation de n pour calculer le symbole de Jacobi d'un entier par rapport à n. On calcule le symbole de Jacobi en utilisant des résultats de théorie des nombres dont le plus important est la loi de réciprocité quadratique:

$$\text{Soient } m, n \in \mathbb{Z} \text{ avec } m \geq 3 \text{ et } n \geq 3 : \left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)(-1)^{(m-1)(n-1)/4}$$

On a de plus:

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$$

$$\left(\frac{2}{n}\right) = \begin{cases} 1, & \text{si } n = 1 \text{ ou } 7 \bmod n \\ -1, & \text{si } n = 3 \text{ ou } 5 \bmod n \end{cases}$$

$$\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$$

Pour une description précise des propriétés et du mode de calcul du symbole de Jacobi, nous référons le lecteur à l'ouvrage de Stinson [80].

6.2.2 Le schéma de signature de Rabin-Williams

Le schéma de signature de Rabin-Williams [83] est défini de la façon suivante. On note $\mathrm{J}(x)$ le symbole de Jacobi de x par rapport à N. Le schéma de Rabin-Williams impose que $\mu(m) = 6 \bmod 16$. Le module $N = p \cdot q$ est tel que $p = 3 \bmod 8$ et $q = 7 \bmod 8$. L'exposant de chiffrement est $e = 2$ et l'exposant de déchiffrement est:

$$d = \frac{N - p - q + 5}{8}$$

Avant de signer, on vérifie que

$$\mathrm{J}(\mu(m)) = 1$$

Si on a $\mathrm{J}(\mu(m)) = -1$, on remplace $\mu(m)$ par $\mu(m)/2$ pour garantir que $\mathrm{J}(\mu(m)) = 1$ car $\mathrm{J}(2) = -1$. La signature du message m est:

$$s = \mu(m)^d \bmod N$$

La signature s est valide si $w = s^2 \bmod n$ est tel que:

$$\mu(m) \stackrel{?}{=} \begin{cases} w & \text{si} \quad w = 6 \bmod 8 \\ 2 \cdot w & \text{si} \quad w = 3 \bmod 8 \\ N - w & \text{si} \quad w = 7 \bmod 8 \\ 2 \cdot (N - w) & \text{si} \quad w = 2 \bmod 8 \end{cases}$$

6.3 Extension de l'attaque de Desmedt et Odlyzko

On décrit dans ce paragraphe l'extension de l'attaque de Desmedt et Odlyzko au cas où il est possible de trouver des constantes a et b telles que l'entier

$$t = a \cdot \mu(m) + b \cdot N$$

soit de petite taille, ou lorsqu'il est possible de trouver une constante c telle que $\mu(m)$ puisse s'écrire:

$$\mu(m) = c \cdot t$$

avec t un entier de petite taille.

On voit qu'en prenant $c = a^{-1} \bmod N$, on peut toujours se ramener au cas où:

$$\mu(m) = c \cdot t \bmod N$$

où t est un entier de petite taille.

L'attaque consiste à chercher un certain nombre de messages m_i tels que l'entier t_i dans

$$\mu(m_i) = c \cdot t_i \bmod N \tag{6.1}$$

soit y-lisse. On dit qu'un entier est y-lisse si tous ses facteurs premiers sont inférieurs ou égaux à y. En notant $(p_1, \ldots, p_k)$ la liste des k nombres premiers inférieurs à y, on peut écrire:

$$\mu(m_i) = c \cdot \prod_{j=1}^{k} p_j^{v_{i,j}} \bmod N \quad \text{pour } 1 \leq i \leq \tau$$

On associe à chacun des $\mu(m_i)$ le vecteur $\boldsymbol{V}_i$ de dimension $k+1$:

$$\mu(m_i) \longmapsto \boldsymbol{V}_i = \{1, v_{i,1} \bmod e, \ldots, v_{i,k} \bmod e\}$$

On exprime ensuite par élimination gaussienne l'un des vecteurs (que l'on renomme $\boldsymbol{V}_\tau$) comme combinaison linéaire des autres:

$$\boldsymbol{V}_\tau = \sum_{i=1}^{\tau-1} \beta_i \boldsymbol{V}_i \mod e \tag{6.2}$$

A partir de l'équation (6.2) on peut donc écrire:

$$v_{\tau,j} = \sum_{i=1}^{\tau-1} \beta_i \cdot v_{i,j} - \gamma_j \cdot e \quad \text{pour tout} \quad 1 \leq j \leq k$$

et en notant

$$\delta = \prod_{j=1}^{k} p_j^{-\gamma_j}$$

on obtient

$$\mu(m_\tau) = \delta^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i} \bmod N$$

Ainsi, pour le chiffrement RSA, l'attaquant demande la signature des $\tau - 1$ premiers messages m_i et peut ainsi forger la signature du message m_τ avec:

$$\mu(m_\tau)^d = \delta \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \bmod N$$

Dans le cas du schéma de Rabin-Williams, on distingue deux cas:

• Si $\mathrm{J}(\delta) = 1$, alors on montre facilement que $\delta^{2d} = \pm\delta$. Comme dans le cas du chiffrement RSA, l'attaquant demande la signature des $\tau - 1$ premiers messages m_i et peut ainsi forger la signature du message m_τ avec:

$$\mu(m_\tau)^d = \pm\delta \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \bmod N$$

• Si $\mathrm{J}(\delta) = -1$, l'attaquant demande la signature des τ messages m_i et peut ainsi obtenir:

$$u = \delta^{2d} = \mu(m_\tau)^d / \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \bmod N$$

On montre facilement que $u^2 = \delta^2 \bmod N$ et donc $(u-\delta)\cdot(u+\delta) = 0 \bmod N$. Comme $u = (\delta^d)^2 \bmod N$, u est un carré modulo N et donc $\mathrm{J}(u) = 1$. Comme $\mathrm{J}(-1) = 1$, on ne peut pas avoir $\delta = \pm u \bmod N$ car sinon on aurait $\mathrm{J}(\delta) = \mathrm{J}(u) = 1$. On en déduit que

$$\mathrm{PGCD}(u - \delta, N)$$

fournit un des facteurs premiers de N. L'attaquant obtient ainsi la factorisation de N, calcule l'exposant d et peut signer n'importe quel message.

6.4 Complexité de l'attaque

Pour évaluer la complexité de l'attaque, il faut déterminer la probabilité que l'entier t_i dans l'équation (6.1) soit y-lisse, ainsi que le nombre d'entiers y-lisses à obtenir pour pouvoir exprimer l'un des $\mu(m_i)$ comme combinaison multiplicative des autres.

La probabilité qu'un entier z inférieur à x soit y-lisse est une fonction de x et de y. On définit $\psi(x, y)$ le nombre d'entier positifs z inférieurs à x tel que z soit y-lisse:

$$\psi(x, y) = \#\{z < x, \text{ tel que } z \text{ est } y\text{-lisse}\}$$

Le théorème suivant permet d'obtenir le comportement asymptotique de $\psi(x, y)$:

Théorème 6.4.1. *Pour tout réel positif u, on a :*

$$\lim_{x\to\infty} \psi(x, x^{1/u})/x = \rho(u) \ ,$$

où la fonction $\rho(u)$ est appelée fonction de Dickman, définie par:

$$\rho(t) = \begin{cases} 1 & si \ \ 0 \leq t \leq 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & si \ \ n \leq t \leq n+1 \ . \end{cases}$$

La fonction $\rho(u)$ est donc une approximation de la probabilité qu'un nombre de taille inférieure à r bits soit $2^{r/u}$-lisse.

En particulier, en notant

$$y = L_x[\beta] = \exp\left(\beta \cdot \sqrt{\log x \log\log x}\right)$$

la probabilité qu'un entier aléatoire compris entre un et x soit $L_x[\beta]$-lisse est:

$$\frac{\psi(x, y)}{x} = L_x\left[-\frac{1}{2\beta} + \circ(1)\right]$$

Si les entiers t_i donnés par l'équation (6.1) étaient aléatoirement distribués entre un et x, il faudrait donc générer en moyenne $L_x[1/(2\beta) + \circ(1)]$ entiers t_i avant d'en trouver un qui soit y-lisse.

On a vu dans le chapitre précédent que l'algorithme de factorisation ECM extrait un facteur p d'un entier n en temps moyen:

$$L_p[\sqrt{2} + \circ(1)]$$

Un entier y-lisse se factorise donc en temps moyen inférieur à:

$$L_y[\sqrt{2} + \circ(1)] = L_x[\circ(1)]$$

On en déduit que la complexité moyenne pour trouver un entier t_i qui soit y-lisse à l'aide de l'algorithme ECM est:

$$L_x\left[\frac{1}{2\beta} + \circ(1)\right]$$

Cependant, les entiers t_i générés à l'aide de l'équation (6.1) ne sont pas uniformément distribués entre un et x. On doit donc considérer la complexité précédente comme heuristique.

Il reste à déterminer le nombre τ d'entiers y-lisses nécessaires en fonction de y ou de k (l'entier k étant le nombre d'entiers premiers inférieurs à y).

• Si e est un nombre premier, alors l'ensemble des vecteurs à $k+1$ coordonnées sur $\mathbb{Z}_e$ est un espace vectoriel de dimension $k+1$. Par conséquent, $\tau = k+2$ vecteurs sont suffisants pour garantir qu'au moins un vecteur est une combinaison linéaire des autres, que l'on obtient par élimination Gaussienne.

• Si $e = p^r$ avec p un nombre premier, on voit facilement que $\tau = k+2$ vecteurs sont aussi suffisants pour garantir qu'au moins un vecteur peut s'exprimer comme une combinaison linéaire des autres modulo e, que l'on trouve par élimination Gaussienne.

• Dans le cas le plus général où

$$e = \prod_{i=1}^{\omega} p_i^{r_i}$$

il suffit de $\tau = 1 + \omega \cdot (k+1) = \mathcal{O}(k \cdot \log e)$ vecteurs pour garantir qu'au moins un vecteur s'exprime comme combinaison linéaire des autres. En effet, modulo chacun des $p_i^{r_i}$, l'attaquant peut trouver un ensemble T_i de $(\omega-1)\cdot(k+1)+1$ vecteurs, dont chacun peut être exprimé par élimination Gaussienne comme combinaison linéaire d'un ensemble de $k+1$ autres vecteurs. En prenant l'intersection des T_i et en utilisant le théorème du reste Chinois, on obtient qu'au moins un vecteur s'exprime comme combinaison linéaire des autres modulo e.

Dans le cas général, le nombre de vecteurs nécessaires est donc $\mathcal{O}(y \cdot \log e)$. On doit donc résoudre un système de $r = L_x[\beta + \circ(1)]$ équations à $L_x[\beta + \circ(1)]$ inconnues. Ces équations sont très creuses: le nombre de termes différents de 0 dans chaque équation est borné par $2 + \log x$. On résout d'habitude un tel système linéaire par élimination Gaussienne. L'inconvénient est que résoudre par élimination Gaussienne un système de r équations à r inconnues se fait en temps $\mathcal{O}(r^3)$ et en espace $\mathcal{O}(r^2)$.

Pour tirer profit du caractère creux du système d'équation, on applique l'algorithme de Lanczos [60] (voir aussi [22]). Il s'agit d'une méthode de résolution itérative. Pour un système creux tel que défini précédemment, la complexité en temps est $\mathcal{O}(r^2)$ et l'espace requis est essentiellement celui nécessaire pour stocker le système d'équations, soit $\mathcal{O}(r \cdot (\log r)^2))$.

En résumé, le temps nécessaire pour obtenir les $L_x[\beta + \circ(1)]$ équations requises est asymptotiquement:

$$L_x\left[\beta + \frac{1}{2\beta} + \circ(1)\right]$$

On résout ensuite le système en temps

$$L_x[2\beta + \circ(1)]$$

et en espace

$$L_x[\beta + \circ(1)]$$

La complexité en temps est minimale en prenant $\beta = 1/\sqrt{2}$. On obtient ainsi une complexité en temps

$$L_x[\sqrt{2} + \circ(1)]$$

et en espace

$$L_x\left[\frac{\sqrt{2}}{2} + \circ(1)\right]$$

On voit donc que la complexité de l'attaque est sub-exponentielle en la taille des entiers t_i. L'attaque ne sera donc vraiment pratique que si l'on parvient à obtenir des entiers t_i petits.

Dans le tableau suivant, on donne les valeurs des fonctions $L_x[\sqrt{2}]$ et $L_x[\sqrt{2}/2]$ correspondant aux complexités en temps et en espace de l'attaque, en fonction de la taille $|x|$ des entiers t_i. Il ne s'agit bien sûr que d'une approximation de la complexité réelle de l'attaque. Le tableau suggère que l'attaque est susceptible d'être mise en oeuvre dans la pratique lorsque $|x| \leq 128$, mais que l'attaque devient impraticable pour des tailles supérieures. Dans le paragraphe suivant, une attaque concrète sera mise en oeuvre pour $|x| = 64$.

$\|x\|$	$\log_2$ **temps**	$\log_2$ **espace**
64	26	13
96	34	17
128	41	20
192	52	26
256	62	31
368	79	39

Tableau 6.1. Complexité de l'attaque en temps et en espace en fonction de la taille des entiers.

6.5 Cryptanalyse du schéma de signature ISO/IEC-9796-1

6.5.1 Le schéma ISO/IEC-9796-1

Le schéma de signature ISO/IEC-9796-1 [3] a été publié en 1991; ce fut le premier standard international en matière de signature numérique. Le standard ISO/IEC-

9796-1 permet de retrouver le message à partir de sa signature et permet de signer des messages de taille limitée. Pour un module N de taille $2\gamma+1$ bits et un message m de taille γ bits, et en supposant que γ est un multiple de 8, le standard est défini de la façon suivante:

On note par $a||b$ la concaténation des chaînes de bits a et b. On note ω_i le i-ème groupe de 4 bits du message m. On définit $\ell = \gamma/4$. On note par $s(x)$ la table de substitution suivante:

$x =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$s(x) =$	E	3	5	8	9	4	2	F	0	D	B	6	7	A	C	1

On note par $\bar{s}(x)$ la valeur de $s(x)$ dont le bit le plus significatif est forcé à un et par $\tilde{s}(x)$ la valeur de $s(x)$ dont le bit le moins significatif est complémenté. La norme ISO 9796-1 spécifie que:

$$\begin{array}{rcccc} \mu(m) = & \bar{s}(\omega_{\ell-1}) & \tilde{s}(\omega_{\ell-2}) & \omega_{\ell-1} & \omega_{\ell-2} \\ & s(\omega_{\ell-3}) & s(\omega_{\ell-4}) & \omega_{\ell-3} & \omega_{\ell-4} \\ & \ldots & & & \\ & s(\omega_3) & s(\omega_2) & \omega_3 & \omega_2 \\ & s(\omega_1) & s(\omega_0) & \omega_0 & 6_{16} \end{array}$$

6.5.2 Attaque sur une version modifiée de ISO/IEC-9796-1

On décrit dans un premier temps une attaque qui s'applique à une variante du schéma ISO 9796-1, dans laquelle $\tilde{s}(x)$ est remplacé par $s(x)$; cette variante notée $\mu'(m)$ diffère de la fonction $\mu(m)$ de ISO 9796-1 par un seul bit.

On note a_j un groupe de 4 bits et on considère un message de la forme:

$$\begin{array}{rl} m_i = & a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ 66_{16} \\ & a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ 66_{16} \\ & \ldots \\ & a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ 66_{16} \end{array}$$

dont la fonction associée est:

$$\begin{array}{rcccccccc} \mu'(m_i) = & \bar{s}(a_6) & s(a_5) & a_6 & a_5 & s(a_4) & s(a_3) & a_4 & a_3 \\ & s(a_2) & s(a_1) & a_2 & a_1 & 2_{16} & 2_{16} & 6_{16} & 6_{16} \\ & \ldots & & & & & & & \\ & s(a_6) & s(a_5) & a_6 & a_5 & s(a_4) & s(a_3) & a_4 & a_3 \\ & s(a_2) & s(a_1) & a_2 & a_1 & 2_{16} & 2_{16} & 6_{16} & 6_{16} \end{array}$$

En restreignant notre choix de a_6 aux 8 cas possibles tels que $s = \bar{s}$, on peut générer 2^{23} nombres de la forme

$$\mu'(m_i) = x \cdot \Gamma_{23}$$

où x est un entier de huit octets:

$$x = s(a_6) \cdot s(a_5) \cdot a_6 \cdot a_5 \cdot s(a_4) \cdot s(a_3) \cdot a_4 \cdot a_3 \cdot s(a_2) \cdot s(a_1) \cdot a_2 \cdot a_1 \cdot 2266_{16}$$

et avec la constante Γ_{23} donnée par:

$$\Gamma_{23} = \sum_{i=0}^{\gamma/32-1} 2^{64i}$$

Comme x est un petit entier (seulement 64 bits), la probabilité qu'il soit lisse n'est pas négligeable. Pour un entier x de 64 bits, la probabilité qu'il soit 2^{16}-lisse est approximativement $2^{-7.7}$, donc on peut s'attendre à trouver $2^{23} \cdot 2^{-7.7} > 2^{15}$ entiers x qui soient 2^{16}-lisses. Comme il y a approximativement 2^{12} nombres premiers inférieurs à 2^{16}, on peut donc espérer obtenir suffisamment d'entiers x qui soient 2^{16}-lisses pour pouvoir exprimer l'un des $\mu'(m_i)$ comme combinaison multiplicative des autres. Dans le paragraphe suivant, on donne une application pratique de l'attaque nécessitant seulement 181 messages. On voit que l'attaque s'applique à tout module de taille $64 \cdot \alpha + 1$ bits et que la complexité de l'attaque est indépendante de $\alpha \in \mathbb{N}$.

6.5.3 Application pratique: attaque sur la version modifiée

On décrit ici une attaque contre la variante du standard ISO 9796-1 qui s'applique à tout module de 1025-bits, avec un exposant $e = 3$.

1 : Pour tout module RSA de 1025 bits, former les 180 messages:

$$m_i = (256 \times \texttt{message}[i]_{16} + 102) \times \sum_{j=0}^{11} 2^{32 \cdot j}$$

où `message[`i`]` sont les éléments de la table suivante:

00014E	008C87	00D1E8	01364B	0194D8	01C764	021864	03442F	0399FB	048D9E	073284	0863DE	09CCE8
0A132E	0A2143	0BD886	0C364A	0C368C	0C6BCF	0D3AC1	0D5C02	0EA131	0F3D68	0F9931	31826A	31BE81
31ED6B	31FCD0	320B25	32B659	332D04	3334D8	33EAFC	33EB1D	343B49	353D02	35454C	35A1A9	36189E
362C79	365174	3743AB	3765F6	37C1E2	3924AC	3998A8	3AF8A7	3B6900	3B9EEB	3BC1FF	3DE2DE	3E51BE
3E8191	3F49F3	3F69AC	4099D9	40BF29	41C36C	41D8C0	424EE8	435DB7	446DC1	4499CC	44AA20	44EE53
4510E8	459041	45A464	45AA03	460B80	4771E7	486B6A	499D40	4A5CF8	4AC449	4ADA0A	4B87A8	4C06A1
4C5C17	4D4685	4E39EA	4EB6B6	4F8464	716729	71C7D3	71FA22	722209	72DBF1	7619AB	765082	767C39
76885C	78F5F3	79E412	79FAD6	7CD0ED	7D0ABA	7DBA1D	7DE6A5	7E06A2	7EA5F2	7EC1ED	7EEC78	90BB4B
90DE38	9139D7	934C2C	9366C5	941809	941BFB	947EB4	94DB29	952D45	9745BD	978897	97A589	9827AF
984FAC	9A193D	9A83E2	9B74E3	9BEAE9	9C704F	9DBA98	9F9337	A00D15	A02E3D	A10370	A429A6	A4DADD
A4F689	A5485D	A6D728	A76B0F	A7B249	A87DF3	A95438	A96AA4	AB1A82	AD06A8	AEA0D0	AEB113	D076C5
D13F0E	D18262	D1B0A7	D35504	D3D9D4	D3DEE4	D4F71B	D91C0B	D96865	DA3F44	DB76A8	DE2528	DE31DD
DE46B8	DE687D	DEB8C8	DF24C3	DFDFCF	DFF19A	E12FAA	E1DD15	E27EC1	E39C56	E40007	E58CC8	E63CE0
E6596C	E7831E	E796FB	E7E80C	E85927	E89243	E912B4	E9BFFF	EA0DFC	EACF65	EB29FA		

2 : Construire le message $m' = \texttt{EE7E8E66}_{16} \times \sum_{j=0}^{11} 2^{32j}$.

3 : On a:

$$\mu'(m') = \prod_{i=0}^{345} p_i^{-e \cdot \texttt{gamma}[i]} \prod_{i=1}^{180} s_i^{\texttt{beta}[i]} \bmod N$$

où p_i est le i-ème premier (avec $p_0 = \Gamma_{23}$) et `beta[`i`]` dénote les éléments de la table suivante:

1	2	1	2	2	2	2	1	2	2	2	1	1	2	2	2	1	1	2	1	2	2	2	
2	2	1	1	2	1	1	2	1	1	2	1	1	1	1	1	1	1	1	2	1	1	1	
1	2	1	1	1	1	2	2	1	1	2	1	2	1	1	2	2	1	1	1	1	2	1	
1	2	1	1	1	1	1	2	2	1	2	1	2	1	1	1	1	1	1	2	2	2	2	
2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	2	1	1	1	2	2	2	2	
1	2	2	1	1	2	2	2	2	1	1	2	1	2	2	2	2	1	1	1	2	1	1	
2	1	1	1	1	2	2	1	1	1	1	2	2	1	2	2	1	1	2	2	2	1	2	
1	2	2	2	1	2	1	2	2	1	2	1	2	2	1	1	2	1	1					

`gamma`$[i]$ représente les valeurs hexadécimales suivantes:

57	57	68	33	27	18	16	13	10	0F	0E	0B	09	09	0D	05	0B	07	04	08	07	07
07	09	0A	03	07	04	05	05	03	04	03	01	02	03	04	03	01	03	03	03	02	06
03	03	04	06	02	04	04	02	02	03	02	04	04	03	04	01	04	03	02	03	02	01
02	02	01	03	01	01	01	01	03	03	01	03	02	02	01	04	02	04	02	02	01	02
01	01	01	03	03	01	02	01	01	00	03	02	03	01	01	02	01	02	02	03	03	04
03	03	02	03	01	02	03	02	01	03	02	02	01	01	00	02	01	01	03	01	01	01
01	01	02	00	02	00	00	01	02	01	01	01	00	01	01	00	01	01	02	02	01	01
01	00	01	00	01	01	04	02	02	02	01	02	02	01	02	01	02	00	01	00	02	01
02	02	00	01	02	01	01	01	02	01	01	01	02	01	00	01	01	00	00	01	02	00
01	00	01	01	00	01	00	01	02	02	01	01	02	00	00	02	01	02	02	01	00	00
01	00	01	00	01	00	02	00	00	00	01	01	00	00	01	01	00	00	00	01	00	00
00	00	00	00	01	01	00	00	01	02	01	01	01	00	01	02	01	01	01	02	00	00
00	01	01	00	01	00	00	00	02	02	01	00	01	02	00	01	00	01	02	00	01	00
00	01	00	01	01	01	00	01	01	00	01	01	01	01	00	00	01	01	00	00	01	01
00	01	01	00	00	01	00	00	00	01	01	02	02	01	01	00	00	01	02	01	02	00
01	01	00	01	00	00	00	00	00	00	01	00	00	01	02	01						

6.5.4 Attaque sur le vrai standard ISO 9796-1

L'attaque précédente ne s'applique pas au vrai standard ISO 9796-1. La différence entre la fonction $\mu(m)$ du standard ISO 9796-1 et la fonction modifiée $\mu'(m)$ est que le bit de poids faible du second groupe de 4 bits le plus significatif de $\mu'(m)$ est inversé. Ainsi, on ne peut pas représenter simplement $\mu(m)$ comme le produit $\Gamma \cdot x$ avec Γ et x tels que précédemment.

Mais l'attaque a été étendue par Coppersmith, Halevi et Jutla [21] pour l'adapter au vrai standard ISO 9796-1. La nouvelle attaque est similaire à la précédente, la différence étant qu'elle utilise pour les entiers Γ et x une structure légèrement différente. Dans l'attaque précédente, la constante Γ était constituée de plusieurs uns séparés par autant de zéros qu'il y a de bits dans x. On parvenait ainsi à répliquer plusieurs fois dans $\mu'(m)$ la même structure donnée par x. Dans la nouvelle attaque, on utilise encore une constante Γ constituée de plusieurs uns séparés par des zéros, mais le nombre de zéros est plus petit, ce qui fait que la partie haute de x s'ajoute à la partie basse de x.

Considérons l'exemple suivant. Soit un entier x de 64 bits, représenté comme 4 mots de 16 bits $abcd$, avec a le mot de poids le plus fort et d le mot de poids le plus faible. Soit la constante de 97 bits $\Gamma = 1001001$, où chaque chiffre représente un mot de 16 bits. En multipliant Γ et x, on obtient:

						a	b	c	d	
			1	0	0	1	0	0	1	
						a	b	c	d	
			a	b	c	d				
a	b	c	d							
a	b	c	e	b	c	e	b	c	d	

où $e = a + d$, et en supposant que l'addition $a + d$ ne génère pas de retenue. On observe que le mot de 16 bits d n'apparaît que dans le mot de poids le plus faible du résultat, tandis que a n'apparaît que dans le mot de poids le plus fort. Il est ainsi possible d'obtenir des mots a et d différents de b, c et e, de façon à respecter les formes particulières des mots de poids faible et de poids fort de $\mu(m)$.

Plus précisément, on dit qu'un mot x de 16 bits est un *mot bas valide* si x est de la forme:

$$x = s(u)\, s(v)\, v\, 6$$

On dit qu'un mot x de 16 bits est un *mot milieu valide* si x est de la forme:

$$x = s(u)\, s(v)\, u\, v$$

On dit qu'un mot x de 16 bits est un *mot haut valide* si x est de la forme:

$$x = \bar{s}(u)\, \tilde{s}(v)\, u\, v$$

On voit qu'il existe 256 mots bas valides, 256 mots milieu valides, et 256 mots haut valides. Dans l'exemple précédent, il faut que a soit un mot haut valide, d un mot bas valide, b et c des mots milieu valides, et aussi que $e = a + d$ soit un mot milieu valide. On peut vérifier qu'il existe 64 couples de valeurs x, y telles que x est un mot haut valide, y un mot bas valide, et $z = x + y$ un mot milieu valide. On appelle un tel (x, y) un *couple haut-bas*.

Pour attaquer le standard ISO 9796-1 avec un module de taille 1025 bits, on procède donc de la façon suivante: on considère un entier x de 64 bits $x = abcd$, où a est un mot haut valide, d un mot bas valide, et b, c et $e = a + d$ sont des mots milieu valides. Le nombre d'entiers x possibles est donc $64 \cdot 256^2 = 2^{22}$. On définit alors:

$$\Gamma = \sum_{i=0}^{20} 2^{48 \cdot i}$$

ce qui donne

$$M = \Gamma \cdot x = a\ bce\ bce\ \ldots\ bce\ bcd$$

qui est un codage valide pour un message m tel que $M = \mu(m)$. Comme la probabilité qu'un entier x soit 2^{16}-lisse est approximativement $2^{-7.7}$, on peut espérer trouver $2^{22} \cdot 2^{-7.7} > 2^{14}$ entiers x convenables qui soient 2^{16}-lisses. Comme il y a approximativement 2^{12} premiers inférieurs à 2^{16}, on peut donc espérer trouver suffisamment d'entiers x pour obtenir une dépendance linéaire dans la factorisation des $\mu(m_i)$. Dans le paragraphe suivant, on applique cette attaque en utilisant seulement 273 messages.

6.5.5 Application pratique: attaque sur ISO 9796-1

Dans ce paragraphe, on donne un exemple concret de forge sur ISO 9796-1 pour un module de 1025 bits avec un exposant public $e = 3$.

Etape 1 : Soit N un module RSA de 1025 bits. On forme les entiers x_i de 64 bits suivants, pour i compris entre 1 et 273.

$$\begin{aligned} a_i &= \bar{s}(u_{i,1})\,\tilde{s}(u_{i,2})\,u_{i,1}\,u_{i,2} \\ b_i &= s(u_{i,3})\,s(u_{i,4})\,u_{i,3}\,u_{i,4} \\ c_i &= s(u_{i,5})\,s(u_{i,6})\,u_{i,5}\,u_{i,6} \\ d_i &= s(u_{i,7})\,s(u_{i,8})\,u_{i,8}\,6 \\ x_i &= a_i\,b_i\,c_i\,d_i \end{aligned}$$

où message$[i] = u_{i,1}\,u_{i,2}\,u_{i,3}\,u_{i,4}\,u_{i,5}\,u_{i,6}\,u_{i,7}\,u_{i,8}$ est donné dans la table suivante pour i compris entre 1 et 273:

113C2789	2103E5FE	213488FE	215041FE	21A1F6FE	23979965	23A9DF65	26013565	26182D65	261B3865
26235B65	26729D65	26EB1465	30157C81	3038C281	304D5B81	30CF6581	34045BF1	340AC4F1	34596BF1
34B660F1	34E1B0F1	34FF49F1	3814BA6A	38585D6A	3873976A	38A9396A	38E2F86A	38EEE56A	385192BD
3854A9BD	3882F7BD	389E88BD	38BB52BD	3A16E425	3A3C6125	3A797525	3A9B4E25	3AB30125	3ABFBC25
3AD30A25	3D12D3F9	3D6C4AF9	3D8AF3F9	3D91E4F9	3D9E3BF9	3DD521F9	3DE363F9	3DEDAFF9	3F09D025
3F198D25	3F3DFC25	3FCE9B25	410AB2F9	4122BDF9	412F08F9	413EDBF9	41C584F9	41EE50F9	41F296F9
4345DC55	43486155	4372C655	43793F55	4385E655	43EE7B55	4617F255	4627D755	463CF255	4665D455
468AA555	46DB9055	484B4E1A	488ED71A	48E4B91A	48EE6D1A	4A55A165	4A6F6565	4A77DA65	4A905D65
4AC74265	4AEE8465	4D069469	4D147369	4D31AB69	4D420C69	4D499369	4D532169	4D56A869	4D758769
4D84EE69	4DD22969	4F2BF565	4F2C2665	4F758F65	4FA5A565	4FD7BD65	51C43089	51DA7A89	51E7E789
590CC262	59733762	59F54062	5B07E9FA	5B9EFDFA	5BBC4BFA	5BDC93FA	5BFCCEFA	5E062FFA	5E157DFA
5E4550FA	5E7CB6FA	5E963AFA	5ED3F8FA	6015AF51	60326151	60372751	604F6B51	60708951	607F0B51
60931F51	60D7FF51	6297391A	6486D321	6496D721	64F0D121	6758901A	675ED11A	67F7F31A	6C3FB8F7
6C9916F7	6CAA47F7	6CD886F7	806BD551	806F2D51	80A83051	831D3465	833A6E65	837B2565	837F0865
83B16265	83DA9C65	840FAF21	84149621	84704721	84802A21	84A25A21	84F1E221	84FDA321	858D66B8
85EB0BB8	861A4765	8634B865	866AB865	868D6165	86AC2F65	891EF962	89220762	892C2662	893ABD62
8950EA62	89CFD062	89DA4562	8A049B55	8A27EF55	8A32DF55	8A489755	8A523055	8A7F9955	8AB3CA55
8AD3AD55	8AF88555	8DA35BBE	8DC6B0BE	8DDAC3BE	8F1F7855	8F5F5F55	8FC42755	8FEC2655	913BD36E
9158BF6E	9199DF6E	91B4856E	91D1546E	91E5696E	A0B92266	A0BA2B66	A4401E16	A4DFFF16	A4ED5A16
A4F64416	A8668A5D	AD0C6EFE	AD8124FE	ADB3D7FE	ADC5A6FE	ADDAF5FE	D00806F1	D07D68F1	D0D26DF1
D0DDC2F1	D20C395A	D25CE85A	D278785A	D2B6C25A	D2BF0D5A	D2E44D5A	D400B761	D41E1961	D4732D61
D494FC61	D4A85061	D79B1B5A	D79FAA5A	D801D7FD	D815D2FD	D868D1FD	D8F292FD	EA43E961	EA485761
EA4E1261	EB355C8A	EB37F78A	EB73DA8A	EED7308A	EEDBF58A	EEE9118A	EF784561	EF7CB861	EF8FDE61
F10F04FE	F146DAFE	F18C0CFE	F196ACFE	F1B831FE	F1CFA5FE	F1D371FE	F269861A	F26A251A	F28A8D1A
F32E2E21	F3369421	F3EB6821	F52952B8	F55C47B8	F5CC08B8	F6202521	F64ABA21	F6683921	F684CE21
F6DE0521	F6F67621	F7BDBD1A	F7D0F01A	F7D2411A	F7F60F1A	FB6E9AFA	FBA2B8FA	FBF809FA	FC8BA450
FCBC2050	FCD65150	FCEFE550	FD705E6E	FDBACE6E	FDE3756E	FE0395FA	FE0F38FA	FE0FABFA	FE2ECFFA
FE56C3FA	FE9C2EFA	FEEFA7FA							

On obtient ainsi $M_i = \Gamma \cdot x_i$ qui est un codage valide pour un certain message m_i tel que $M_i = \mu(m_i)$.

Etape 2 : Obtenir du signataire les 272 signatures $s_i = \mu_{\mathrm{ISO}}(m_i)^d \bmod N$ pour i compris entre 1 et 272.

Etape 3 : La signature du message m_{273} est donnée par:

$$\mu(m_{273})^d = \Gamma^{-139} \prod_{i=1}^{587} p_i^{-\mathtt{g}[i]} \prod_{i=1}^{272} s_i^{\mathtt{b}[i]} \bmod N$$

où p_i est le i-ème nombre premier et $\mathtt{b}[i]$ est donné par la table suivante:

2	2	1	2	1	2	2	2	2	1	2	2	2	1	1	1	2	1	2	1	1	2	2	1	1	1
2	2	2	1	2	1	1	2	2	1	2	1	1	2	1	2	2	2	1	2	2	2	2	1	2	2
1	2	1	1	1	2	2	1	1	2	1	2	2	2	2	1	2	1	2	2	2	2	2	1	1	1
1	1	1	1	2	1	1	2	1	2	2	2	1	2	1	1	1	2	1	1	2	1	2	2	2	1
1	1	2	1	1	2	1	1	2	2	1	1	2	1	1	1	2	1	2	2	2	2	2	1	2	2
1	2	2	2	1	1	2	2	1	2	1	1	1	1	2	2	2	2	1	1	2	2	1	2	1	2
2	2	2	1	1	2	1	2	2	2	1	1	1	1	1	2	2	1	1	1	2	1	2	2	2	2
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	2	1	2	1	2	1	2	2
2	1	2	1	2	2	2	1	2	1	1	2	1	1	2	2	2	1	2	2	2	2	2	1	2	2
2	1	2	2	2	1	1	2	2	2	1	1	1	2	2	1	2	2	1	2	1	2	1	1	1	2
2	1	1	1	1	1	1	1	2	1	2	2														

et $\mathtt{g}[i]$ est donnée par la table suivante:

8B	89	4F	3D	20	25	1D	14	14	13	11	0F	10	0B	0D	0B	0A	0B	07	08
09	07	0B	08	0B	07	05	04	08	08	05	04	08	01	07	04	07	04	02	04
0A	05	07	07	06	05	05	04	03	05	03	04	05	04	03	04	05	05	03	04
02	03	03	02	02	02	02	02	03	02	02	02	02	01	01	02	04	05	02	02
06	04	02	01	01	04	01	02	02	01	04	03	02	02	01	02	01	02	03	02
00	02	02	02	03	02	01	01	02	03	04	03	02	02	02	02	02	01	01	02
02	05	00	00	01	01	03	01	02	02	00	01	01	02	01	00	02	03	02	01
02	02	01	01	02	02	01	02	01	03	01	00	01	01	02	01	01	02	00	02
02	00	02	00	02	01	02	01	03	01	01	01	01	03	02	00	01	01	02	02
00	01	02	01	00	01	01	01	01	01	01	01	02	01	01	01	02	01	03	02
02	01	01	01	03	03	01	00	00	01	01	02	01	01	01	01	02	02	02	01
02	01	00	01	01	00	01	02	01	02	00	01	01	02	00	04	02	01	01	01
00	02	00	01	00	00	01	00	01	00	01	01	00	00	01	00	03	00	01	00
02	03	02	01	01	01	01	01	00	02	01	02	00	00	02	02	00	01	00	01
02	02	02	01	00	01	01	02	00	02	01	02	00	01	00	00	02	01	01	01
01	01	00	01	00	01	01	02	00	01	02	00	01	03	02	00	00	02	00	01
01	00	02	00	00	00	01	00	01	01	00	01	00	01	01	00	02	01	01	00
02	00	00	00	01	01	01	02	01	01	00	00	00	00	01	01	01	00	01	01
02	02	01	01	01	01	01	00	00	01	00	00	00	01	01	01	01	00	01	00
00	01	00	00	00	02	02	00	01	00	00	00	01	01	00	00	00	02	02	00
00	00	00	01	00	00	01	00	00	00	01	01	01	00	01	02	00	01	00	00
01	01	01	01	00	01	01	01	00	00	01	01	00	00	01	00	01	00	01	01
01	00	01	00	01	00	02	00	01	00	01	00	02	01	00	00	01	00	00	00
00	00	02	01	00	00	00	01	00	00	00	00	00	00	03	00	00	01	00	00
00	01	00	00	01	02	00	00	01	00	02	00	00	00	00	02	00	01	00	00
00	00	00	00	01	01	01	00	00	01	02	00	00	00	00	01	00	00	01	00
00	00	00	00	01	01	00	01	00	00	00	01	00	01	00	00	00	01	00	00
01	01	00	00	00	00	00	01	00	01	01	00	00	01	00	01	00	00	00	00
01	01	02	00	00	00	00	01	00	00	00	01	00	01	01	00	00	00	01	00
01	00	00	01	00	02	00													

6.5.6 L'attaque de Grieu sur ISO 9796-1

Lors de la conférence Eurocrypt 2000, François Grieu a présenté une attaque beaucoup plus efficace sur ISO 9796-1[54]. Le principe de son attaque consiste à chercher tous les messages m, m' tels que:

$$\frac{\mu(m)}{\mu(m')} = \frac{a}{b}$$

pour des entiers a, b petits. En obtenant deux paires de messages m_1, m_1' et m_2, m_2' solutions de l'équation précédente, on obtient 4 messages tels que:

$$\mu(m_1) \cdot \mu(m_2') = \mu(m_1') \cdot \mu(m_2)$$

ce qui permet d'exprimer la signature du message m_1 en fonction de la signature des message m_1', m_2 et m_2'.

6.6 Sécurité des signatures ISO 9796-2

ISO 9796-2 est un standard de signature permettant de retrouver totalement ou une partie seulement du message. Le standard permet l'utilisation de fonctions de hachage de taille variable L. Le paragraphe 5, note 4 du standard [4] recommande de prendre $64 \leq L \leq 80$ lorsqu'on retrouve totalement le message et $128 \leq L \leq 160$ lorsqu'on retrouve partiellement le message.

6.6.1 Inclusion partielle du message

On commence par décrire l'attaque dans le cas où une partie seulement du message est retrouvée lors de la vérification de la signature. Pour simplifier, on suppose que le nombre de bits $|N|$ du module N est un multiple de 8, ainsi que L et la taille du message m. Le message

$$m = m[1] \| m[2]$$

est séparé en deux parties où $m[1]$ est constitué des $|N| - L - 16$ bits les plus significatifs et $m[2]$ de tous les autres bits de m. La fonction de padding est alors:

$$\mu(m) = \mathtt{6A}_{16} \| m[1] \| \mathrm{HASH}(m) \| \mathtt{BC}_{16}$$

et le message $m[2]$ est transmis avec la signature.

En divisant $(\mathtt{6A}_{16} + 1) \cdot 2^{|N|}$ par N on obtient:

$$(\mathtt{6A}_{16} + 1) \cdot 2^{|N|} = i \cdot N + r \quad \text{avec} \quad r < N < 2^{|N|}$$

On définit N' tel que:

$$N' = i \cdot N = \mathtt{6A}_{16} \cdot 2^{|N|} + (2^{|N|} - r) = \mathtt{6A}_{16} \| N'[1] \| N'[0]$$

où N' est de taille $|N| + 7$ bits et $N'[1]$ est de taille $|N| - L - 16$ bits. En prenant $m[1] = N'[1]$ on obtient:

$$t = i \cdot N - \mu(m) \cdot 2^8 = N'[0] - \mathrm{HASH}(m) \| \mathtt{BC00}_{16}$$

et la taille de t est inférieure à $L + 16$ bits.

L'attaquant modifie ainsi $m[2]$ (et donc $\mathrm{HASH}(m)$) jusqu'à ce qu'il trouve un nombre suffisant de messages dont les valeurs de t correspondantes se factorisent en produit de petits nombres premiers, de façon à pouvoir exprimer l'un des $\mu(m_i)$ comme combinaison multiplicative des autres. On remarque que la complexité de l'attaque est indépendante de la taille de N, mais que contrairement aux attaques précédentes sur ISO 9796-1, les messages forgés dépendent de N et ne peuvent pas être réutilisés pour un module N différent.

On obtient les complexités approximatives suivantes en temps et en espace, obtenues comme dans le paragraphe 6.4. Même si ces complexités sont inférieures à celles correspondant à une attaque par paradoxe des anniversaires sur la fonction de hachage, on peut considérer que pour $L = 160$ on conserve un niveau de sécurité raisonnable.

L	$\log_2$ **temps**	$\log_2$ **espace**
128	44	22
160	49	25

Tableau 6.2. Complexité des attaques sur ISO 9796-2 avec inclusion partielle du message.

6.6.2 Inclusion totale du message

On suppose encore que la taille du module N et la taille L de la fonction de hachage sont des multiples de 8. On suppose que la taille du message m est $|N| - L - 16$. Dans ce cas, la fonction de padding est donnée par:

$$\mu(m) = \mathtt{4A}_{16} \| m \| \mathrm{HASH}(m) \| \mathtt{BC}_{16}$$

On sépare le message $m = m[1] \| m[0]$ en deux parties, où $m[0]$ est constituée des ℓ bits les moins significatifs de m et $m[1]$ du reste. Le paramètre ℓ sera déterminé par la suite. On calcule comme dans le cas précédent, un entier i tel que:

$$N' = i \cdot N = \mathtt{4A}_{16} \| N'[1] \| N'[0]$$

où $N'[0]$ est de taille $(L + \ell + 16)$ bits et $N'[1] \| N'[0]$ est de taille $|N|$ bits.

En prenant $m[1] = N'[1]$, on obtient:

$$t = i \cdot N - \mu(m) \cdot 2^8 = N'[0] - m[0] \| \mathrm{HASH}(m) \| \mathtt{BC00}_{16}$$

où la taille de t est inférieure à $L + \ell + 16$ bits.

L'attaquant modifie $m[0]$ (et donc $\mathrm{HASH}(m)$) de façon à obtenir suffisamment de messages dont la valeur de t correspondante se factorise en produit de petits premiers. Le paramètre ℓ doit être fixé de façon à trouver suffisamment d'entiers t qui soient lisses. On obtient ainsi les complexités en temps suivantes, qui suggèrent une révision du standard.

L	$\log_2$ **temps**	$\log_2$ **espace**	ℓ
64	35	18	35
80	39	20	39

Tableau 6.3. Complexité des attaques sur ISO 9796-2 avec inclusion totale du message.

6.7 Conclusion

Nous avons décrit une extension de l'attaque de Desmedt et Odlyzko sur les schémas de signature RSA. Cette extension permet de construire une attaque pratique sur le standard de signature ISO 9796-1. Les attaques sur le standard ISO 9796-2 sont

moins efficaces, mais montrent que le standard devrait être modifié dans le cas de l'inclusion totale du message dans la signature. Ces attaques montrent le danger de l'utilisation d'un schéma de signature pour lequel il n'existe pas de preuve de sécurité. Nous étudierons dans les chapitres suivants ce qu'on entend par preuve de sécurité pour un schéma de signature numérique.

Partie IV

Preuves de sécurité pour schémas de signature

7. Introduction aux preuves de sécurité pour les schémas de signature numérique

7.1 Introduction

Depuis l'invention de la cryptographie à clef publique par Diffie et Hellman [40], de nombreux schémas de chiffrement ou de signature ont été proposés, et beaucoup d'entre eux se sont révélés vulnérables. La recherche de schémas qui soient à la fois efficaces dans la pratique et qui possèdent une preuve de sécurité est donc un domaine de recherche important. Une preuve de sécurité est généralement une réduction calculatoire entre la résolution d'un problème mathématique bien connu et jugé difficile, et une attaque de la sécurité du schéma. Autrement dit, on montre qu'à partir d'une attaque sur le schéma de chiffrement ou de signature, on peut construire un algorithme permettant de résoudre le problème jugé difficile. On obtient ainsi une garantie sur la sécurité du schéma lui-même. Comme exemples de problèmes mathématiques jugés difficiles, on peut citer la factorisation de grands entiers, le calcul de logarithmes discrets dans certains groupes, ou l'extraction d'une racine e-ème modulo un entier composite. La sécurité de RSA [75] est basée sur ce dernier problème.

La notion de sécurité la plus forte pour les schémas de signature numérique a été définie par Goldwasser, Micali et Rivest dans [52]: il doit être impossible pour un attaquant de produire une signature valide d'un message quelconque, même en ayant la possibilité d'obtenir la signature de messages de son choix.

Goldwasser, Micali et Rivest ont proposé dans [52] un schéma de signature basé sur des arbres de signature, qui remplit la condition précédente. L'efficacité de ce schéma a ensuite été améliorée par Dwork et Naor dans [42], et par Cramer et Damgård dans [35]. Le principal inconvénient de ces schémas de signature est que la signature d'un message dépend des messages précédemment signés: le signataire doit stocker de l'information au fur et à mesure qu'il génère des signatures.

Cramer et Shoup ont présenté dans [36] un schéma de signature prouvé sûr contre les attaques à clair choisi adaptatives, qui est à la fois sans mémoire et plus efficace. La sécurité du schéma est basée sur le problème RSA fort. Gennaro, Halevi et Rabin ont présenté dans [48] un nouveau schéma de signature de type hache-et-signe, prouvé sûr contre les attaques à clair choisi adaptatives, également sans mémoire, efficace, et basé sur le problème RSA fort. On dit qu'un schéma de signature est de type hache-et-signe si le message est d'abord haché en utilisant une fonction de

hachage, dont le résultat est ensuite signé en utilisant un schéma de signature comme RSA.

Le modèle de l'oracle aléatoire, introduit par Bellare et Rogaway dans [7], permet de prouver la sécurité de schémas de signature de la famille hache-et-signe. Dans ce modèle, la fonction de hachage est remplacée par un oracle qui renvoie une valeur aléatoire à chaque nouvelle requête. Bellare et Rogaway ont montré dans [7] comment construire un schéma de signature prouvé sûr, à partir de n'importe quelle permutation à sens unique à trappe. Ils ont défini dans [9] le schéma de signature *Full Domain Hash* (FDH), qui est prouvé sûr dans le modèle de l'oracle aléatoire en supposant qu'inverser RSA est difficile, et le schéma *Probabilistic Signature Scheme* (PSS), qui a une meilleure preuve de sécurité que le schéma FDH. De même, David Pointcheval et Jacques Stern ont prouvé dans [72] dans le modèle de l'oracle aléatoire la sécurité de certains schémas de signature basés sur le problème du logarithme discret.

Cependant, une preuve dans le modèle de l'oracle aléatoire ne constitue pas une vraie preuve de sécurité, parce que dans la pratique la fonction de hachage n'est pas un oracle aléatoire: c'est une fonction bien définie que l'attaquant peut calculer lui même; Canetti, Goldreich et Halevi ont montré dans [17] qu'une preuve de sécurité dans le modèle de l'oracle aléatoire n'implique par forcément que le schéma est sûr dans le monde réel.

Dans ce chapitre, nous rappelons la définition d'un schéma de signature, et comment on formalise la sécurité d'un schéma de signature.

7.2 Schéma de signature

La signature numérique d'un message est une chaîne de bits qui dépend du message et d'un secret que seul le signataire connaît. Une signature numérique doit être vérifiable: n'importe qui doit pouvoir vérifier la validité de la signature. Les définitions suivantes sont basées sur [52].

Définition 7.2.1 (schéma de signature). *Un schéma de signature est défini de la façon suivante:*

- L'algorithme de génération de clef `Gen` *est un algorithme probabiliste qui étant donné* 1^k*, renvoie une paire* (pk, sk) *de clef publique* pk *et privée* sk *correspondantes.*

- L'algorithme de signature `Sign` *prend en entrée un message* M *et la clef secrète* sk *et retourne la signature* $x = \texttt{Sign}_{sk}(M)$*. L'algorithme de signature peut être probabiliste.*

- L'algorithme de vérification `Verify` *prend en entrée un message* M*, la signature candidate* x' *et la clef publique* pk*. Il renvoie un bit noté* $\texttt{Verify}_{pk}(M, x')$*, égal à* 1 *si la signature est acceptée, et* 0 *sinon. On requiert que si* $x \leftarrow \texttt{Sign}_{sk}(M)$*, alors* $\texttt{Verify}_{pk}(M, x) = 1$.

La plupart des schémas de signature utilisent une fonction de hachage. Dans ce qui suit, on remplace la fonction de hachage par un oracle aléatoire: la sortie de la fonction de hachage est uniformément distribuée dans l'espace d'arrivée de h. Bien sûr, si la même entrée est requise deux fois, l'oracle de hachage renvoie deux fois la même réponse.

7.3 Sécurité d'un schéma de signature

La sécurité des schémas de signature a été formalisée dans un cadre asymptotique par Goldwasser, Micali et Rivest dans [52]. Dans ce qui suit, nous reprenons les définitions de [7] et [9], qui prennent en compte la présence de l'oracle aléatoire, et donnent une analyse concrète (*i.e.* pour une taille de paramètre donnée) de la sécurité d'un schéma de signature. On considère ici la résistance contre les attaques à message choisi adaptatives : un forgeur $\mathcal{F}$ peut obtenir la signature de messages de son choix, et son but est de renvoyer une forge valide. Une *forge valide* est une paire message/signature (M, x) telle que $\texttt{Verify}_{pk}(M, x) = 1$, mais la signature de M n'a jamais été requise par $\mathcal{F}$.

Définition 7.3.1. *On dit qu'un forgeur $\mathcal{F}$ $(t, q_{hash}, q_{sig}, \epsilon)$-casse le schéma de signature $(Gen, Sign, Verify)$ si après au plus $q_{hash}(k)$ requêtes de hachage à l'oracle aléatoire, $q_{sig}(k)$ requêtes de signature et un temps de calcul $t(k)$, il renvoie une forge valide avec probabilité au moins $\epsilon(k)$ pour tout $k \in \mathbb{N}$.*

Définition 7.3.2. *Un schéma de signature $(Gen, Sign, Verify)$ est $(t, q_{hash}, q_{sig}, \epsilon)$-sûr s'il n'existe pas de forgeur qui $(t, q_{hash}, q_{sig}, \epsilon)$-casse le schéma de signature.*

7.4 Preuve de sécurité d'un schéma de signature

La sécurité d'un schéma de signature est le plus souvent basée sur l'existence d'un problème difficile à résoudre. Pour prouver la sécurité d'un schéma de signature, on montre que s'il existe un forgeur qui casse le schéma, on peut utiliser ce forgeur pour résoudre le problème difficile. C'est ce qu'on appelle *réduire* le problème difficile à casser le schéma de signature.

Le problème difficile à résoudre peut être par exemple le problème consistant à inverser RSA, c'est à dire à calculer la racine e-ème modulo N d'un entier y donné. Plus précisément, on définit le cryptosystème RSA de la façon suivante:

Définition 7.4.1 (Le cryptosystème RSA). *Le cryptosystème RSA est une famille de permutations à sens unique à trappe, spécifiée par:*

- Le générateur $\mathcal{RSA}$, qui prenant en entrée 1^k, sélectionne aléatoirement deux nombre premiers p et q distincts de taille $k/2$-bits, et calcule le module $N = p \cdot q$. Il génère aléatoirement un exposant de chiffrement e dans $\mathbb{Z}^*_{\phi(N)}$ et calcule l'exposant de déchiffrement correspondant tel que

$$e \cdot d = 1 \bmod \phi(N)$$

Le générateur renvoie (N, e, d).

- *La fonction de chiffrement* $f : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ *définie par:*

$$f(x) = x^e \bmod N$$

- *La fonction de déchiffrement* $f^{-1} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ *définie par*

$$f^{-1}(y) = y^d \bmod N$$

Pour quantifier la sécurité du cryptosystème RSA, on considère un algorithme inverseur pour RSA. Un *algorithme inverseur* pour RSA prend en entrée (N, e, y) et renvoie en sortie $f^{-1}(y)$. Sa probabilité de succès est la probabilité qu'il renvoie $f^{-1}(y)$ lorsque N, e, d sont obtenus en exécutant $\mathcal{RSA}(1^k)$ et y est généré aléatoirement dans $\mathbb{Z}_N^*$.

Définition 7.4.2. *On dit qu'un algorithme inverseur* $\mathcal{I}$ (t, ϵ)*-casse RSA si après un temps de calcul d'au plus* $t(k)$ *sa probabilité de succès est au moins* $\epsilon(k)$ *pour tout* $k \in \mathbb{N}$.

Définition 7.4.3. *On dit que RSA est* (t, ϵ)*-sûr s'il n'existe pas d'inverseur qui* (t, ϵ)*-casse RSA.*

7.5 Conclusion

Nous avons rappelé la façon dont on formalise la sécurité d'un schéma de signature numérique. Dans les chapitres suivants, nous étudierons plus en détail les preuves de sécurité des schémas de signature de Gennaro-Halevi-Rabin ainsi que celle du schéma Full Domain Hash.

8. Etude de la sécurité du schéma de signature de Gennaro-Halevi-Rabin

Dans ce chapitre nous étudions la sécurité du schéma de signature proposé par Gennaro, Halevi et Rabin à la conférence Eurocrypt '99 [48]. La sécurité de ce schéma de signature est basée sur l'hypothèse RSA-fort et l'existence d'une fonction de hachage division-résistante. Pour cette dernière hypothèse de sécurité, les auteurs du schéma ont conjecturé un niveau de sécurité exponentiel en la taille de la sortie de la fonction de hachage, mais nous mettons en évidence une attaque de complexité sub-exponentielle. Nous modifions ensuite la preuve de sécurité du schéma dans le modèle de l'oracle aléatoire pour prendre en compte de façon plus précise la taille de la fonction de hachage. On montre en particulier que pour obtenir un niveau de sécurité équivalent à celui de RSA avec un module de 1024 bits, il faut utiliser une fonction de hachage de taille 1024 bits environ, au lieu de 512 bits comme cela était suggéré. Le contenu de ce chapitre a été présenté à la conférence Eurocrypt 2000 [33].

8.1 Introduction

Le schéma de signature Gennaro-Halevi-Rabin (noté GHR par la suite) utilise un module RSA N et une base publique aléatoire s modulo N. Pour signer un message m, on calcule la racine e-ème modulo N de s avec $e = H(m)$ où H est une fonction de hachage. La signature σ est vérifiée avec $\sigma^{H(m)} = s \bmod N$.

Le schéma de signature GHR est prouvé sûr contre les attaques à message choisi adaptatives, en admettant deux hypothèses de complexité: l'hypothèse RSA-fort et l'existence de fonction de hachage division-résistante. L'originalité de ce schéma de signature tient au fait que sa sécurité peut être prouvée sans utiliser le modèle de l'oracle aléatoire.

Dans ce chapitre, nous étudions la seconde hypothèse de complexité, c'est à dire l'existence de fonctions de hachage division-résistantes. Une fonction de hachage est dite division-résistante s'il est calculatoirement infaisable de trouver une valeur de la fonction qui divise le produit d'autres valeurs de la fonction. Les auteurs du schéma ont conjecturé dans [48] que la meilleure attaque générique avait une complexité exponentielle en la taille de la fonction de hachage, mais nous mettons au contraire en évidence une attaque générique de complexité sub-exponentielle.

Nous modifions ensuite la preuve de sécurité du schéma de Gennaro-Halevi-Rabin dans le modèle de l'oracle aléatoire, pour pouvoir fixer la taille de la fonction de

hachage de manière rigoureuse. En particulier, pour obtenir un niveau de sécurité comparable à la sécurité d'un RSA de 1024-bits, il faut utiliser une fonction de hachage de taille approximativement 1024 bits au lieu de 512 bits comme cela était suggéré dans [48].

8.2 Le schéma de signature Gennaro-Halevi-Rabin

8.2.1 Définition

Le schéma de signature GHR est un schéma de type "hache-et-signe" qui possède certaines caractéristiques en commun avec le schéma de signature standard RSA :

Génération de la clef : Générer un module RSA $N = p \cdot q$, produit de deux premiers p et q de taille approximativement égale, les entiers $(p-1)/2$ et $(q-1)/2$ étant aussi premiers, ainsi qu'un élément aléatoire $s \in \mathbb{Z}_N^*$. La clef publique est (N, s) et la clef privée est (p, q).

Signature : Pour signer un message m, calculer un exposant impair $e = H(m)$. La signature σ est :

$$\sigma = s^{e^{-1} \bmod \phi(N)} \bmod N$$

où $\phi(N) = (p-1) \cdot (q-1)$ est la fonction d'Euler.

Vérification de la signature : Vérifier que :

$$\sigma^{H(m)} = s \bmod N$$

8.2.2 Preuve de sécurité

L'originalité du schéma de signature de Gennaro-Halevi-Rabin tient au fait que sa sécurité peut être prouvée dans le modèle standard, c'est à dire sans utiliser le modèle de l'oracle aléatoire. Au lieu de cela, la fonction de hachage doit vérifier certaines hypothèses calculatoires. En particulier, la fonction de hachage doit être division-résistante.

Définition 8.2.1 (Division-résistance [48]). *On dit qu'une famille $\mathcal{H}$ de fonction de hachage est division-résistante si il est calculatoirement infaisable de trouver $h \in \mathcal{H}$ et des variables distinctes $X_1, \ldots, X_n$, Y telles que $h(Y)$ divise le produit des $h(X_i)$.*

Dans le modèle de l'oracle aléatoire, le schéma de signature de Gennaro-Halevi-Rabin est résistant contre les attaques à message choisi adaptatives, sous l'hypothèse que le problème RSA-fort est difficile à résoudre.

Définition 8.2.2 (problème RSA-fort [5]). *Etant donné un module RSA N choisi aléatoirement et un entier aléatoire $s \in \mathbb{Z}_N^*$, trouver une paire (e, r) avec $e > 1$ telle que $r^e = s \bmod N$.*

Théorème 8.2.1. *Dans le modèle de l'oracle aléatoire, le schéma de signature de Gennaro-Halevi-Rabin est résistant contre les attaques à message choisi adaptatives, en admettant l'hypothèse que le problème RSA-fort est difficile à résoudre.*

Même si le schéma de Gennaro-Halevi-Rabin est prouvé sûr dans le modèle de l'oracle aléatoire, le théorème précédent ne dit pas comment choisir la taille de la sortie de la fonction de hachage. Les auteurs du schéma conjecturent dans [48] que la complexité de la meilleure attaque générique est exponentielle en la taille de la sortie de la fonction de hachage. Nous montrons au contraire dans le paragraphe suivant qu'il existe une attaque générique de complexité sub-exponentielle en la taille de la fonction de hachage.

8.3 Attaque du schéma de Gennaro-Halevi-Rabin

8.3.1 Méthode d'attaque

Une méthode d'attaque du schéma de signature GHR consiste à rechercher un ensemble de messages $m, m_1, \ldots, m_r$ tels que $H(m)$ divise le plus petit commun multiple de $H(m_1), \ldots, H(m_r)$. Dans ce cas, on dit qu'une *division-collision* pour H a été exhibée.

Ensuite, en utilisant l'algorithme d'Euclide, l'attaquant obtient $a_1, \ldots, a_r, k$ tels que :

$$\frac{a_1}{H(m_1)} + \ldots + \frac{a_r}{H(m_r)} = \frac{1}{\mathrm{PPCM}(H(m_1), \ldots, H(m_r))} = \frac{1}{k \cdot H(m)}$$

et peut forger la signature σ de m à partir des signatures σ_i des messages m_i avec :

$$\sigma = \left(\prod_{i=1}^{r} \sigma_i^{a_i} \right)^k \mod N$$

Si la famille $\mathcal{H}$ de fonctions de hachage est division-résistante, il est calculatoirement infaisable de trouver une division-collision pour une fonction de hachage de $\mathcal{H}$. En particulier, [48] montre qu'un oracle aléatoire est division-résistant.

Une question naturelle est la complexité de trouver une division-collision, si on suppose que la fonction de hachage se comporte comme un oracle aléatoire. La réponse à cette question conditionne le choix des paramètres du schéma de signature. Les auteurs du schéma conjecturent dans [48] que le nombre d'appels à l'oracle de hachage est exponentiel en la taille du haché: ils conjecturent que le nombre de hachés nécessaires est proche de $2^{k/8}$ où k est la taille du haché en bits. Nous mettons en évidence une attaque où le nombre de hachés nécessaires est sub-exponentiel en la taille du haché.

8.3.2 Quelques propriétés des nombres lisses

Nous rappelons quelques propriétés des nombres lisses. Soit y un entier positif. On dit que z est y-lisse si chaque premier divisant z est $\leq y$. Un entier z est dit y-superlisse si chaque puissance de premier divisant z est $\leq y$. En notant $\psi(x, y)$ le nombre d'entiers $z \leq x$ tels que z soit y-lisse, le théorème suivant donne une estimation de la densité des nombres lisses [43] :

Théorème 8.3.1. *Si ϵ est un réel strictement positif, alors uniformément pour $x \geq 10$ et $y \geq (\log x)^{1+\epsilon}$,*

$$\psi(x, y) = x \cdot u^{-u+\circ(u)} \quad \textit{lorsque } x \to \infty$$

où $u = (\log x)/(\log y)$.

En particulier, en notant $y = L_x[\beta] = \exp(\beta\sqrt{\log x \log\log x})$, la probabilité qu'un entier aléatoire compris entre un et x soit $L_x[\beta]$-lisse est :

$$\frac{\psi(x, y)}{x} = L_x\left[-\frac{1}{2\beta} + \circ(1)\right]$$

La proportion d'entiers sans carrés est asymptotiquement $6/\pi^2$ [58] (on dit qu'un entier est *sans carrés* s'il n'apparaît pas de carrés dans la décomposition de l'entier en facteurs premiers). On note $\psi_1(x, y)$ le nombre d'entiers z sans carrés compris entre un et x tels que z soit y-lisse. Le théorème 3 de [58] implique que la même proportion se vérifie pour les entiers y-lisses :

$$\psi_1(x, y) \sim \frac{6}{\pi^2}\psi(x, y) \tag{8.1}$$

sous la condition :

$$\frac{\log y}{\log\log x} \to \infty, \qquad (x \to \infty)$$

Si un entier sans carré est y-lisse, alors il est aussi y-superlisse, donc en notant par $\psi'(x, y)$ le nombre d'entiers $z \leq x$ tels que z est y-superlisse, nous avons pour tout $x, y > 0$:

$$\psi_1(x, y) \leq \psi'(x, y) \leq \psi(x, y)$$

ce qui montre en utilisant (8.1) que pour $y = L_x[\beta]$, la probabilité qu'un entier aléatoire compris entre un et x soit y-superlisse est :

$$\frac{\psi'(x, y)}{x} = L_x\left[-\frac{1}{2\beta} + \circ(1)\right]$$

On obtient les mêmes résultats lorsqu'on considère les entiers impairs compris entre un et x (on utilise pour cela le fait qu'un entier pair de k bits x est y-lisse si et seulement si l'entier de $k-1$ bits $x/2$ est aussi y-lisse).

8.3.3 Attaque générique du schéma de signature

Le principe de l'attaque est le suivant : on cherche d'abord parmi un grand nombre de valeurs de la fonction de hachage, un haché qui soit lisse, c'est à dire divisible par des petits nombres premiers p_i seulement. Ensuite pour chacun des facteurs premiers p_i, on cherche une valeur de haché qui soit divisible par p_i, de sorte que le haché lisse divise le plus petit commun multiple des autres hachés. L'attaque est générique au sens où l'on ne suppose rien sur la fonction de hachage: on assimile la fonction de hachage à un oracle aléatoire.

Théorème 8.3.2. *En assimilant la fonction de hachage H à un oracle aléatoire dont l'espace d'arrivé est l'ensemble des entiers compris entre un et x, il existe un algorithme permettant de trouver une division-collision pour H en temps moyen inférieur à $L_x[\sqrt{2}/2 + \circ(1)]$ avec un nombre moyen d'appels à H inférieur à $L_x[\sqrt{2}/2 + \circ(1)]$.*

Preuve. **Algorithme trouvant une division-collision :**
Entrée : une fonction de hachage H.
Sortie : une division-collision pour H.

1. Rechercher un entier $e = H(M)$ tel que e soit superlisse par rapport à $y = L_x[\beta]$, où β est un paramètre qui sera déterminé par la suite. On utilise pour cela la méthode de factorisation de Pollard-Brent ou la méthode des courbes elliptiques de Lenstra (ECM), et on obtient:

$$e_k = \prod_{i=1}^{r} p_i^{\alpha_i} \quad \text{avec} \quad {p_i}^{\alpha_i} \leq y \text{ pour } 1 \leq i \leq r \tag{8.2}$$

2. Pour chaque facteur premier p_i, rechercher un entier $e_i = H(M_i)$ tel que

$$e_i = 0 \bmod p_i^{\alpha_i}$$

On obtient finalement une division collision pour H, car:

$$e | \text{ PPCM}(e_1, \ldots, e_r)$$

La méthode de Pollard-Brent extrait un facteur p d'un entier n en $\mathcal{O}(\sqrt{p})$ opérations en moyenne. La méthode des courbes elliptiques de Lenstra extrait un facteur p d'un entier n en $L_p[\sqrt{2} + \circ(1)]$ opérations en moyenne.

En utilisant la méthode Pollard-Brent à l'étape 1, un haché $L_x[\beta]$-superlisse $e = H(M)$ est donc trouvé en temps:

$$L_x\left[\frac{1}{2\beta} + \circ(1)\right] \cdot L_x\left[\beta/2 + \circ(1)\right] = L_x\left[\frac{1}{2\beta} + \beta/2 + \circ(1)\right]$$

tandis qu'en utilisant l'algorithm ECM, un entier $L_x[\beta]$-superlisse $e = H(M)$ est trouvé en temps:

$$L_x\left[\frac{1}{2\beta}+\circ(1)\right]\cdot L_x[\circ(1)]=L_x\left[\frac{1}{2\beta}+\circ(1)\right]$$

Dans tous les cas, comme $p_i^{\alpha_i}\leq y$, la seconde étape nécessite moins que $L_x[\beta+\circ(1)]$ opérations.

La complexité totale de l'algorithme précédent est donc inférieure à:

$$L_x\left[\frac{1}{2\beta}+\frac{\beta}{2}+\circ(1)\right]+L_x[\beta+\circ(1)]$$

dans le cas de l'utilisation de la méthode de Pollard-Brent et

$$L_x\left[\frac{1}{2\beta}+\circ(1)\right]+L_x[\beta+\circ(1)]$$

dans le cas de l'utilisation d'ECM.

En prenant $\beta=1$, on obtient une complexité inférieure à $L_x[1+\circ(1)]$ pour la méthode de Pollard-Brent. En prenant $\beta=\sqrt{2}/2$ on obtient une complexité inférieure à $L_x[\sqrt{2}/2+\circ(1)]$ pour l'ECM. □

8.3.4 Temps de calcul de l'attaque en pratique

Comme nous l'avons vu dans le paragraphe précédent, la complexité asymptotique de l'attaque est sub-exponentielle en la taille $k=\log_2 x$ du haché:

$$L_x[\sqrt{2}/2]=\exp\left(\left(\frac{\sqrt{2}}{2}+\circ(1)\right)\sqrt{\log x\log\log x}\right) \tag{8.3}$$

Il paraît difficile de donner une expression exacte du temps de calcul de l'attaque, puisqu'il faudrait pour cela connaître la valeur du terme $\circ(1)$ dans l'équation (8.3).

Cependant, en extrapolant à partir de l'équation (8.3) et des temps de calcul observés pour des petites tailles de haché, nous pouvons estimer le nombre de hachés nécessaires pour mettre l'attaque en oeuvre: le nombre de hachés nécessaires en moyenne est donné approximativement par:

$$\exp\left(\left(\frac{\sqrt{2}}{2}+0.62\cdot(\log x)^{-0.31}\right)\sqrt{\log x\log\log x}\right)$$

Les résultats de la table 8.3.4 suggèrent que pour atteindre un niveau de sécurité équivalent à un RSA avec module de 1024 bits, la taille des hachés devrait être aussi de l'ordre de 1024 bits.

taille des hachés	$\log_2$ nombre de hachés
128	25
256	36
512	53
640	60
768	66
1024	77

Tableau 8.1. Estimation du nombre moyen de hachés nécessaires pour l'attaque.

8.4 Nouvelle preuve de sécurité et attaque générique optimale

8.4.1 Nouvelle preuve de sécurité

Dans ce paragraphe nous donnons une nouvelle preuve de sécurité dans le modèle de l'oracle aléatoire pour le schéma de signature de Gennaro-Halevi-Rabin, analogue à celle de [48], mais permettant une meilleure estimation de la taille nécessaire pour la sortie de la fonction de hachage.

Théorème 8.4.1. *Si le problème RSA-fort est (t', ϵ')-difficile, le schéma de signature de Gennaro-Halevi-Rabin est $(t, q_{hash}, q_{sig}, \epsilon)$-sûr dans le modèle de l'oracle aléatoire, avec $x = 2^k$ et:*

$$\begin{aligned} \epsilon &= q_{hash} \cdot \left(\epsilon' + q_{hash} \cdot L_x[-\sqrt{2} + \circ(1)]\right) \\ t &= t' - \mathtt{poly}(q_{hash}, q_{sig}, k, k_0) \end{aligned}$$

Preuve. Soit $\mathcal{F}$ un attaquant qui $(t, q_{hash}, q_{sig}, \epsilon)$-casse le schéma de Gennaro-Halevi-Rabin. On construit un algorithme $\mathcal{A}$ qui (t', ϵ')-résout le problème RSA-fort. L'algorithme $\mathcal{A}$ répond lui-même aux requêtes de hachage et aux requêtes de signature de l'attaquant. On suppose que lorsque l'attaquant effectue une requête de signature ou renvoie une forge, il a effectué auparavant la requête de hachage correspondante.

Algorithme $\mathcal{A}$:
Entrée: (N, s) et (q_{hash}, q_{sig}), où $N \leftarrow RSA(1^k)$ et $s \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$.
Sortie: (r, e) avec $e > 1$ tel que $r^e = s \bmod N$.

1. Soit $i \leftarrow 1$.
2. Soit j un entier choisi aléatoirement dans $[1, q_{hash}]$.
3. Soient $e_1, \ldots, e_{q_{hash}}$ une suite d'entiers aléatoires impairs de taille k_0 bits.
4. Calculer $E = (\prod_i e_i)/e_j$.
5. Soit $y \leftarrow s^E \bmod N$.
6. Envoyer la clef publique (N, y) à $\mathcal{F}$.

7. Si $\mathcal{F}$ effectue une requête de hachage pour un message M:
   Faire $M_i \leftarrow M$.
   Renvoyer e_i et faire $i \leftarrow i+1$.
8. Si $\mathcal{F}$ effectue une requête de signature pour M_i:
   Si $i \neq j$ renvoyer $\sigma_i = s^{E/e_i} \bmod N$, sinon stopper.
9. Si $\mathcal{F}$ renvoie une forge (M_i, x):
   Si $i \neq j$ stopper.
   Si e_j divise E stopper.
   Sinon, soit $g = \mathrm{PGCD}(e_j, E)$ et $a, b \in \mathbb{Z}$ tels que $a \cdot e_j + b \cdot E = g$.
   Soit $e = e_j/g$ et $r = s^a \cdot x^b \bmod N$. Renvoyer (r, e).

L'algorithme $\mathcal{I}$ renvoie (r, e) avec $e > 1$ et $r^e = s \bmod N$ si $\mathcal{F}$ renvoie une forge à l'étape 9 avec $i = j$, et e_j ne divise pas E. Dans ce cas, on a en effet $g < e_j$ et $e = e_j/g > 1$, et on vérifie de plus que $r^e = s \bmod N$. On note `DIV` l'événement correspondant à e_j divise E.

Les événements "$\mathcal{F}$ renvoie une forge" et "$i = j$ à l'étape 9" étant indépendants et se produisant avec probabilité respectivement ϵ et $1/q_{hash}$, la probabilité que $\mathcal{F}$ renvoie une forge pour le message M_j est ϵ/q_{hash}. La probabilité de succès de $\mathcal{I}$ est donc supérieure à:

$$\epsilon' = \epsilon/q_{hash} - \Pr[\mathtt{Div}]$$

La preuve du théorème découle du lemme suivant:

Lemme 8.4.1. *Soient v entiers $e_1, \ldots, e_v$ distribués aléatoirement dans $[1, 2^{k_0} - 1]$, et e un entier distribué aléatoirement dans $[1, 2^{k_0} - 1]$. Soit $E = \prod_{i=1}^{v} e_i$ et $x = 2^{k_0}$. La probabilité que e divise E est inférieure à:*

$$v \cdot L_x[-\sqrt{2} + \circ(1)]$$

Preuve. Soit $\mathcal{S} = \{e_1, \ldots, e_v\}$ la liste des entiers. On note `EDIV` l'événement selon lequel e divise le produit E des entiers de $\mathcal{S}$, et $P(x, v)$ la probabilité de `EDIV`. Si l'événement `EDIV` est réalisé, alors chaque facteur premier de e divise au moins un des entiers e_i. On définit les événements suivants:

- `Sm` : e est $L_x[\sqrt{2}/4]$-lisse.
- `Sm`(γ, ϵ) : e est $L_x[\gamma + \epsilon]$-lisse et n'est pas $L_x[\gamma]$-lisse, pour $\sqrt{2}/4 < \gamma < \sqrt{2}$ et $\epsilon > 0$.
- $\neg$`Sm`: e n'est pas $L_x[\sqrt{2}]$-lisse.

La probabilité de l'événement `Sm` est d'après le théorème 8.3.1:

$$\Pr[\mathtt{Sm}] = L_x[-\sqrt{2} + \circ(1)]$$

ce qui donne:

$$\Pr[\mathtt{EDIV} \wedge \mathtt{Sm}] \leq L_x\left[-\sqrt{2} + \circ(1)\right] \tag{8.4}$$

La probabilité de l'événement `Sm`(γ, ϵ) est :

$$\Pr[\mathtt{Sm}(\gamma,\epsilon)] = L_x\left[\frac{-1}{2\cdot(\gamma+\epsilon)}+\circ(1)\right] - L_x\left[\frac{-1}{2\cdot\gamma}+\circ(1)\right] \quad (8.5)$$

$$= L_x\left[\frac{-1}{2\cdot(\gamma+\epsilon)}+\circ(1)\right] \quad (8.6)$$

Si e est $L_x[\gamma+\epsilon]$-lisse sans être $L_x[\gamma]$-lisse, alors e contient un facteur premier p supérieur à $L_x[\gamma]$. La probabilité pour que p divise un entier e_i pour un i fixé est inférieure à $1/p$. Par conséquent, la probabilité pour que p divise l'un des entiers e_i de $\mathcal{S}$ est inférieure à v/p, et donc à $v\cdot L_x[-\gamma]$, ce qui donne:

$$\Pr[\mathtt{EDIV}|\mathtt{Sm}(\gamma,\epsilon)] \leq v\cdot L_x[-\gamma] \quad (8.7)$$

En utilisant (8.6) et l'inégalité

$$\gamma + \frac{1}{2(\gamma+\epsilon)} \geq \sqrt{2}-\epsilon$$

on obtient:

$$\begin{aligned}\Pr[\mathtt{EDIV}\wedge\mathtt{Sm}(\gamma,\epsilon)] &= \Pr[\mathtt{EDIV}|\mathtt{Sm}(\gamma,\epsilon)]\cdot\Pr[\mathtt{Sm}(\gamma,\epsilon)]\\ &\leq v\cdot L_x[-\gamma]\cdot L_x\left[\frac{-1}{2\cdot(\gamma+\epsilon)}+\circ(1)\right]\\ &\leq v\cdot L_x\left[-\gamma-\frac{1}{2\cdot(\gamma+\epsilon)}+\circ(1)\right]\\ &\leq v\cdot L_x\left[-\sqrt{2}+\epsilon+\circ(1)\right]\end{aligned}$$

Si e n'est pas $L_x[\sqrt{2}]$-lisse, il contient un facteur premier supérieur à $L_x[\sqrt{2}]$ qui divise l'un des autres entiers e_i, et donc:

$$\Pr[\mathtt{EDIV}\wedge\neg\mathtt{Sm}] \leq v\cdot L_x[-\sqrt{2}] \quad (8.8)$$

On partitionne l'intervalle $[\sqrt{2}/4,\sqrt{2}]$ en n intervalles $[x_i,x_{i+1}]$ pour i compris entre 0 et $n-1$ avec $x_i=\sqrt{2}/4\cdot(1+3\cdot i/n)$ pour obtenir:

$$\Pr[\mathtt{EDIV}] = \Pr[\mathtt{EDIV}\wedge\mathtt{Sm}] + \Pr[\mathtt{EDIV}\wedge\neg\mathtt{Sm}] + \sum_{i=0}^{n-1}\Pr\left[\mathtt{EDIV}\wedge\mathtt{Sm}(x_i,\frac{3\cdot\sqrt{2}}{4\cdot n})\right]$$

En utilisant les équations (8.4) et (8.8), on obtient:

$$\Pr[\mathtt{EDIV}] \leq n\cdot v\cdot L_x\left[-\sqrt{2}+\frac{3\cdot\sqrt{2}}{4\cdot n}+\circ(1)\right]$$

et finalement en prenant $n=\log x$,

$$\Pr[\mathtt{EDIV}] \leq v\cdot L_x[-\sqrt{2}+\circ(1)]$$

□

8.4.2 Attaque générique optimale

Le théorème précédent montre que l'attaque générique du paragraphe 8.3.3 sur la division-résistance de la fonction de hachage est optimale. Le théorème 8.4.1 montre en effet que pour forger une signature avec probabilité $\epsilon \simeq 1$, en supposant que le problème RSA-fort soit (t', ϵ') difficile avec ϵ' négligeable, le nombre minimum q_{hash} d'appels à l'oracle doit être tel que:

$$1 = (q_{hash})^2 \cdot L_x \left[-\sqrt{2} + \circ(1) \right]$$

ce qui donne:

$$q_{hash} = L_x \left[\frac{\sqrt{2}}{2} + \circ(1) \right]$$

Avec une complexité moyenne en temps de

$$L_x \left[\frac{\sqrt{2}}{2} + \circ(1) \right]$$

l'attaque du paragraphe 8.3.3 est donc optimale.

8.5 Conclusion

Dans ce chapitre, nous avons étudié la sécurité du schéma de Gennaro-Halevi-Rabin. Nous avons mis en évidence une attaque de complexité sub-exponentielle en la taille du haché. Nous avons aussi donné une preuve de sécurité du schéma dans le modèle de l'oracle aléatoire, qui montre que l'attaque précédente est optimale et permet de fixer la taille du haché avec une meilleure précision. En particulier, on voit que pour obtenir un niveau de sécurité équivalent à RSA avec un module de 1024 bits, il faut prendre un haché de taille voisine de 1024 bits.

9. Sécurité du schéma de signature Full Domain Hash

Le schéma de signature Full Domain Hash (FDH) est un schéma de signature basé sur RSA dans lequel la sortie de la fonction de hachage a la même taille que le module N. FDH admet une preuve de sécurité dans le modèle de l'oracle aléatoire, basée sur la difficulté du problème consistant à inverser RSA. Dans ce chapitre nous donnons une preuve de sécurité améliorée pour ce schéma de signature, qui permet d'obtenir une meilleure garantie sur sa sécurité. On améliore ainsi l'efficacité du schéma car un module RSA de taille plus petite peut être utilisé pour un même niveau de sécurité. La même technique permet d'obtenir une preuve de sécurité améliorée pour le schéma de signature de Rabin [76], pour le schéma de signature de Paillier [71] et pour le schéma de signature Gennaro-Halevi-Rabin [48]. Le contenu de ce chapitre a été présenté lors de la conférence Crypto 2000 [25].

9.1 Introduction

Lors de l'utilisation pratique d'un schéma de signature prouvé sûr, il faut tenir compte de la qualité de la réduction. Une réduction est de bonne qualité lorsque casser le schéma de signature conduit à résoudre le problème difficile avec une probabilité proche de un. Dans ce cas, le schéma de signature est presque aussi sûr que le problème difficile sur lequel il est basé. Au contraire, si cette probabilité est trop faible, il n'y aura qu'une faible garantie sur la sécurité du schéma de signature; il faudra utiliser des clés plus longues, ce qui diminuera l'efficacité du schéma.

La preuve de sécurité de [9] pour FDH borne supérieurement la probabilité ϵ de casser FDH en temps t par $\epsilon' \cdot (q_{hash} + q_{sig})$, où ϵ' est la probabilité d'inverser RSA en un temps t' comparable à t, et q_{hash} et q_{sig} sont le nombre maximum de requêtes de hachage et de signature que l'attaquant peut réaliser.

La nouvelle preuve de sécurité permet de borner la probabilité de casser FDH par $\epsilon \simeq q_{sig} \cdot \epsilon'$ avec toujours des temps de calcul t et t' comparables. Comme dans la pratique, q_{sig} est très inférieur à q_{hash}, on obtient une meilleure garantie sur la sécurité de FDH. Cela permet d'utiliser des modules RSA plus petits, et donc de diminuer le temps nécessaire pour générer une signature.

9.2 Le schéma de signature FDH

Le schéma de signature FDH (`GenFDH` , `SignFDH`, `VerifyFDH`) [9] est défini de la façon suivante. L'algorithme de génération de clef `GenFDH` reçoit en entrée 1^k, exécute l'algorithme de génération de clef $\texttt{RSA}(1^k)$ pour obtenir (N, e, d). L'algorithme `GenFDH` renvoie (pk, sk), où la clef publique pk est (N, e) et la clef secrète sk est (N, d). Les algorithmes de génération et de vérification de signature utilisent une fonction de hachage $H_{FDH} : \{0,1\}^* \to \mathbb{Z}_N^*$, prenant en entrée des messages de taille arbitrairement longue et renvoyant en sortie un entier inversible modulo N. Les algorithmes de génération et de vérification de signature sont les suivants :

$\texttt{SignFDH}_{N,d}(M)$
$y \leftarrow H_{FDH}(M)$
Retourner $y^d \bmod N$

$\texttt{VerifyFDH}_{N,e}(M, x)$
$y \leftarrow x^e \bmod N; y' \leftarrow H_{FDH}(M)$
si $y = y'$ alors retourner un sinon retourner zéro.

Le théorème suivant prouve la sécurité de FDH dans le modèle de l'oracle aléatoire. Nous rappelons la preuve de ce théorème, qui se trouve dans [9].

Théorème 9.2.1. *Si le problème RSA est (t', ϵ')-sûr, le schéma de signature FDH est $(t, q_{hash}, q_{sig}, \epsilon)$-sûr, avec :*

$$t = t' - (q_{hash} + q_{sig} + 1) \cdot \mathcal{O}(k^3) \tag{9.1}$$
$$\epsilon = (q_{hash} + q_{sig} + 1) \cdot \epsilon' \tag{9.2}$$

Preuve. Nous construisons un algorithme $\mathcal{R}$ qui inverse RSA en utilisant un attaquant $\mathcal{F}$. L'algorithme $\mathcal{R}$ répond lui-même aux requêtes de hachage et aux requêtes de signature de $\mathcal{F}$. Nous supposons que lorsque l'attaquant fait une requête de signature sur un message M, il a déjà fait une requête de hachage sur le même message M. Si ce n'est pas le cas, l'algorithme $\mathcal{R}$ fait la requête de hachage correspondante. De manière similaire, nous supposons qu'une requête de hachage a été faite sur le message M dont la signature est forgée par l'attaquant. Si ce n'est pas le cas, l'algorithme de réduction réalise la requête de hachage correspondante.

Algorithme $\mathcal{R}$.
Entrée: (N, e, y) et (q_{hash}, q_{sig}), où $(N, e) \leftarrow RSA(1^k)$ et $y \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$.
Sortie: $y^d \bmod N$.

1. Fixer $i \leftarrow 0$
2. Envoyer (N, e) à $\mathcal{F}$.
3. Sélectionner aléatoirement un entier $j \in [1, q_{hash} + q_{sig} + 1]$.
4. Si $\mathcal{F}$ réalise une requête de hachage pour le message M:
   $i \leftarrow i + 1$; $M_i \leftarrow M$; $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$

Si $i = j$, renvoyer $H(M_i) = y$
Si $i \neq j$, renvoyer $H(M_i) = r_i^e \bmod N$

5. Si $\mathcal{F}$ réalise une requête de signature pour le message M :
   Retourner r_i si $H(M_i) = r_i^e \bmod N$; sinon stopper.
6. Si $\mathcal{F}$ renvoie une signature forgée (M_i, x):
   Si $i = j$ renvoyer $x = y^d \bmod N$, sinon stopper.

Soit M le message dont la signature est forgée par l'attaquant. Par hypothèse une requête de hachage a été réalisée pour M, donc il existe un indice $i \in [1, q_{hash}+q_{sig}+1]$ tel que $M = M_i$. A l'étape 6, l'algorithme $\mathcal{R}$ renvoie $y^d \bmod N$ si $i = j$, ce qui se produit avec probabilité $1/(q_{hash} + q_{sig} + 1)$. Comme l'algorithme $\mathcal{F}$ renvoie une signature forgée (M, x) avec probabilité supérieure à ϵ, la probabilité de succès de $\mathcal{R}$ est supérieure à :

$$\epsilon' = \frac{\epsilon}{q_{hash} + q_{sig} + 1}$$

ce qui donne l'équation (9.2).

Le temps d'exécution de $\mathcal{R}$ est le temps d'exécution de $\mathcal{F}$ plus le temps nécessaire pour calculer les termes $r_i^e \bmod N$, ce qui prend un temps $\mathcal{O}(k^3)$. On obtient ainsi l'équation (9.1). □

Même si le schéma de signature FDH est prouvé sûr dans le modèle de l'oracle aléatoire, l'inconvénient du résultat précédent est que la probabilité maximum ϵ de casser FDH peut être beaucoup plus grande que la probabilité ϵ' d'inverser RSA. Par exemple, si on suppose que l'attaquant peut faire $q_{sig} = 2^{30}$ requêtes de signature et $q_{hash} = 2^{60}$ requêtes de hachage, même si la probabilité d'inverser RSA pour une taille de module donnée et un temps de calcul t est inférieure à 2^{-61}, on obtient seulement que la probabilité de casser FDH est inférieure à $1/2$, ce qui n'est pas satisfaisant. Pour obtenir un niveau de sécurité acceptable pour FDH, il faut utiliser un module RSA plus grand, ce qui rend le schéma plus consommateur en temps de calcul.

Pour obtenir une meilleur garantie de sécurité, Bellare et Rogaway ont réalisé un nouveau schéma de signature, *Probabilistic Signature Scheme* (PSS), qui possède une garantie de sécurité bien meilleure. En effet, la probabilité de forger une signature est presque aussi faible que la probabilité d'inverser RSA ($\epsilon \simeq \epsilon'$). Nous montrons dans la section suivante qu'une meilleure preuve de sécurité peut malgré tout être obtenue pour le schéma FDH.

9.3 Nouvelle preuve de sécurité

Dans ce paragraphe nous montrons qu'il existe une meilleure preuve de sécurité pour le schéma de signature FDH. Nous prouvons en effet le théorème suivant:

Théorème 9.3.1. *Si le problème consistant à inverser RSA est (t', ϵ')-difficile, alors FDH est $(t, q_{hash}, q_{sig}, \epsilon)$-sûr, avec:*

$$t = t' - (q_{hash} + q_{sig} + 1) \cdot \mathcal{O}(k^3) \tag{9.3}$$

$$\epsilon = \frac{1}{(1 - \frac{1}{q_{sig}+1})^{q_{sig}+1}} \cdot q_{sig} \cdot \epsilon' \tag{9.4}$$

Preuve. Soit $\mathcal{F}$ un attaquant parvenant à $(t, q_{hash}, q_{sig}, \epsilon)$-casser le schéma FDH. Nous construisons un algorithme $\mathcal{R}$ qui inverse RSA en utilisant l'attaquant $\mathcal{F}$. Comme précédemment, l'algorithme $\mathcal{R}$ répond lui-même aux requêtes de hachage et aux requêtes de signature. On suppose aussi que lorsque l'attaquant $\mathcal{F}$ fait une requête de signature, il a déjà fait la requête de hachage correspondante, et qu'une requête de hachage a été faite pour le message dont la signature est forgée.

Algorithme $\mathcal{R}$.
Entrée: (N, e, y) et (q_{hash}, q_{sig}), où $(N, e) \leftarrow RSA(1^k)$ et $y \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$.
Sortie: $y^d \bmod N$.

1. Fixer $i \leftarrow 0$
2. Envoyer (N, e) to $\mathcal{F}$.
3. Si $\mathcal{F}$ réalise une requête de hachage pour M:
   $i \leftarrow i + 1$; $M_i \leftarrow M$; $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$
   Lancer une pièce c_i avec un biais γ qui sera précisé par la suite.
   $c_i = 0$ avec probabilité γ et $c_i = 1$ avec probabilité $1 - \gamma$.
   Retourner $H(M) = y^{c_i} \cdot r_i^e \bmod N$
4. Si $\mathcal{F}$ réalise une requête de signature pour le message M_i:
   Retourner r_i si $H(M_i) = r_i^e \bmod N$. Sinon arrêter.
5. Si $\mathcal{F}$ renvoie une forge (M, x):
   Si $H(M) = y \cdot r_i^e \bmod N$ alors renvoyer $y^d = x/r_i \bmod N$. Sinon arrêter.

La probabilité que $\mathcal{R}$ puisse répondre à une requête de signature à l'étape 4 est γ; la probabilité que $\mathcal{R}$ réponde à toutes les requêtes de signature est donc supérieure à $\gamma^{q_{sig}}$. Finalement, $\mathcal{F}$ renvoie une forge (M, x) avec probabilité ϵ; l'algorithme $\mathcal{R}$ peut utiliser cette forge à l'étape 5 pour renvoyer $y^d \bmod N$ avec probabilité $1 - \gamma$. La probabilité de succès de $\mathcal{R}$ est donc supérieure à:

$$\gamma^{q_{sig}} \cdot (1 - \gamma) \cdot \epsilon$$

Cette probabilité est maximale lorsque

$$\gamma = 1 - \frac{1}{q_{sig} + 1}$$

et on obtient l'équation (9.4). □

Notons qu'avec $(1 - 1/i)^i \geq 1/4$ pour tout entier $i \geq 2$ on peut prendre $\epsilon = 4 \cdot q_{sig} \cdot \epsilon'$. Par exemple, si on suppose que pour un paramètre de sécurité donné k, la probabilité d'inverser RSA est inférieure à 2^{-60} pour une borne de temps donnée t, et si le forgeur a le droit de faire au plus 2^{60} requêtes de hachage et 2^{30} requêtes de signature, la probabilité de casser FDH est inférieure à 2^{-28} pour une borne de temps proche de t.

9.4 Conclusion

Dans beaucoup de preuves de sécurité dans le modèle de l'oracle aléatoire, l'algorithme de réduction doit "deviner" quelle requête de hachage sera finalement utilisée par l'attaquant pour forger la signature, ce qui donne un facteur $1/q_{hash}$ dans la probabilité de succès finale de l'algorithme de réduction. La preuve de sécurité précédente montre qu'une méthode plus efficace est d'inclure le challenge y dans un nombre suffisant de réponses aux requêtes de hachage pour que la forge de l'attaquant puisse être finalement utilisée avec une plus grande probabilité. En dehors de FDH, cette observation s'applique au schéma de signature de Rabin [76], au schéma de signature de Paillier [71], et aussi au schéma Gennaro-Halevi-Rabin [48], pour lesquels le facteur q_{hash} dans la preuve de sécurité peut être remplacé par q_{sig}.

L'intérêt de cette nouvelle preuve de sécurité est que la probabilité de succès de la réduction est maintenant indépendante du nombre de requêtes de hachage réalisées par l'attaquant. L'intérêt pratique est que dans les applications réelles, l'oracle aléatoire est remplacé par une fonction de hachage bien définie, et le nombre de hachés que l'attaquant peut calculer est seulement limité par sa capacité de calcul. Au contraire, le nombre de requêtes de signature peut être limité : il suffit que le signataire refuse de signer plus de 2^{20} ou 2^{30} messages pour une clef donnée.

10. Preuves de sécurité pour schémas de signature à signature unique

Comme nous l'avons vu dans les chapitres précédents, on prouve généralement la sécurité d'un schéma de signature numérique en montrant que tout algorithme susceptible de forger des signatures peut être transformé en un algorithme résolvant un problème mathématique jugé difficile à résoudre, comme par exemple factoriser de grands entiers ou calculer un logarithme discret. On décrit dans ce chapitre une nouvelle technique permettant d'analyser la preuve de sécurité d'un schéma de signature contre les attaques à message choisi. Cette technique très simple permet d'obtenir une borne supérieure sur l'efficacité de la preuve de sécurité, à la fois dans le modèle de l'oracle aléatoire et dans le modèle standard, pour un schéma de signature qui associe à chaque message une unique signature. En utilisant cette technique, on montre que le schéma *Full Domain Hash*, le schéma de Gennaro-Halevi-Rabin et le schéma de Paillier ont une preuve de sécurité optimale dans le modèle de l'oracle aléatoire.

10.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, la preuve de sécurité de [9] borne supérieurement la probabilité ε de casser FDH en temps t par $(q_{hash} + q_{sig}) \cdot \varepsilon'$, où ε' est la probabilité d'inverser RSA en temps t' proche de t. Dans le chapitre précédent, nous avons vu que cette borne pouvait être améliorée en $\varepsilon \simeq q_{sig} \cdot \varepsilon'$.

Cependant, la garantie de sécurité obtenue n'est pas parfaite, car la probabilité de casser FDH est toujours largement supérieure à celle d'inverser RSA, et par conséquent FDH n'est pas encore aussi sûr que RSA. Au contraire, le schéma de signature *Probabilistic Signature Scheme* (PSS), proposé par Bellare et Rogaway dans [9], possède quant à lui un niveau de sécurité équivalent à celui du problème RSA. Une question intéressante est donc de savoir s'il est possible ou non d'améliorer encore la preuve de sécurité pour FDH, de façon à obtenir un niveau de sécurité qui serait équivalent à celui de PSS.

Dans ce chapitre, on décrit une nouvelle technique pour analyser la sécurité de schémas de signature contre les attaques à message choisi. Cette technique permet d'obtenir une borne supérieure pour la sécurité de FDH, qui montre que la preuve de sécurité du chapitre précédent avec $\varepsilon \simeq q_{sig} \cdot \varepsilon'$ est optimale, lorsque le forgeur est utilisé en boîte noire. Contrairement à PSS, FDH ne peut pas être prouvé aussi sûr

qu'inverser RSA, ce qui répond à la question ouverte posée par Bellare et Rogaway dans [9] sur l'existence d'une meilleure preuve de sécurité pour FDH.

Plus généralement, cette technique s'applique à tout schéma de signature où chaque message possède une signature unique. On obtient une borne supérieure pour les preuves de sécurité utilisant le forgeur en boîte noire, à la fois dans le modèle de l'oracle aléatoire et dans le modèle standard, pour les schémas de signature à signature unique. Contrairement aux schémas de signature pour lesquels un message peut avoir plusieurs signatures (comme pour PSS), un schéma de signature à signature unique ne peut pas être prouvé aussi sûr que le problème mathématique sur lequel il est basé.

10.2 Sécurité des schémas de signature à signature unique dans le modèle de l'oracle aléatoire

10.2.1 Preuve de sécurité pour FDH

Il paraît difficile d'obtenir une meilleure preuve de sécurité pour FDH que celle présentée dans le chapitre précédent. Intuitivement, soit la réduction "connaît" la signature de M et n'apprend rien d'une forge de M, soit la réduction "ne connaît pas" la signature de M mais échoue si le forgeur requiert la signature de M. La preuve de sécurité du chapitre précédent pour FDH montre que la meilleure proportion de "signatures inconnues" est d'environ $1/q_{sig}$, ce qui donne une probabilité de succès d'environ ε_F/q_{sig} avec un forgeur qui renvoie une forge avec probabilité ε_F.

Considérons maintenant un forgeur qui renvoie une forge avec probabilité égale à un, et supposons qu'il existe une réduction $\mathcal{R}$ pour FDH, dont la probabilité de succès soit égale à un en utilisant ce forgeur. En d'autres termes, pour toute suite de requêtes de signature et toute forge produite par le forgeur, la réduction $\mathcal{R}$ renvoie $y^d \bmod N$. Dans ce cas on peut utiliser la réduction pour inverser RSA en utilisant la technique suivante: on commence par envoyer y à $\mathcal{R}$, et ensuite on demande à $\mathcal{R}$ de hacher le message M et de le signer, pour obtenir $H(M)$ et $s = H(M)^d \bmod N$. Ensuite on relance la réduction avec le même ruban aléatoire, en demandant à nouveau à la réduction de hacher le message M. On obtient la même valeur de haché $H(M)$ que précédemment, parce que pour un ruban aléatoire fixé, $\mathcal{R}$ se comporte comme un algorithme déterministe qui renvoie la même sortie pour une entrée donnée. De plus, on connaît la signature $s = H(M)^d \bmod N$ de M, et on l'envoie en tant que forge à $\mathcal{R}$. C'est une forge valide pour $\mathcal{R}$ parce que la signature du message M n'a pas été demandée à $\mathcal{R}$ après qu'il a été relancé, et donc finalement $\mathcal{R}$ renvoie $y^d \bmod N$. Ainsi, cela montre que sans être capable de casser FDH, on peut construire à partir de $\mathcal{R}$ un algorithme inversant RSA, dont la probabilité de succès est égale à un et dont le temps de calcul est au plus deux fois le temps de calcul de la réduction. En conséquence, si inverser RSA est difficile, il n'y a pas de réduction efficace pour FDH dont la probabilité de succès soit égale à un.

La technique décrite est donc très simple: elle consiste à tout d'abord demander à la réduction de signer un message M, ensuite à relancer la réduction avec le même ruban aléatoire, pour ensuite envoyer la signature de M comme forge à la réduction. En relançant avec le même ruban aléatoire, on est sûr d'obtenir la même valeur de haché pour M, et donc la signature $s = H(M)^d \bmod N$ obtenue avant de relancer est encore valide après que $\mathcal{R}$ a été relancé. En fait, cette technique permet de simuler un forgeur, sans être capable de briser la sécurité du schéma de signature. Cependant, notre simulation n'est pas parfaite: en effet, lorsqu'on renvoie la signature de M comme forge à $\mathcal{R}$, on sait que $\mathcal{R}$ est capable de répondre à la requête de signature de M, ce qui n'est pas le cas pour un vrai forgeur: lorsqu'un forgeur renvoie la signature forgée d'un message M', $\mathcal{R}$ peut être ou ne pas être capable de répondre à la requête de signature sur M'.

Bien sûr, si on applique cette technique à la réduction de FDH du chapitre précédent, dont la probabilité de succès est d'environ ε_F/q_{sig}, on ne sera pas capable d'inverser RSA. En effet, lorsque la réduction répond à la requête de signature sur le message M_i, cela signifie qu'on a $H(M_i) = r_i^e \bmod N$, et donc après avoir relancé la réduction et avoir donné la signature de M_i comme forge, la réduction ne va pas renvoyer $y^d \bmod N$. On verra dans le prochain chapitre que dès que la probabilité de succès de la réduction dépasse environ ε_F/q_{sig}, la réduction peut être utilisée pour inverser RSA. En conséquence, si inverser RSA est difficile, la probabilité de succès de la réduction ne peut pas être supérieure à environ ε_F/q_{sig}, lorsqu'elle exécute une seule fois un forgeur dont la probabilité de succès est ε_F. La réduction pour FDH du chapitre précédent est donc optimale.

10.2.2 Sécurité des schémas à signature unique dans le modèle de l'oracle aléatoire

On a vu que notre technique permettait de simuler un forgeur lors de l'interaction avec une réduction pour FDH: on commence par demander la signature d'un message M, on relance ensuite la réduction avec le même ruban aléatoire, et on renvoie comme forge la signature du message M. En fait, cette technique est plus générale que FDH et peut s'appliquer à tout schéma de signature. Cependant, la technique ne s'applique que si le schéma de signature est à signature unique, *i.e.* lorsque chaque message possède une unique signature, sinon on ne peut pas simuler le forgeur. En effet, si le message M a plusieurs signatures, notre simulation envoie comme forge du message M une signature s qui a été reçue de $\mathcal{R}$, tandis qu'un vrai forgeur n'a aucune information sur s (puisqu'il n'a pas demandé à $\mathcal{R}$ de signer M), et peut donc renvoyer une signature $s' \neq s$ pour M. Au contraire, lorsque chaque message possède une signature unique, notre simulation obtient de $\mathcal{R}$ la seule signature possible s de M, qui est naturellement la même que celle qu'un vrai forgeur aurait envoyée pour M.

On commence par appliquer notre technique aux schémas de signature prouvés sûrs dans le modèle de l'oracle aléatoire. Dans le modèle de l'oracle aléatoire, la fonction de hachage est remplacée par une fonction aléatoire.

Définition 10.2.1 (oracle aléatoire). *Pour toute constante k, un oracle aléatoire est une fonction H sélectionnée uniformément dans l'ensemble $\mathcal{H}_k$ des fonctions de $\{0,1\}^*$ vers $\{0,1\}^k$.*

On dit qu'un schéma de signature est du type hache-et-signe si l'algorithme de génération de signature commence par hacher le message pour ensuite signer le haché en utilisant la clé privée.

Définition 10.2.2 (schéma de signature hache-et-signe). *On dit qu'un schéma de signature* (`Gen`, `Sign`, `Verify`) *est un schéma de type hache-et-signe si* `Sign` *prend en entrée le message M, la clef publique pk et la clef privée sk, exécute* `Hashing` *avec M et pk, obtient h, et exécute ensuite* `Signing` *avec h et sk, pour obtenir et retourner la signature x, où:*
- `Hashing` *est un algorithme prenant en entrée un message M à signer et la clef publique pk, et retournant le haché h.* `Hashing` *peut avoir accès à un oracle aléatoire.*
- `Signing` *est un algorithme prenant en entrée h et la clef privée sk et retournant la signature x.* `Signing` *n'a pas accès à un oracle aléatoire.*

Comme exemples de schémas de signature hache-et-signe, on peut citer FDH, PSS, le schéma de Gennaro-Halevi-Rabin (GHR) dans le modèle de l'oracle aléatoire [48], le schéma de signature de Paillier [71] ainsi que DSA [1].

L'algorithme `Hashing` peut faire plusieurs requêtes à l'oracle de hachage, par exemple deux requêtes comme dans PSS. Pour simplifier, on dira par la suite qu'un forgeur peut faire q_{hash} requêtes de hachage s'il peut appliquer l'algorithme `Hashing` à q_{hash} messages. Le véritable nombre de requêtes de hachage q'_{hash} sera alors un multiple de q_{hash} (pour PSS, on a $q'_{hash} = 2 \cdot q_{hash}$).

On dit qu'un algorithme de signature est à signature unique si chaque message possède une unique signature, étant donné l'oracle aléatoire $H \in \mathcal{H}_k$; formellement:

Définition 10.2.3 (schéma de signature à signature unique). *On dit qu'un schéma de signature est à signature unique si pour toute clé publique pk, tout message M et tout oracle aléatoire H dans $\mathcal{H}_k$, il existe un unique x tel que* $\texttt{Verify}_{pk}(M,x) = 1$.

Comme exemples de schéma de signature hache-et-signe à signature unique, on peut citer FDH, GHR dans le modèle de l'oracle aléatoire ainsi que le schéma de Paillier. En revanche, PSS et DSA ne sont pas à signature unique.

Dans la définition 10.2.2, les algorithmes `Hashing` et `Signing` peuvent être probabilistes. Cependant, pour un schéma de signature à signature unique, étant donné l'oracle aléatoire $H \in \mathcal{H}_k$, la signature donnée par $x \leftarrow \texttt{Sign}_{pk,sk}(M)$ est unique. Donc sans perte de généralité, on considérera par la suite que pour un schéma de

signature hache-et-signe à signature unique, les algorithmes `Hashing` and `Signing` sont déterministes.

Notons que le fait que la génération de signature soit déterministe n'implique pas réciproquement que le schéma de signature soit à signature unique. Par exemple, considérons PSS où la source d'aléa est un générateur pseudo-aléatoire dont la graine est la clef privée. La génération de signature devient alors déterministe, mais étant donné un message M il y a plusieurs valeurs distinctes de x telles que $\texttt{Verify}_{pk}(M, x) = 1$.

Pour un schéma de signature hache-et-signe, la signature dépend de la clef publique, de la clef privée, du message et de l'oracle aléatoire $H \in \mathcal{H}_k$. Cependant, pour un schéma hache-et-signe à signature unique, le théorème suivant montre qu'étant donné $h \leftarrow \texttt{Hashing}_{pk}(M)$, la signature de M est unique et ne dépend pas de l'oracle aléatoire $H \in \mathcal{H}_k$. On note $\texttt{Sign}^H_{pk,sk}$, $\texttt{Hashing}^H_{pk}$ et $\texttt{Verify}^H_{pk}$ les algorithmes `Sign`, `Hashing` and `Verify` ayant accès à l'oracle $H \in \mathcal{H}_k$.

Théorème 10.2.1. *Soit $\mathcal{S}$ un schéma de signature hache-et-signe à signature unique. Soit pk la clef publique et M le message. Soient $H, H' \in \mathcal{H}_k$ deux oracle aléatoires et x, x' les signatures uniques telles que $\texttt{Verify}^H_{pk}(M, x) = 1$ et $\texttt{Verify}^{H'}_{pk}(M, x') = 1$. Soient $h \leftarrow \texttt{Hashing}^H_{pk}(M)$ et $h' \leftarrow \texttt{Hashing}^{H'}_{pk}(M)$. Alors si $h = h'$, on a $x = x'$.*

Preuve. On a $x \leftarrow \texttt{Sign}^H_{pk,sk}(M)$, ce qui étant donné $h \leftarrow \texttt{Hashing}^H_{pk}(M)$ implique que la signature x est telle que $x \leftarrow \texttt{Signing}_{sk}(M, h)$. Si $h = h'$, cela donne $x \leftarrow \texttt{Signing}_{sk}(M, h')$, et en utilisant $h' \leftarrow \texttt{Hashing}^{H'}_{pk}(M)$ cela donne $x \leftarrow \texttt{Sign}^{H'}_{pk,sk}(M)$ et ainsi $x = x'$. □

La sécurité du schéma de signature que l'on considère n'est pas nécessairement basée sur la difficulté d'inverser RSA; elle peut être basée sur la difficulté de n'importe quel problème de recherche Π, défini de la façon suivante:

Définition 10.2.4. *Un problème de recherche Π est un triplet $(\texttt{Gen}\Pi, D, S)$ où D est un ensemble d'éléments appelés instances et pour chaque instance $I \in D$, $S[I]$ est un ensemble d'éléments appelés solutions de I. $\texttt{Gen}\Pi$ est un algorithme qui prend en entrée 1^k et renvoie en sortie une instance $I \in D$ de taille k.*

Définition 10.2.5. *On dit qu'un algorithme $\mathcal{A}$ (t_A, ε_A)-résout le problème Π si après avoir reçu une instance I telle que $I \leftarrow \texttt{Gen}\Pi(1^k)$ et avoir effectué $t_A(k)$ opérations, l'algorithme renvoie une solution z de I avec probabilité supérieure à $\varepsilon_A(k)$ pour tout $k \in \mathbb{N}$.*

Définition 10.2.6. *On dit qu'un problème Π est (t_A, ε_A)-difficile s'il n'existe pas d'algorithme $\mathcal{A}$ qui (t_A, ε_A)-résout Π.*

Dans ce qui suit, on considère un schéma de signature hache-et-signe à signature unique, prouvé sûr dans le modèle de l'oracle aléatoire, en supposant que résoudre

un problème donné Π est difficile. Cela signifie qu'il existe une réduction entre résoudre le problème difficile Π et casser le schéma de signature $\mathcal{S}$. Une réduction entre résoudre Π et casser $\mathcal{S}$ est un algorithme utilisant un forgeur pour $\mathcal{S}$ pour résoudre Π. On se place dans le cadre d'attaques à messages choisis, et donc le forgeur a le droit de requérir la signature de messages de son choix. De plus, dans le modèle de l'oracle aléatoire, le forgeur ne peut pas calculer la fonction de hachage par lui-même: il doit effectuer une requête de hachage. En conséquence, lorsqu'il interagit avec la forgeur, l'algorithme de réduction doit répondre aux requêtes de hachage et aux requêtes de signatures effectuées par le forgeur. Par exemple, dans la preuve de sécurité de FDH du chapitre précédent, la réduction répond aux requêtes de hachage du forgeur à l'étape 3 et aux requêtes de signature à l'étape 4, pour finalement inverser RSA à l'étape 5.

Définition 10.2.7. *Une réduction $\mathcal{R}$ dans le modèle de l'oracle aléatoire entre résoudre $(Gen\Pi, D, S)$ et casser* (`Gen`, `Sign`, `Verify`) *est un algorithme probabiliste prenant en entrée une instance I du problème Π ainsi que (q_{hash}, q_{sig}), où $I \leftarrow Gen\Pi(1^k)$, et renvoyant une solution z de I. L'algorithme de réduction interagit avec un forgeur $\mathcal{F}$ pour le schéma* (`Gen`, `Sign`, `Verify`)*, lequel renvoie une forge après au plus q_{hash} requêtes de hachage et q_{sig} requêtes de signatures. L'algorithme de réduction répond lui-même aux requêtes de hachage et de signature du forgeur $\mathcal{F}$.*

On quantifie l'efficacité de la réduction en donnant la probabilité que la réduction renvoie une solution du problème Π en utilisant un forgeur qui $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-casse le schéma de signature, après un temps de calcul supplémentaire de t_R. Dans ce paragraphe, on considère seulement une réduction exécutant une seule fois le forgeur, comme la réduction du chapitre précédent pour FDH. On considérera les réductions exécutant un forgeur plusieurs fois dans le paragraphe suivant.

Définition 10.2.8. *On dit qu'une réduction $\mathcal{R}$ $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-réduit la résolution de $(\mathtt{Gen}\Pi, D, S)$ au cassage du schéma de signature* (`Gen`, `Sign`, `Verify`) *si après avoir exécuté une seule fois tout forgeur qui $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-casse le schéma* (`Gen`, `Sign`, `Verify`)*, la réduction renvoie une solution de Π avec probabilité supérieure à ε_R, en un temps de calcul supplémentaire de t_R.*

Dans la définition précédente, $t_R(k)$ est le temps de calcul de la réduction seule et ne comprend pas le temps de calcul du forgeur. Globalement, le temps nécessaire pour résoudre Π sera $t_F(k) + t_R(k)$, où $t_F(k)$ est le temps de calcul du forgeur. Par exemple, la réduction du chapitre précédent pour FDH $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-réduit inverser RSA à casser FDH, avec $t_R = (q_{hash} + q_{sig}) \cdot \mathcal{O}(k^3)$ et $\varepsilon_R = 1/(4 \cdot q_{sig}) \cdot \varepsilon_F$.

De plus, dans la définition précédente, la réduction doit réussir avec probabilité supérieure à ε_F pour *tout* forgeur qui casse le schéma de signature avec probabilité ε_F, et pas seulement pour un forgeur particulier. Il est facile de construire une réduction très efficace pour un forgeur particulier. Par exemple, si un forgeur pour FDH envoie la factorisation de N en faisant une requête de hachage, la réduction peut réussir à renvoyer $y^d \bmod N$ avec probabilité un.

Le théorème suivant montre que pour tout schéma de signature hache-et-signe à signature unique prouvé sûr dans le modèle de l'oracle aléatoire, en supposant la difficulté d'un problème Π, la preuve de sécurité ne peut pas renvoyer une solution avec une probabilité supérieure à ε_F/q_{sig}. En effet, à partir d'une réduction $\mathcal{R}$ on peut résoudre le problème Π avec probabilité supérieure à environ $\varepsilon_R - \varepsilon_F/q_{sig}$, en à peu près le même temps de calcul. Donc, si résoudre le problème Π est difficile, cette probabilité doit être négligeable, et la probabilité de succès ε_R de $\mathcal{R}$ ne peut pas être supérieure à environ ε_F/q_{sig}.

Théorème 10.2.2. *Soit $\mathcal{R}$ une réduction qui $(t_R, q_{hash}, q_{sig}, \varepsilon_R, \varepsilon_F)$-réduit la résolution de Π au cassage d'un schéma de signature hache-et-signe à signature unique. A partir de $\mathcal{R}$ on peut construire un algorithme qui (t_A, ε_A)-résout Π, avec:*

$$t_A = 2 \cdot t_R + (q_{hash} + q_{sig}) \cdot \mathcal{O}(k) \tag{10.1}$$

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1)}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{10.2}$$

En conséquence, si Π est (t_A, ε_A)-difficile, il n'y a pas de réduction qui $(t_R, q_{hash}, q_{sig}, \varepsilon_R, \varepsilon_F)$-réduit la résolution de Π au cassage du schéma de signature.

Preuve. A partir d'une réduction $\mathcal{R}$ qui $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-réduit résoudre Π à casser le schéma de signature $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$, on construit un algorithme $\mathcal{A}$ qui (t_A, ε_A)-résout Π

On applique la technique décrite dans le paragraphe précédent: d'abord on effectue une requête de signature auprès de $\mathcal{R}$ pour un message M, ensuite on relance $\mathcal{R}$ avec le même ruban aléatoire, mais cette fois sans faire la requête de signature pour M, et on envoie à $\mathcal{R}$ la signature de M comme forge. C'est une forge valide pour $\mathcal{R}$ car après que $\mathcal{R}$ a été relancé, la signature du message M n'a pas été demandée.

En fait, il se peut que la réduction $\mathcal{R}$ ne réponde pas à la requête de signature du message M si c'est la première requête de signature, mais réponde si il y a déjà eu d'autres requêtes de signature auparavant. On simule donc le forgeur de façon légèrement différente: on sélectionne d'abord q_{sig} messages $M_1, \ldots, M_{q_{sig}}$ et un entier i aléatoire entre un et q_{sig}, et on demande à $\mathcal{R}$ la signature des messages $M_1, \ldots, M_{i-1}, M$, successivement. Ensuite on relance $\mathcal{R}$ avec le même ruban aléatoire et on demande la signature des messages $M_1, \ldots, M_{q_{sig}}$; finalement on renvoie comme forge la signature de M.

Formellement, on construit un algorithme $\mathcal{A}$ qui reçoit en entrée une instance I d'un problème Π et renvoie une solution z de I en utilisant une réduction $\mathcal{R}$ pour le schéma de signature. L'algorithme $\mathcal{A}$ va simuler un forgeur par rapport à $\mathcal{R}$. L'algorithme $\mathcal{A}$ sélectionne arbitrairement q_{hash} messages distincts $M_1, \ldots, M_{q_{hash}}$ de taille $\mathcal{O}(k)$. Les messages $M_1, \ldots, M_{q_{hash}}$ peuvent être choisis arbitrairement car dans la définition de $\mathcal{R}$, on ne suppose rien sur la façon dont le forgeur choisit les messages. L'algorithme $\mathcal{A}$ va faire des requêtes de signature sur un sous-ensemble

de ces q_{hash} messages. On note $(\alpha_1, \ldots, \alpha_i)$ la suite des requêtes de signature des messages $M_{\alpha_1}, \ldots, M_{\alpha_i}$ dans cet ordre, où $\alpha_1, \ldots, \alpha_i$ sont des entiers compris entre un et q_{hash}. Par exemple, on dit que l'on fait les requêtes de signature correspondant à $(4, 3)$ si on commence par requérir la signature de M_4 pour ensuite demander la signature de M_3.

Algorithme $\mathcal{A}$ utilisant $\mathcal{R}$:
Entrée: une instance I et (q_{hash}, q_{sig}) où $I \leftarrow \mathtt{Gen}\Pi(1^k)$.
Sortie: une solution z de I.

1. Envoyer l'instance I à $\mathcal{R}$.
2. Recevoir de $\mathcal{R}$ la clef publique pk.
3. Exécuter `Hashing` pour les q_{hash} messages $M_1, \ldots, M_{q_{hash}}$, effectuer les requêtes de hachage correspondantes auprès de $\mathcal{R}$, et obtenir les hachés correspondants $h_1, \ldots, h_{q_{hash}}$.
4. Sélectionner aléatoirement un entier $\beta \in [1, q_{hash}]$.
5. Sélectionner aléatoirement et indépendamment une suite α de q_{sig} entiers dans $[1, q_{hash}] \setminus \{\beta\}$. On note $\alpha = (\alpha_1, \ldots, \alpha_{q_{sig}})$.
6. Sélectionner un entier aléatoire $i \in [1, q_{sig}]$.
7. Définir la suite de i entiers $\alpha' = (\alpha_1, \ldots, \alpha_{i-1}, \beta)$. Si $i = 1$, on prend $\alpha' = (\beta)$.
8. Effectuer auprès de $\mathcal{R}$ les i requêtes de signature correspondant à α' et recevoir de $\mathcal{R}$ les signatures correspondantes, la dernière étant la signature s_β de M_β.
9. Relancer $\mathcal{R}$ avec le même ruban aléatoire ω.
10. Envoyer l'instance I à $\mathcal{R}$.
11. Recevoir de $\mathcal{R}$ la clef publique pk.
12. Exécuter `Hashing` pour les q_{hash} messages $M_1, \ldots, M_{q_{hash}}$, effectuer les requêtes de hachage correspondantes auprès de $\mathcal{R}$, et obtenir les hachés correspondants $h_1, \ldots, h_{q_{hash}}$.
13. Effectuer auprès de $\mathcal{R}$ les q_{sig} requêtes de signature correspondant à α.
14. Si $\mathcal{R}$ a répondu à toutes les requêtes de signatures, alors avec probabilité ε_F, envoyer (M_β, s_β) en tant que forge à $\mathcal{R}$.
15. Recevoir de $\mathcal{R}$ une solution z et renvoyer z.

L'algorithme $\mathcal{A}$ obtient à l'étape 8 la signature s_β du message M_β, et on a donc $\mathtt{Verify}_{pk}(M_\beta, s_\beta) = 1$. Ensuite $\mathcal{R}$ est relancé avec le même ruban aléatoire ω, donc $\mathcal{R}$ va renvoyer la même clef publique pk à l'étape 11. De plus, `Hashing` étant un algorithme déterministe, $\mathcal{R}$ reçoit les mêmes requêtes de hachage à l'étape 12. Etant exécuté avec le même ruban aléatoire, $\mathcal{R}$ donne la même réponse aux requêtes de hachage, et donc le même haché h_β est obtenu pour le message M_β à l'étape 12.

En répondant aux requêtes de hachage et de signature, $\mathcal{R}$ définit un oracle aléatoire $H \in \mathcal{H}_k$. La suite des requêtes de hachage et de signature effectuées par $\mathcal{A}$ est différente avant que $\mathcal{R}$ ne soit relancé et après qu'il est relancé. En conséquence, si $\mathcal{R}$ définit un oracle aléatoire $H \in \mathcal{H}_k$ avant qu'il ne soit relancé, il peut définir un autre oracle aléatoire $H' \in \mathcal{H}_k$ après qu'il a été relancé. Cependant, le même

haché h_β pour M_β est obtenu avant et après que $\mathcal{R}$ ne soit relancé, et donc on a $h_\beta \leftarrow \texttt{Hashing}_{pk}^{H}(M_\beta)$ et $h_\beta \leftarrow \texttt{Hashing}_{pk}^{H'}(M_\beta)$. Comme $\texttt{Verify}_{pk}^{H}(M_\beta, s_\beta) = 1$, on déduit du théorème 10.2.1 que $\texttt{Verify}_{pk}^{H'}(M_\beta, s_\beta) = 1$. En conséquence, s_β est encore une signature valide de M_β après que $\mathcal{R}$ a été relancé, signature que $\mathcal{A}$ envoie à $\mathcal{R}$ en tant que forge à l'étape 14. C'est une forge valide pour $\mathcal{R}$ car la signature de M_β n'a pas été demandée à $\mathcal{R}$ après qu'il a été relancé. Finalement $\mathcal{A}$ reçoit de $\mathcal{R}$ une solution z à l'étape 15 et renvoie z.

On note D la distribution induite en sélectionnant aléatoirement $i \in [1, q_{sig}]$, $\beta \in [1, q_{sig}]$, $\alpha \in ([1, q_{hash}] \setminus \{\beta\})^{q_{sig}}$, l'instance $I \leftarrow \texttt{Gen}\Pi(1^k)$ et le ruban aléatoire w de $\mathcal{R}$. On note $\mathcal{R}(\omega)$ l'algorithme déterministe correspondant à $\mathcal{R}$ exécuté avec le ruban aléatoire ω. On note $\mathcal{U}(\omega, I)$ l'ensemble des suites de requêtes de signature auxquelles $\mathcal{R}(\omega)$ répond, en temps inférieur à t_R, lorsque $\mathcal{R}$ reçoit l'instance I et les q_{hash} requêtes de hachage correspondant à $M_1, \ldots, M_{q_{hash}}$. Par exemple, $(3, 2) \in \mathcal{U}(\omega, I)$ signifie que $\mathcal{R}$ peut répondre à la requête de signature de M_3 et M_2, dans cet ordre. L'ensemble $\mathcal{U}(\omega, I)$ satisfait la propriété suivante: pour tout $(\alpha_1, \ldots, \alpha_j) \in \mathcal{U}(\omega, I)$, on a $(\alpha_1, \ldots, \alpha_{j-1}) \in \mathcal{U}(\omega, I)$. On note `ans` l'événement correspondant à $(\alpha_1, \ldots, \alpha_{q_{sig}}) \in \mathcal{U}(\omega, I)$, et `ans'` l'événement correspondant à $(\alpha_1, \ldots, \alpha_{i-1}, \beta) \in \mathcal{U}(\omega, I)$. L'événement `ans` correspond à $\mathcal{R}$ répondant à toutes les requêtes de signature à l'étape 13, tandis que l'événement `ans'` correspond à $\mathcal{R}$ répondant à toutes les requêtes de signatures à l'étape 8.

Considérons maintenant le forgeur suivant $\mathcal{F}^*$ pour le schéma de signature $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$. $\mathcal{F}^*$ sélectionne les mêmes q_{hash} messages $M_1, \ldots, M_{q_{hash}}$. Le forgeur sélectionne aléatoirement $\beta \in [1, q_{hash}]$ et une séquence aléatoire α de q_{sig} entiers pris dans $[1, q_{hash}] \setminus \{\beta\}$. Après avoir reçu la clef publique pk, $\mathcal{F}^*$ exécute `Hashing` pour les q_{hash} messages $M_1, \ldots, M_{q_{hash}}$, effectue auprès de $\mathcal{R}$ les requêtes de hachage correspondantes, et obtient les hachés $h_1, \ldots, h_{q_{hash}}$. Ensuite le forgeur recherche exhaustivement l'unique signature s_β du message M_β telle que $\texttt{Verify}_{pk}(M_\beta, s_\beta) = 1$. D'après le théorème 10.2.1, étant donné h_β, la signature de M_β ne dépend pas de l'oracle aléatoire $H \in \mathcal{H}_k$ et donc $\mathcal{F}^*$ n'a pas besoin d'effectuer de nouvelles requêtes de hachage pour calculer $\texttt{Verify}_{pk}(M_\beta, s_\beta)$. Ensuite, $\mathcal{F}^*$ effectue q_{sig} requêtes de signature auprès de $\mathcal{R}$ correspondant à α. Si $\mathcal{R}$ a répondu à toutes les requêtes de signatures, alors avec probabilité ε_F, le forgeur renvoie la forge (M_β, s_β) à $\mathcal{R}$. Sinon, le forgeur s'arrête et ne renvoie pas de forge.

On voit clairement que $\mathcal{F}^*$ $(t_{F^*}, q_{hash}, q_{sig}, \varepsilon_F)$-casse la schéma de signature. En conséquence, la réduction $\mathcal{R}$ réussit avec probabilité au moins ε_R lorsqu'elle interagit avec $\mathcal{F}^*$. Le temps de calcul de $\mathcal{F}^*$ est exponentiel en le paramètre de sécurité k, puisque $\mathcal{F}^*$ recherche exhaustivement la signature s_β de M_β, mais cela n'a pas d'importance ici puisque l'algorithme $\mathcal{A}$ n'utilise pas le forgeur $\mathcal{F}^*$; au lieu de cela, l'algorithme $\mathcal{A}$ simule $\mathcal{F}^*$ vis à vis de $\mathcal{R}$.

L'algorithme $\mathcal{A}$ envoie avec probabilité ε_F une forge à $\mathcal{R}$ à l'étape 14 s'il a reçu la signature de M_β à l'étape 8, donc si l'événement `ans'` est vrai, et si $\mathcal{R}$ a répondu aux requêtes de signature à l'étape 13, ce qui se produit si l'événement `ans` est vrai. En

conséquence, après que $\mathcal{R}$ a été relancé, $\mathcal{R}$ échange exactement les mêmes données en interagissant avec $\mathcal{A}$ qu'en interagissant avec $\mathcal{F}^*$, sauf si l'événement ans est vrai et que l'événement ans' est faux: dans ce cas, $\mathcal{F}$ renvoie une forge avec probabilité ε_F tandis que $\mathcal{A}$ ne renvoie pas de forge. Cela se produit donc avec probabilité:

$$\varepsilon_F \cdot \Pr_D[\mathtt{ans} \wedge \neg \mathtt{ans'}]$$

En conséquence, $\mathcal{A}$ réussit avec probabilité supérieure à :

$$\varepsilon_R - \varepsilon_F \cdot \Pr_D[\mathtt{ans} \wedge \neg \mathtt{ans'}] \tag{10.3}$$

En utilisant le lemma suivant avec $Q = \mathcal{U}(\omega, I)$, $n = q_{sig}$ et $k = q_{hash}$, on obtient:

$$\Pr_D[\mathtt{ans} \wedge \neg \mathtt{ans'}] \leq \frac{\exp(-1)}{q_{sig}} \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{10.4}$$

Lemme 10.2.1. *Soit Q un ensemble de suites d'au plus n entiers dans $[1,k]$, tel que pour toute suite $(\alpha_1, \ldots, \alpha_j) \in Q$, on a $(\alpha_1, \ldots, \alpha_{j-1}) \in Q$. Alors on a:*

$$\Pr_{\substack{i \leftarrow [1,n] \\ (\alpha_1,\ldots,\alpha_n,\beta) \leftarrow [1,k]^{n+1}}} [(\alpha_1, \ldots, \alpha_n) \in Q \wedge (\alpha_1, \ldots, \alpha_{i-1}, \beta) \notin Q] \leq \frac{\exp(-1)}{n}$$

Preuve. On montre par récurrence sur n qu'en notant D_n la distribution:

$$D_n = \left\{ \begin{array}{rcl} i & \leftarrow & [1,n] \\ (\alpha_1, \ldots, \alpha_n) & \leftarrow & [1,k]^n \\ \beta & \leftarrow & [1,k] \end{array} \right.$$

et en notant pour tout $j \in [1,n]$ les événements:

$$\begin{array}{rcl} A_j & : & (\alpha_1, \ldots, \alpha_{j-1}, \alpha_j) \in Q \\ B_j & : & (\alpha_1, \ldots, \alpha_{j-1}, \beta) \in Q \end{array}$$

avec $A_j \Rightarrow A_{j-1}$ pour tout $j \in [2,n]$, alors on a:

$$\Pr_{D_n}[A_n \wedge \neg B_i] \leq \Pr_{D_n}[A_n] \cdot \left(1 - \Pr_{D_n}[A_n]^{1/n}\right) \tag{10.5}$$

L'équation (10.5) est clairement vérifiée pour $n = 1$. En supposant que l'équation (10.5) est vérifiée pour $n-1$, on montre qu'elle est vérifiée pour n. Par la suite, les probabilités sont prises suivant la distribution D_n. Comme i est pris aléatoirement entre un et n, on a:

$$\Pr[A_n \wedge \neg B_i] = \frac{1}{n} \Pr[A_n \wedge \neg B_1] + \frac{n-1}{n} \Pr[A_n \wedge \neg B_i | i \geq 2] \tag{10.6}$$

Les événements A_n et B_1 sont indépendants, ce qui donne:

$$\Pr[A_n \wedge \neg B_1] = \Pr[A_n] \cdot \Pr[\neg B_1] = \Pr[A_n] \cdot (1 - \Pr[A_1]) \tag{10.7}$$

En notant $L_1 = \{a_1 \in [1,k] \mid (a_1) \in Q\}$, on a:

$$\Pr[A_n \wedge \neg B_i | i \geq 2] = \frac{1}{k} \sum_{a_1 \in L_1} \Pr\left[A_n \wedge \neg B_i | \alpha_1 = a_1 \wedge i \geq 2\right]$$

Conditionné sur $i \geq 2$ et $\alpha_1 = a_1$, les événements A_j et B_j sont sélectionnés suivant la distribution:

$$D'_{n-1} = \left\{ \begin{array}{rcl} i & \leftarrow & [2,n] \\ (\alpha_2, \ldots, \alpha_n) & \leftarrow & [1,k]^{n-1} \\ \beta & \leftarrow & [1,k] \end{array} \right.$$

Par conséquent, en utilisant l'équation (10.5) pour $n-1$, on obtient:

$$\Pr\left[A_n \wedge \neg B_i | i \geq 2\right] \leq \frac{1}{k} \sum_{a_1 \in L_1} \Pr[A_n | \alpha_1 = a_1] \cdot \left(1 - \Pr[A_n | \alpha_1 = a_1]^{\frac{1}{n-1}}\right) \tag{10.8}$$

On a, en utilisant $\Pr[A_1] = \#L_1/k$:

$$\frac{1}{\#L_1} \sum_{a_1 \in L_1} \Pr[A_n | \alpha_1 = a_1] = \Pr[A_n \wedge A_1] / \Pr[A_1] = \Pr[A_n | A_1]$$

D'après l'inégalité:

$$\frac{1}{t} \sum_{i=1}^{t} x_i^r \geq \left(\frac{1}{t} \sum_{i=1}^{t} x_i\right)^r \quad \text{pour} \quad r \geq 1$$

on obtient

$$\frac{1}{\#L_1} \sum_{a_1 \in L_1} \Pr[A_n | \alpha_1 = a_1]^{\frac{n}{n-1}} \geq \Pr[A_n | A_1]^{\frac{n}{n-1}}$$

et alors

$$\frac{1}{k} \sum_{a_1 \in L_1} \Pr[A_n | \alpha_1 = a_1]^{\frac{n}{n-1}} \geq \Pr[A_1] \cdot \Pr[A_n | A_1]^{\frac{n}{n-1}} \geq \Pr[A_n] \cdot \Pr[A_n | A_1]^{\frac{1}{n-1}}$$

ce qui donne en utilisant (10.8):

$$\Pr[A_n \wedge \neg B_i | i \geq 2] \leq \Pr[A_n] \cdot \left(1 - \Pr[A_n | A_1]^{\frac{1}{n-1}}\right)$$

Alors en utilisant les équations (10.6) et (10.7), on obtient:

$$\Pr[A_n \wedge \neg B_i] \leq \Pr[A_n] \left(1 - \frac{\Pr[A_1]}{n} - \frac{n-1}{n} \Pr[A_n | A_1]^{\frac{1}{n-1}}\right)$$

En utilisant l'inégalité bien connue $P \leq S$ entre la moyenne arithmétique S et la moyenne géométrique P, on obtient:

$$\Pr[A_n \wedge \neg B_i] \leq \Pr[A_n] \left(1 - \Pr[A_n]^{1/n}\right)$$

ce qui montre que l'équation (10.5) est vérifiée pour n et termine la preuve par récurrence.

Ensuite, en notant $x = \Pr[A_n]^{1/n}$ et en utilisant l'inégalité $x^n \cdot (1-x) \leq \exp(-1)/n$ pour $x \in [0,1]$, on obtient:

$$\Pr[A_n \wedge \neg B_i] \leq \frac{\exp(-1)}{n}$$

□

Le terme $(1 - q_{sig}/q_{hash})$ dans l'équation (10.4) est dû au fait que $\mathcal{A}$ sélectionne $\alpha_1, \ldots, \alpha_{q_{sig}}$ dans $[1, q_{hash}] \setminus \{\beta\}$, tandis que dans le lemme 10.2.1 les entiers α_i sont pris dans $[1, q_{hash}]$. A partir des équations (10.3) et (10.4) on obtient que $\mathcal{A}$ réussit avec probabilité supérieure à ε_A donné par (10.2). □

10.2.3 Extension aux réductions exécutant le forgeur plus d'une fois

Dans le paragraphe précédent, on a considéré des réductions exécutant le forgeur seulement une fois. On a prouvé qu'une réduction ne peut pas réussir avec probabilité supérieure à approximativement ε_F/q_{sig}, sinon la réduction peut être utilisée pour résoudre un problème difficile. Que se passe-t'il si la réduction peut exécuter le forgeur plus d'une fois ? Bien sûr, si la réduction de la preuve de sécurité de FDH du chapitre précédent exécute le forgeur r fois, sa probabilité de succès sera d'environ $r \cdot \varepsilon_F/q_{sig}$, et le temps de calcul total sera environ r fois le temps de calcul du forgeur. Mais il se pourrait qu'il existe une meilleure réduction qui donnerait un meilleur compromis temps/probabilité. Par exemple, si une réduction pour FDH pouvait réussir avec probabilité environ ε_F en exécutant un forgeur au plus $\log q_{sig}$ fois, alors FDH serait presque aussi sûr qu'inverser RSA. De plus, la réduction pourrait relancer le forgeur avec le même ruban aléatoire mais avec différentes entrées, comme pour les schémas de signature basés sur les preuves de connaissance [72, 69].

Dans ce qui suit, on montre que qu'il n'y a pas de meilleur compromis temps/probabilité: pour un schéma de signature hache-et-signe à signature unique, une réduction pouvant relancer le forgeur au plus r fois ne peut pas réussir avec probabilité supérieure à environ $r \cdot \varepsilon_F/q_{sig}$.

Définition 10.2.9. *On dit qu'une réduction $\mathcal{R}$ $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R, r)$-réduit résoudre Π à casser le schéma de signature* (`Gen`, `Sign`, `Verify`) *si après avoir reçu une instance I telle que $I \leftarrow Gen\Pi(1^k)$ et exécuté au plus r fois tout forgeur qui $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-casse le schéma de signature, la réduction $\mathcal{R}$ renvoie une solution z de I avec probabilité au moins $\varepsilon_R(k)$ après un temps de calcul additionnel d'au plus $t_R(k)$ pour tout $k \in \mathbb{N}$. La réduction est autorisée à relancer le forgeur avec le même ruban aléatoire.*

De plus, la réduction pourrait rembobiner le forgeur vers un état antérieur S et ensuite l'exécuter avec des entrées différentes. Ceci est équivalent à relancer le forgeur avec le même ruban aléatoire jusqu'à ce que l'état S soit atteint, et donc on ne considère pas ce cas.

En utilisant la même technique que précédemment, on montre qu'à partir d'une réduction $\mathcal{R}$ qui peut relancer le forgeur au plus r fois, on peut résoudre le problème Π avec probabilité supérieure à environ $\varepsilon_R - \varepsilon_F \cdot r/q_{sig}$ avec environ le même temps de calcul. Donc, si résoudre le problème Π est difficile, cette probabilité doit être négligeable, et la probabilité de succès de $\mathcal{R}$ ne peut pas être supérieure à environ $\varepsilon_F \cdot r/q_{sig}$.

Théorème 10.2.3. *Soit $\mathcal{R}$ une réduction qui $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R, r)$-réduit résoudre Π à casser un schéma de signature hache-et-signe à signature unique. A partir de $\mathcal{R}$ on peut construire un algorithme qui (t_A, ε_A)-résout Π, avec:*

$$t_A = (r+1) \cdot (t_R + (q_{hash} + q_{sig}) \cdot \mathcal{O}(k)) \tag{10.9}$$

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{10.10}$$

Preuve. La preuve est très similaire à la preuve du théorème 10.2.2. Supposons dans un premier temps que la réduction ne peut pas relancer le forgeur avec le même ruban aléatoire. La réduction peut seulement exécuter le forgeur au plus r fois. On dit qu'une réduction est dans le j-ème tour si la réduction a déjà exécuté le forgeur $j-1$ fois; il y a donc au plus r tours. Dans le premier tour de la réduction, on applique la même technique que précédemment: on demande la signature du message M_1, on relance ensuite la réduction avec le même ruban aléatoire, et on envoie la signature de M_1 comme forge. Ensuite la réduction se trouve dans le second tour, et commence à interagir pour la deuxième fois avec le forgeur. On demande alors la signature du message M_2, et on rembobine la réduction au même état où elle se trouvait au début du deuxième tour. Ceci équivaut à relancer la réduction avec le même ruban aléatoire et à réaliser la même interaction que dans le premier tour. Ensuite on envoie la signature du message M_2 comme forge. Ceci est une forge valide parce qu'une fois la réduction rembobinée, la signature de M_2 n'a pas été demandée. On procède ensuite de la même façon pour les tours suivants: au j-ème tour, on demande la signature du message M_j, puis on rembobine la réduction au début du j-ème tour, et on envoie la signature de M_j comme forge. Par cette méthode, on peut ainsi simuler un forgeur exécuté par la réduction au plus r fois.

Formellement, on construit un algorithme $\mathcal{A}'$ qui reçoit en entrée une instance I du problème Π et renvoie en sortie une solution z de I en utilisant $\mathcal{R}$. $\mathcal{A}'$ sélectionne arbitrairement q_{hash} messages distincts $M_1, \ldots, M_{q_{hash}}$, de longueur $\mathcal{O}(k)$. Notre construction de $\mathcal{A}'$ utilise l'algorithme $\mathcal{A}$ défini dans la preuve du théorème 10.2.2.

Algorithme $\mathcal{A}'$ utilisant $\mathcal{R}$:
Entrée: une instance I et (q_{hash}, q_{sig}) où $I \leftarrow \mathtt{Gen}\Pi(1^k)$.
Sortie: une solution z de I.

1. Fixer $j \leftarrow 1$.
2. Envoyer l'instance I à $\mathcal{R}$.
3. Répéter jusqu'à ce qu'une solution z soit reçu de $\mathcal{R}$ ou si $j > r$:
   $\mathcal{R}$ est maintenant au début du j-ème tour.
   Exécuter les étapes 2 à 8 de l'algorithme $\mathcal{A}$.
   Rembobiner $\mathcal{R}$ jusqu'au début du j-ème tour.
   Exécuter les étapes 11 à 14 de l'algorithme $\mathcal{A}$.
   Exécuter $j \leftarrow j + 1$.
4. Renvoyer une solution z de I.

On considère le même forgeur $\mathcal{F}^*$ que dans le théorème 10.2.2, qui $(t_{F^*}, q_{hash}, q_{sig}, \varepsilon_F)$-casse le schéma de signature. D'après la définition, la réduction $\mathcal{R}$, en exécutant le forgeur $\mathcal{F}^*$ au plus r fois, réussit avec probabilité au moins ε_R.

Soit $\mathtt{ans}'_j$ l'événement correspondant à la réduction répondant au j-ème tour à toutes les requêtes de signature avant d'être rembobinée, et soit $\mathtt{ans}_j$ l'événement correspondant à la réduction répondant aux requêtes de signature après qu'elle a été rembobinée. En interagissant avec notre simulation, la réduction $\mathcal{R}$ échange exactement les mêmes données au j-ème tour que lorsqu'elle interagit avec le forgeur $\mathcal{F}^*$, sauf si l'événement $\mathtt{ans}_j$ est vrai alors que l'événement $\mathtt{ans}'_j$ est faux: dans ce cas, $\mathcal{F}^*$ renvoie une forge avec probabilité ε_F, tandis que $\mathcal{A}$ ne renvoie pas de forge. Cela se produit avec probabilité:

$$\varepsilon_F \cdot \Pr[\mathtt{ans}_j \wedge \neg \mathtt{ans}'_j]$$

Comme il y a au plus r tours, $\mathcal{R}$ renvoie une solution avec probabilité supérieure à:

$$\varepsilon_R - \sum_{j=1}^{r} \varepsilon_F \cdot \Pr[\mathtt{ans}_j \wedge \neg \mathtt{ans}'_j]$$

En utilisant le lemme 10.2.1, on obtient pour tout j:

$$\Pr[\mathtt{ans}_j \wedge \neg \mathtt{ans}'_j] \leq \frac{\exp(-1)}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1}$$

En conséquence, $\mathcal{A}'$ réussit avec probabilité supérieure à:

$$\varepsilon_{A'} = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{10.11}$$

La réduction $\mathcal{R}$ est relancée au plus r fois, donc le temps de calcul de $\mathcal{A}'$ est au plus $r + 1$ fois le temps de calcul de $\mathcal{R}$, plus le temps nécessaire pour transmettre les requêtes de signature et de hachage, ce qui donne:

$$t_{A'} = (r + 1) \cdot (t_R + (q_{hash} + q_{sig}) \cdot \mathcal{O}(k)) \tag{10.12}$$

Supposons maintenant que la réduction $\mathcal{R}$ ait la possibilité de relancer le forgeur avec le même ruban aléatoire. On montre que cela ne l'aide en rien. En effet, si la réduction envoie la même clef publique et donne la même réponse aux requêtes de hachage, la réduction voit exactement le même échange obtenu avant que $\mathcal{F}^*$ ne soit relancé. Si au contraire la réduction envoie une clef publique différente ou fournit une réponse différente aux requêtes de hachage, le forgeur $\mathcal{F}^*$ peut faire des requêtes de signatures pour d'autres messages sélectionnés aléatoirement, et forger la signature d'un autre message pris aléatoirement, ce que $\mathcal{R}$ ne peut pas distinguer d'un forgeur $\mathcal{F}^*$ exécuté avec un nouveau ruban aléatoire. Donc dans les deux cas, cela n'aide pas la réduction de relancer le forgeur avec le même ruban aléatoire.

Formellement, à partir d'une réduction $\mathcal{R}$ qui $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R, r)$-réduit Π à casser le schéma de signature, on construit une réduction $\mathcal{R}'$ qui réussit à résoudre Π avec probabilité supérieure à $\varepsilon_{R'} = \varepsilon_R$ après un temps de calcul additionnel de $t_{R'}$ en exécutant au plus r fois le forgeur $\mathcal{F}^*$, avec:

$$t_{R'} = t_R + (r+1) \cdot (q_{hash} + q_{sig}) \cdot \mathcal{O}(k)$$

La différence est que la nouvelle réduction $\mathcal{R}'$ n'est pas autorisée à relancer le forgeur $\mathcal{F}^*$ avec le même ruban aléatoire; au lieu de cela, $\mathcal{R}'$ exécute toujours $\mathcal{F}^*$ avec un nouveau ruban aléatoire.

La réduction $\mathcal{R}'$ va se comporter comme une interface entre $\mathcal{R}$ et $\mathcal{F}^*$. La réduction $\mathcal{R}'$ reçoit en entrée un instance I et envoie I à $\mathcal{R}$. La réduction $\mathcal{R}'$ fait suivre la clef publique envoyée par $\mathcal{R}$ à $\mathcal{F}^*$. Lorsque le forgeur fait une requête de hachage ou de signature, $\mathcal{R}'$ fait suivre la requête auprès de $\mathcal{R}$ et la réponse pour $\mathcal{F}^*$. $\mathcal{R}'$ transmet aussi la forge à $\mathcal{R}$. Lorsque la réduction $\mathcal{R}$ relance le forgeur avec le même ruban aléatoire, $\mathcal{R}'$ continue à exécuter le forgeur $\mathcal{F}^*$ si $\mathcal{R}$ envoie la même clef publique et la même réponse aux requêtes de hachage. Sinon $\mathcal{R}$ exécute une nouvelle fois le forgeur $\mathcal{F}^*$ avec un nouveau ruban aléatoire. Dans les deux cas la réduction $\mathcal{R}$ voit le même échange de données que si le forgeur $\mathcal{F}'$ avait été relancé avec le même ruban aléatoire, et donc $\mathcal{R}$ renvoie une solution z avec probabilité supérieure à ε_R. En conséquence, la réduction $\mathcal{R}'$ réussit avec probabilité supérieure à $\varepsilon_{R'} = \varepsilon_R$. Le temps de calcul de $\mathcal{R}'$ est le temps de calcul de $\mathcal{R}$ plus le temps nécessaire pour transmettre les requêtes de hachage et de signature. Finalement en utilisant l'algorithme $\mathcal{A}'$ avec cette réduction $\mathcal{R}'$, le problème Π est résolu avec probabilité au moins $\varepsilon_{A'}$ en temps inférieur à $t_{A'}$, où $\varepsilon_{A'}$ et $t_{A'}$ sont donnés par les équations (10.11) et (10.12), ce qui termine la preuve.

10.2.4 Application au schéma de signature Full Domain Hash

Le schéma de signature Full Domain Hash que nous avons déjà présenté est un schéma de signature hache-et-signe à signature unique. Le théorème précédent montre qu'une réduction exécutant un forgeur au plus une fois ne peut pas avoir une probabilité de succès supérieure à environ $\varepsilon_R \simeq \varepsilon_F \cdot r / q_{sig}$, en supposant qu'inverser

RSA soit difficile (q_{sig} est généralement beaucoup plus petit que q_{hash}, ce qui rend le facteur $(1 - q_{sig}/q_{hash})$ proche de un). Cela montre que la preuve de sécurité de FDH du chapitre précédent est optimale. En particulier, FDH ne peut pas avoir une preuve de sécurité telle que $\varepsilon_F \simeq \varepsilon_R$ avec $r = 1$, contrairement à PSS. On ne peut donc pas montrer que FDH possède exactement le même niveau de sécurité que RSA.

Supposons par exemple qu'un forgeur soit autorisé à réaliser au plus $q_{hash} = 2^{60}$ requêtes de hachage et $q_{sig} = 2^{30}$ requêtes de signature, et que pour un paramètre de sécurité k donné et une borne en temps t_I la probabilité d'inverser RSA soit inférieure à $\varepsilon_I = 2^{-80}$. Alors une réduction d'inverser RSA à casser FDH, exécutant un forgeur une seule fois, ne peut pas inverser RSA avec probabilité supérieure à $\varepsilon_R = 2^{-80} + 2^{-31} \cdot \varepsilon_F$, en un temps de calcul additionnel d'environ $t_I/2$, où ε_F est la probabilité que le forgeur renvoie une forge.

10.3 Preuves de sécurité dans le modèle standard

La même technique s'applique aux preuves de sécurité dans le modèle standard, et la même borne supérieure en $1/q_{sig}$ se vérifie pour les schémas de signature à signature unique. Un schéma de signature à signature unique est nécessairement sans mémoire: la signature d'un message ne dépend pas des messages précédemment signés. Les définitions de sécurité pour un schéma de signature sont analogues dans le modèle standard.

Définition 10.3.1. *On dit qu'un forgeur $\mathcal{F}$ $(t, q_{sig}, \varepsilon)$-casse le schéma de signature* (`Gen`, `Sign`, `Verify`) *si après au plus $q_{sig}(k)$ requêtes de signature et un temps de calcul $t(k)$, il renvoie une forge avec probabilité supérieure à $\varepsilon(k)$ pour tout $k \in \mathbb{N}$.*

Définition 10.3.2. *Un schéma de signature* (`Gen`, `Sign`, `Verify`) *est $(t, q_{sig}, \varepsilon)$-sûr s'il n'existe pas de forgeur qui $(t, q_{sig}, \varepsilon)$-casse le schéma de signature.*

Définition 10.3.3 (schéma de signature à signature unique). *Un schéma de signature est dit à signature unique si pour toute clef publique pk et tout message M, il existe une unique signature x telle que $\texttt{Verify}_{pk}(M, x) = 1$.*

On suppose que la sécurité du schéma de signature (`Gen`, `Sign`, `Verify`) est basée sur la difficulté d'un problème Π, de sorte qu'il existe une réduction entre résoudre Π et casser le schéma de signature.

Définition 10.3.4. *Une réduction $\mathcal{R}$ entre résoudre $(Gen\Pi, D, S)$ et casser* (`Gen`, `Sign`, `Verify`) *est un algorithme prenant en entrée une instance I et q_{sig}, où $I \leftarrow Gen\Pi(1^k)$, et renvoyant une solution z pour I. L'algorithme de réduction interagit avec un forgeur $\mathcal{F}$ pour* (`Gen`, `Sign`, `Verify`)*, qui renvoie une forge après au plus q_{sig} requêtes de signature. L'algorithme de réduction répond aux requêtes de signature de $\mathcal{F}$.*

Définition 10.3.5. *On dit qu'une réduction $\mathcal{R}$ $(t_R, q_{sig}, \varepsilon_F, \varepsilon_R, r)$-réduit résoudre Π à casser le schéma de signature* (`Gen`, `Sign`, `Verify`) *si après avoir reçu une instance I telle que $I \leftarrow Gen\Pi(1^k)$ et exécuté au plus r fois tout forgeur qui $(t_F, q_{sig}, \varepsilon_F)$-casse le schéma de signature, la réduction $\mathcal{R}$ renvoie une solution z de I avec probabilité supérieure à $\varepsilon_R(k)$ après un temps additionnel de $t_R(k)$ pour tout $k \in \mathbb{N}$. La réduction peut relancer le forgeur avec le même ruban aléatoire.*

Le théorème suivant est analogue au théorème 10.2.3. Il prouve que pour tout schéma de signature à signature unique, une réduction de résoudre Π à casser le schéma de signature ne peut pas réussir avec probabilité meilleure qu'environ $r \cdot \varepsilon_F / q_{sig}$, en supposant que le problème Π soit difficile à résoudre. En effet, une meilleure réduction peut être convertie en un algorithme pour résoudre Π, avec approximativement le même temps de calcul.

Théorème 10.3.1. *Soit $\mathcal{R}$ une réduction qui $(t_R, q_{sig}, \varepsilon_F, \varepsilon_R, r)$-réduit résoudre Π à casser un schéma de signature à signature unique. On suppose que la taille de l'espace des messages est supérieure à 2^ℓ. Alors à partir de $\mathcal{R}$ on peut construire un algorithme $\mathcal{A}$ qui (t_A, ε_A)-résout Π, avec:*

$$t_A = (r+1) \cdot (t_R + q_{sig} \cdot \mathcal{O}(k)) \tag{10.13}$$

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{2^\ell}\right)^{-1} \tag{10.14}$$

Preuve. La preuve est très similaire à la preuve du théorème 10.2.3. La seule différence est qu'il n'y a pas de requêtes de hachage, et donc les étapes 3 et 12 de l'algorithme $\mathcal{A}$ du théorème 10.2.2 ne sont pas exécutées. De plus, on remplace dans l'algorithme $\mathcal{A}$ le nombre de requêtes de hachage q_{hash} par la borne inférieure 2^ℓ sur la taille de l'espace des messages; au lieu de sélectionner q_{hash} messages distincts, on sélectionne 2^ℓ messages distincts $M_1, \ldots, M_{2^\ell}$. Le reste de l'algorithme est le même, et la même analyse montre qu'à partir d'une réduction qui réussit avec probabilité supérieure à ε_R après avoir exécuté une fois un forgeur qui casse le schéma de signature avec probabilité au moins ε_F, et avec un temps de calcul additionnel de t_R, on peut construire un algorithme qui (t_A, ε_A)-résout le problème Π, avec:

$$t_A = 2 \cdot t_R + q_{sig} \cdot \mathcal{O}(k)$$

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1)}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{2^\ell}\right)^{-1}$$

Alors, en utilisant la même technique que dans le théorème 10.2.3, à partir d'une réduction exécutant un forgeur au plus r fois et autorisée à relancer le forgeur avec le même ruban aléatoire, on construit un algorithme $\mathcal{A}'$ qui $(t_{A'}, \varepsilon_{A'})$-résout Π, avec:

$$t_{A'} = (r+1) \cdot (t_R + q_{sig} \cdot \mathcal{O}(k))$$

$$\varepsilon_{A'} = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{2^\ell}\right)^{-1}$$

Le schéma de sécurité de Gennaro-Halevi-Rabin possède une preuve de sécurité exacte ($\varepsilon_F \simeq \varepsilon_R$) dans le modèle standard, mais le théorème précédent ne s'applique pas ici car la preuve de sécurité de [48] nécessite que tout message ait plusieurs signatures possibles. C'est aussi le cas du schéma de signature de Cramer-Shoup [36]. Cependant, on peut construire une variante du schéma de Gennaro-Halevi-Rabin prouvée sûre dans le modèle standard, et à signature unique. Cette variante est prouvée sûre pour les messages courts seulement (de l'ordre de 40 bits); on ne sait pas s'il existe un schéma de signature à signature unique qui soit à la fois pratique et prouvé sûr, et atteignant la borne supérieure en $1/q_{sig}$.

La clef publique du schéma de Gennaro-Halevi-Rabin est $N = p \cdot q$ et un entier $y \in \mathbb{Z}_N^*$, où p et q sont des premiers aléatoires de $k/2$-bits et $(p-1)/2$ et $(q-1)/2$ sont aussi premiers. La clef privée est (p, q). Le schéma utilise une fonction de hachage qui renvoie des entiers impairs de taille k_0 bits. Pour signer un message m, on calcule $e = h(m)$ et la signature σ est la racine e-ième de y modulo N, qui s'obtient en utilisant p, q. Pour vérifier la signature σ d'un message m, on calcule $e = h(m)$ et on vérifie que

$$\sigma^e = y \bmod N$$

La sécurité du schéma de Gennaro-Halevi-Rabin est basée sur la difficulté du problème RSA fort:

Définition 10.3.6 (Problème RSA fort). *Etant donné un module RSA aléatoire N et un élément aléatoire $s \in \mathbb{Z}_N^*$, trouver une paire (e, r) avec $e > 1$ telle que $r^e = s \bmod N$.*

On remplace maintenant la fonction de hachage par une fonction injective Ψ prenant en entrée une chaîne de ℓ bits et renvoyant un entier premier. Une telle fonction est construite dans [48]. On obtient ainsi un schéma de signature à signature unique, prouvé sûr dans le modèle standard. Cependant, ce schéma n'est pas utilisable dans la pratique, car seuls des messages très courts peuvent être signés; cela est dû au facteur 2^ℓ dans la borne de temps t_F de l'attaquant. On note $t(\ell)$ le temps nécessaire pour calculer Ψ.

Théorème 10.3.2. *Si le problème RSA fort est (t_I, ε_I)-difficile, le schéma de signature précédent est $(t_F, q_{sig}, \varepsilon_F)$-sûr, avec:*

$$t_I = t_F + \mathtt{poly}(2^\ell, k, t(\ell)) \tag{10.15}$$

$$\varepsilon_I = \frac{\varepsilon_F}{q_{sig}} \cdot \left(1 - \frac{1}{q_{sig}+1}\right)^{q_{sig}+1} \tag{10.16}$$

Preuve. On suppose qu'il existe un attaquant $\mathcal{F}$ qui $(t_F, q_{sig}, \varepsilon_F)$-casse le schéma de signature (`Gen`, `Sign`, `Verify`). On construit un algorithme $\mathcal{I}$ qui résout le problème RSA-fort. $\mathcal{I}$ répond lui-même aux requêtes de l'attaquant. L'espace des messages est $\{0,1\}^\ell$.

Algorithme pour $\mathcal{I}$.
Entrée: (N, s) et (ℓ, q_{sig}), où $N \leftarrow RSA(1^k)$ et $s \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$.
Sortie: (r, e) avec $e > 1$ tel que $r^e = s \bmod N$.

1. Soit $E \leftarrow 1$.
2. Pour tous les messages $M_i \in \{0,1\}^\ell$, faire:
   Lancer une pièce c_i avec biais γ, paramètre qui sera déterminé par la suite.[1]
   Si $c_i = 0$ alors $E \leftarrow E \cdot \Psi(M_i)$.
3. Soit $y \leftarrow s^E \bmod N$.
4. Envoyer la clef publique (N, y) à $\mathcal{F}$.
5. Si $\mathcal{F}$ effectue une requête de signature pour M_i:
   Si $c_i = 0$ alors retourner $s^{E/\Psi(M_i)} \bmod N$. Sinon stopper.
6. Si $\mathcal{F}$ renvoie une forge (M_i, x):
   Si $c_i = 0$ alors stopper.
   Si $c_i = 1$ alors $\Psi(M_i) \wedge E = 1$. Soient $a, b \in \mathbb{Z}$ tels que $a \cdot \Psi(M_i) + b \cdot E = 1$.
   Soient $r \leftarrow x^b \cdot s^a \bmod N$ et $e \leftarrow \Psi(M_i)$; renvoyer (r, e).

La probabilité que $\mathcal{I}$ réponde à toutes les requêtes de signature est supérieure à $\gamma^{q_{sig}}$. Finalement $\mathcal{F}$ renvoie une forge avec probabilité ε_F que $\mathcal{I}$ peut utiliser avec probabilité $1-\gamma$ pour renvoyer (r, e). En conséquence $\mathcal{I}$ renvoie (r, e) avec probabilité $\gamma^{q_{sig}} \cdot (1-\gamma) \cdot \varepsilon_F$, qui est maximale pour $\gamma = 1 - 1/(q_{sig}+1)$ et donne (10.16). □

10.4 Conclusion

Nous avons décrit une nouvelle technique permettant d'analyser les preuves de sécurité des schémas de signature. Cette technique à la fois générale et simple permet d'obtenir une borne supérieure sur les preuves de sécurité de schémas de signature à signature unique, à la fois dans le modèle de l'oracle aléatoire et dans le modèle standard.

De plus, nous avons obtenu un nouveau critère pour qu'une preuve de sécurité soit optimale: on dit qu'une preuve de sécurité est optimale si à partir d'une meilleure preuve de sécurité on peut résoudre un problème difficile, comme celui consistant à inverser RSA. Cette technique permet de montrer que le schéma Full Domain Hash, le schéma de Gennaro-Halevi-Rabin, ainsi que le schéma de signature de Paillier ont des preuves de sécurité optimales dans le modèle de l'oracle aléatoire: on ne peut pas faire mieux que de perdre un facteur q_{sig} dans la preuve de sécurité.

[1] $c_i = 0$ avec probabilité γ et $c_i = 1$ avec probabilité $1 - \gamma$

Conclusion

Dans cette thèse, nous avons donc étudié la sécurité de certains schémas de chiffrement et de signature basés sur l'algorithme RSA.

Dans un premier temps, nous avons montré que certains schémas existant et couramment utilisés dans la pratique présentaient certaines faiblesses qui les rendaient vulnérables à certaines attaques. En particulier, nous avons mis en défaut la sécurité du standard de chiffrement PKCS v1.5, ainsi que la sécurité des standards de signature ISO 9796-1 et ISO 9796-2. Nous avons aussi montré comment étendre l'attaque de Girault et Misarsky sur les signatures RSA à redondance affine.

Ensuite, nous avons étudié les moyens de prouver la sécurité des schémas de signature à clef publique. Nous avons donné une preuve de sécurité améliorée du schéma de signature Full-Domain-Hash, qui montre que ce schéma est plus sûr qu'on ne le pensait, ce qui permet de l'utiliser avec des tailles de paramètres plus faibles pour un même niveau de sécurité, et donc d'augmenter son efficacité. Nous avons aussi donné une preuve de sécurité du schéma de signature de Gennaro-Halevi-Rabin, qui permet de prendre en compte la taille de la sortie de la fonction de hachage. Finalement, nous avons aussi montré que lorsque chaque message possède une signature unique, on ne peut pas atteindre le même niveau de sécurité que l'on obtient si chaque message a plusieurs signatures possibles.

Les attaques présentées dans la première partie de cette thèse illustrent le risque pris en utilisant un schéma dont la sécurité n'est pas clairement justifiée. Cela confirme une nouvelle fois l'importance des preuves de sécurité, un domaine de la cryptographie en plein essor actuellement.

Bibliographie

1. FIPS 186. Digital signature standard. Federal Information Processing Standards Publication 186, U.S. Department of Commerce/NIST, 1994.
2. FIPS 46. Data encryption standard. *Federal Processing Standards Publication 46, U.S. Department of Commerce*, 1977.
3. ISO/IEC 9796. Information technology - security techniques - digital signature scheme giving message recovery, part 1 : Mechanisms using redundancy, 1999.
4. ISO/IEC 9796-2. Information technology - security techniques - digital signature scheme giving message recovery, part 2 : Mechanisms using a hash-function, 1997.
5. N. Baric et B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editeur, *Advances in Cryptology - EUROCRYPT '97*, volume 1233, pages 480–494. Springer-Verlag, 1997. Lecture Notes in Computer Science.
6. M. Bellare, A. Desai, D. Pointcheval et P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editeur, *Advances in Cryptology - CRYPTO' 98*, volume 1462, pages 26–45. Springer-Verlag, 1998. Lecture Notes in Computer Science.
7. M. Bellare et P. Rogaway. Random oracles are practical : a paradigm for designing efficient protocols. In *Proceedings of the first annual conference on computer and communication security*, 1993.
8. M. Bellare et P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT '94*, volume 950, pages 92–111. Springer-Verlag, 1995. Lecture Notes in Computer Science.
9. M. Bellare et P. Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. In Ueli Maurer, editeur, *Advances in Cryptology - EUROCRYPT '96*, volume 1070, pages 399–416. Springer-Verlag, 1996. Lecture Notes in Computer Science.
10. E. Biham et A. Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editeur, *Advances in Cryptology - CRYPTO '92*, volume 740, pages 487–496. Springer-Verlag, 1992. Lecture Notes in Computer Science.
11. D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard. In *Advances in Cryptology: Proceedings of CRYPTO '98*, volume 1462, pages 1–12. Springer Verlag, 1998. Lecture Notes in Computer Science.

12. D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society (AMS)*, 46(2):203–213, 1999.

13. D. Boneh et G. Durfee. Cryptanalysis of RSA with private key d less than $n^{0.292}$. In *Advances in Cryptology - EUROCRYPT '99*, volume 1592. Springer-Verlag, 1999. Lecture Notes in Computer Science.

14. D. Boneh, A. Joux et P.Q. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In T. Okamoto, editeur, *Advances in Cryptology - Asiacrypt 2000*, volume 1976, pages 30–43. Springer-Verlag, 2000. Lecture Notes in Computer Science.

15. R. Brent. An imporved Monte-Carlo factorization algorithm. *Nordisk Tidskrift för Informationsbehandling (BIT)*, 20:176–184, 1980.

16. E. Brier, C. Clavier, J.S. Coron et D. Naccache. Cryptanalysis of RSA signatures with fixed-pattern padding. In *CRYPTO 2001*. Springer-Verlag, 2001. Lecture Notes in Computer Science.

17. R. Canetti, O. Goldreich et S. Halevi. The random oracle methodology, revisited. *STOC' 98, ACM*, 1998.

18. C. Clavier, J.S. Coron et N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In *CHES 2000*, volume 1965, pages 252–263. Springer-Verlag, 2000. Lecture Notes in Computer Science.

19. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. of Cryptology*, 10:233–260, 1997.

20. D. Coppersmith, M.K. Franklin, J. Patarin et M.K. Reiter. Low-exponent RSA with related messages. In Ueli Maurer, editeur, *Advances in Cryptology - EUROCRYPT '96*, volume 1070, pages 1–9. Springer-Verlag, 1996. Lecture Notes in Computer Science.

21. D. Coppersmith, S. Halevi et C. Jutla. ISO 9796-1 and the new forgery strategy. Rapport technique, Contribution de recherche au groupe P1363, 1999.

22. D. Coppersmith, A. M. Odlzyko et R. Schroeppel. Discrete logarithms in $GF(p)$. *Algorithmica*, 1:1–15, 1986.

23. J.S. Coron. On the security of random sources. In *Proceedings of PKC '99*, volume 1560, pages 29–42. Springer-Verlag, 1999. Lecture Notes in Computer Science.

24. J.S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Proceedings of CHES '99*, volume 1717, pages 292–302. Springer-Verlag, 1999. Lecture Notes in Computer Science.

25. J.S. Coron. On the exact security of full domain hash. In M. Bellare, editeur, *Prooceedings of CRYPTO 2000*, volume 1880, pages 229–235. Springer Verlag, 2000. Lecture Notes in Computer Science.

26. J.S. Coron et L. Goubin. On boolean and arithmetic masking against differential power analysis. In *CHES 2000*, volume 1965, pages 231–237. Springer-Verlag, 2000. Lecture Notes in Computer Science.

27. J.S. Coron, H. Handschuh et D. Naccache. ECC: do we need to count ? In *Advances in Cryptology - ASIACRYPT '99*, volume 1716, pages 122–134. Springer-Verlag, 1999. Lecture Notes in Computer Science.

28. J.S. Coron, M. Joye, D. Naccache et P. Paillier. New attacks on PKCS#1 v1.5 encryption. In B. Preneel, editeur, *Prooceedings of EUROCRYPT 2000*, volume 1807, pages 369–381. Springer Verlag, 2000. Lecture Notes in Computer Science.

29. J.S. Coron, P. Kocher et D. Naccache. Statistics and secret leakage. In *Financial Cryptography 2000*. Springer-Verlag, 2000. Lecture Notes in Computer Science.

30. J.S. Coron, F. Koeune et D. Naccache. From fixed-lenght to arbitrary-length padding schemes. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976, pages 90–96. Springer-Verlag, 2000. Lecture Notes in Computer Science.

31. J.S. Coron et D. Naccache. An accurate evaluation of maurer's universal test. In *Selected Areas in Cryptography, SAC '98*, volume 1556, pages 57–71. Springer-Verlag, 1998. Lecture Notes in Computer Science.

32. J.S. Coron et D. Naccache. On the security of RSA screening. In *Proceedings of PKC '99*, volume 1560, pages 197–203. Springer-Verlag, 1999. Lecture Notes in Computer Science.

33. J.S. Coron et D. Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In B. Preneel, editeur, *Prooceedings of EUROCRYPT 2000*, volume 1807, pages 91–101. Springer Verlag, 2000. Lecture Notes in Computer Science.

34. J.S. Coron, D. Naccache et J.P. Stern. On the security of RSA padding. In *Advances in Cryptology - CRYPTO '99*, volume 1666, pages 1–18. Springer-Verlag, 1999. Lecture Notes in Computer Science.

35. R. Cramer et I. Damgård. New generation of secure and practical RSA-based signatures. In *Proceedings of CRYPTO'96*, volume 1109, pages 173–185. Springer-Verlag, 1996. Lecture Notes in Computer Science.

36. R. Cramer et V. Shoup. Signature schemes based on the strong RSA assumption. In *Proc. 6th ACM Conf. on Computer and Communications Security*, 1996. Disponible à `http://www.shoup.net/`.

37. W. de Jonge et D. Chaum. Attacks on some RSA signatures. In Hugh C. Williams, editeur, *Advances in Cryptology - CRYPTO '85*, volume 218, pages 18–27. Springer-Verlag, 1986. Lecture Notes in Computer Science.

38. Y. Desmedt et A.M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In Hugh C. Williams, editeur, *Advances in Cryptology - CRYPTO '85*, volume 218, pages 516–522. Springer-Verlag, 1986. Lecture Notes in Computer Science.

39. K. Dickman. On the frequency of numbers containing prime factors of a certain relative magnitude. *Arkiv för matematik, astronomi och fysik*, 22A(10):1–14, 1930.

40. W. Diffie et M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

41. D. Dolev, C. Dwork et M. Naor. Non-malleable cryptography. *In Proc. of the 23rd Symposium on the theory of Computing, ACM*, 1991.

42. C. Dwork et M. Naor. An efficient existentially unforgeable signature scheme and its applications. *J. of Cryptology*, 11(3):187–208, Summer 1998.

43. P. Erdös et C. Pomerance. On a problem of Oppenheim concerning 'factorisatio numerorum'. *J. Number Theory*, 17:1–28, 1983.

44. A. Joux et J. Stern. Lattice reduction : A toolbox for the cryptanalyst. *J. of Cryptology*, 11:161–185, 1998.

45. G. Qiao et K.Y. Lam. RSA signature algorithm for microcontroller implementation. *Proceedings of CARDIS '98*, 1998.

46. S. Cavallar *et al.* Factorization of a 512-bit RSA modulus. In B. Preneel, editeur, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807, pages 1–18. Springer-Verlag, 2000. Lecture Notes in Computer Science.

47. M. Garey et D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman, 1979.

48. R. Gennaro, S. Halevi et T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Proceedings of EUROCRYPT '99*, volume 1592, pages 123–139. Springer-Verlag, 1999. Lecture Notes in Computer Science.

49. J.Y. Girard. *La machine de Turing.* Sources du savoir, éditions du Seuil, 1995.

50. M. Girault et J.-F. Misarsky. Selective forgery of RSA signatures using redundancy. In Walter Fumy, editeur, *Advances in Cryptology - EUROCRYPT '97*, volume 1233, pages 495–507. Springer-Verlag, 1997. Lecture Notes in Computer Science.

51. S. Goldwasser et S. Micali. Probabilistic encryption. *J. of Computer and System Sciences*, 28:270–299, 1984.

52. S. Goldwasser, S. Micali et R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. of computing*, 17(2):281–308, april 1988.

53. S. Goldwasser, S. Micali et P. Tong. Why and how to establish a private code on a public network. *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 134–144, 1982.

54. F. Grieu. A chosen message attacks on the ISO/IEC 9796-1 signature scheme. In B. Preneel, editeur, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807, pages 70–80. Springer-Verlag, 2000. Lecture Notes in Computer Science.

55. G.H. Hardy et E.M. Wright. *An Introduction to the theory of numbers.* Fifth edition, Oxford University Press, 1979.

56. J. Hastad. Solving simultaneous modular equations of low degree. *SIAM J. of Computing*, 17:336–341, 1988.

57. K. Hickman. The SSL protocol, December 1995. Disponible à l'adresse : `http://www.netscape.com/newsref/std/ssl.html`.

58. A. Ivić et G. Tenenbaum. Local densities over integers free of large prime factors. *Quart. J. Math. Oxford (2)*, 37:401–417, 1986.

59. B. Kalisky et M. Robshaw. The secure use of RSA. *CryptoBytes*, 1(3):7–13, 1995.

60. C. Lanczos. An iterative method for the solution of the eigenvalue problem of linear differential and integral operator. *J. Res. Nat. Bur. Standards*, 45:255–282, 1950.

61. A.K. Lenstra, H.W. Lenstra Jr. et L. Lovász. Factoring polynomials with rational coefficients. *Mathematischen Annalen*, 261:515–535, 1982.

62. H. Lenstra. Factoring integers with elliptic curves. *Annals of mathematics*, 126, 1987.

63. M. Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editeur, *Advances in Cryptology - EUROCRYPT '93*, volume 765, pages 386–397. Springer-Verlag, 1993. Lecture Notes in Computer Science.

64. A. Menezes, P. van Oorschot et S. Vanstone. *Handbook of Applied Cryptography.* 1996.

65. Multiprecision integer and rational arithmetic c/c++ library. Disponible à `ftp://ftp.compapp.dcu.ie/pub/crypto/miracl.zip`.

66. J.-F. Misarsky. A multiplicative attack using LLL algorithm on RSA signatures with redundancy. In Burt Kaliski, editeur, *Advances in Cryptology - CRYPTO '97*, volume 1294, pages 221–234. Springer-Verlag, 1997. Lecture Notes in Computer Science.

67. J.-F. Misarsky. How (not) to design RSA signature schemes. In *Public-key cryptography*, volume 1431, pages 14–28. Springer-Verlag, 1998. Lectures notes in computer science.

68. J.-F. Misarsky. *Cryptanalyse et spécification de schémas de signature RSA avec redondance.* PhD thesis, 1999.

69. K. Ohta et T. Okamoto. On concrete security treatment of signatures derived from identification. In *Prooceedings of CRYPTO'98*, volume 1462, pages 354–369. Springer Verlag, 1998. Lecture Notes in Computer Science.

70. T. Okamoto et A. Shiraishi. A fast signature scheme based on quadratic inequalities. *Proc. of the 1985 Symposium on Security and Privacy*, April 1985.

71. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of EUROCRYPT '99*, volume 1592, pages 223–238. Springer-Verlag, 1999. Lecture Notes in Computer Science.

72. D. Pointcheval et J. Stern. Security proofs for signature schemes. In *Advances in Cryptology - EUROCRYPT '96*, volume 1070, pages 387–398. Springer-Verlag, 1996. Lecture Notes in Computer Science.

73. J. Pollard. Factoring with cubic integers. In *The development of the number field sieve*, volume 1554, pages 4–10. Springer-Verlag, 1993. Lectures notes in computer science.

74. C. Pomerance. The quadratic sieve factoring algorithm. In Thomas Beth, Norbert Cot, et Ingemar Ingemarsson, editors, *Advances in Cryptology: Proceedings of EUROCRYPT '84*, volume 209, pages 169–182. Springer-Verlag, 1984. Lecture Notes in Computer Science.

75. A. Shamir R. Rivest et L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21-2:120–126, 1978.

76. M.O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Rapport technique, MIT Laboratory for Computer Science, 1979.

77. PKCS #1: *RSA cryptography specifications*, September 1998. version 2.0.

78. B. Schneier. *Cryptographie appliquée*. International Thomson Publishing France, Paris, 1995.

79. RSA Data Security. PKCS #1: *RSA Encryption Standard*, November 1993. Version 1.5.

80. D. Stinson. *Cryptographie : théorie et pratique*. International Thomson Publishing France, 1996. Traduction par S. Vaudenay de "Cryptography: theory and practice", CRC Press, Inc., 1995.

81. A. Turing. On computable numbers with an application to the entscheidung problem. *Proc. London. Math. Society*, 42(2):230–265, 1936.

82. M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Informations Theory*, 36:553–558, 1990.

83. H. Williams. A modification of the RSA public key encryption procedure. *IEEE TIT*, 26:726–729, 1980.