Analyse cryptographique des altérations d'algorithmes

Alexandre Berzati

Université de Versailles Saint-Quentin — Laboratoire de Recherche en Informatique

Analyse cryptographique des altérations d'algorithmes

THÈSE

présentée et soutenue publiquement le 29 Septembre 2010

pour l'obtention du

Doctorat de l'Université de Versailles Saint-Quentin-en-Yvelines

(spécialité informatique)

par

Alexandre Berzati

Composition du jury

Rapporteurs :	Jean-Claude Bajard	Professeur à l'Université Paris VI
	Jean-Jacques Quisquater	Professeur à l'Université catholique de Louvain
Examinateurs :	Cécile Dumas	*Encadrante*, Ingénieur au CEA-Leti Minatec
	Louis Goubin	*Directeur de Thèse*, Professeur à l'Université de Versailles St-Quentin-en-Yvelines
	Marc Joye	Ingénieur à Technicolor
	Christof Paar	Professeur à la Ruhr-Universität Bochum
	Pascal Paillier	Ingénieur à CryptoExperts
	Frédéric Valette	Ingénieur à la DGA

CEA-Leti Minatec – Centre d'Evaluation de la Sécurité des Technologies de l'Information (CESTI)

energie atomique • energies alternatives

Mis en page avec la classe thloria.

Remerciements

En premier lieu, je souhaite remercier Cécile Dumas et Louis Goubin de m'avoir donner l'opportunité de réaliser cette thèse au CEA Grenoble. En particulier, je souhaite remercier Cécile pour son accompagnement au quotidien, sa gentillesse et qui a toujours su me conseiller pour que je puisse donner le meilleur de moi-même dans mes différents travaux. Je tiens aussi à remercier Louis d'avoir accepté de diriger cette thèse et qui, malgré la distance, m'a toujours soutenu dans mes travaux et m'a fait profiter de sa grande expérience pour étudier la sécurité des systèmes embarqués.

J'aimerais ensuite remercier Jean-Claude Bajard et Jean-Jacques Quisquater d'avoir accepté la lourde tâche de rapporteur. Un grand merci à Christof Paar, Marc Joye, Pascal Paillier et Frédéric Valette d'avoir accepté de faire partie de mon jury de thèse. J'en profite pour également remercier tous mes coauteurs qui ont contribué à faire de cette thèse ce qu'elle est. Je remercie donc chaleureusement Guilhem Castagnos, Blandine Debraize, Cécile Dumas, Jean-Guillaume Dumas, Louis Goubin, Aline Gouget, Pascal Paillier et Stéphanie Salgado pour leur fructueuse collaboration.

Alors que je ne me destinais pas à faire une thèse à la sortie d'école d'ingénieur, certaines personnes m'ont convaincu de me lancer dans cette grande et belle aventure. Je pense notamment à Julien Bringer, Hervé Chabanne et Hervé Pelletier qui, dès mon stage de fin d'étude à SAGEM Sécurité, m'ont donné le goût de la recherche. Je pense également à François VACHERAND qui, lors de notre première entrevue au LETI, a fini de me convaincre.

Mais, il serait réducteur de résumer cette thèse à un projet professionnel. En effet, celle-ci a aussi été une aventure humaine extrêmement enrichissante. J'ai particulièrement apprécié la bonne humeur qui a régné au sein de mon laboratoire de "rattachement" ainsi que dans mon laboratoire "d'adoption". Je tiens à remercier les membres du CESTI pour leur accueil ainsi que pour tous les moments récréatifs qu'ils m'ont offert (poissons d'avril, canulars téléphoniques, discussions footballistiques ou automobilistiques, semaines du goût, blagues en tous genres, etc ...). Merci donc à Stéphanie, Marielle, Didier, Elizabeth, Jean-Yves, Jean-Pierre, Karine, Olivier, Mathilde sans oublier Philippe "un" et Philippe "de(ux)". Un merci tout particulier va à Jessy Clédière dont la contribution à cette thèse va au-delà de la partie récréative. Un grand merci va à mes autres collègues du BOC, avec qui j'ai pris un grand plaisir à travailler. Il s'agit de mes deux "co-bureau" Christine et Lionel, mon acolyte doctorant Pierre-Henri (accessoirement co-fondateur et chef du Projet "Jeudi-P'tit Dèj") ainsi que Rémi, Manuel et Florian. Je remercie également tous ceux qui ont partagé mon quotidien au CEA (ou en conférences) ces trois dernières années et qui ont participé, de près ou de loin, au bon déroulement de cette thèse.

J'aimerais consacrer cette dernière partie à remercier les personnes qui comptent le plus pour moi et sans qui je ne serais certainement pas là aujourd'hui, à savoir ma famille et mes amis. Je souhaite remercier sincèrement ma mère pour l'éducation qu'elle m'a donnée et pour le dévouement dont elle a toujours fait preuve pour la réussite de ses enfants, parfois dans des situations difficiles. C'est pourquoi je souhaite inscrire ici, noir sur blanc, toute mon admiration pour elle et mon immense reconnaissance. Merci aussi à mon petit frère Biz avec qui j'ai sûrement passé les meilleurs moments de ma vie. Une pensée particulière va à mes grands-parents, mes oncles et mes cousins pour leur soutien permanent. Je tiens à remercier également mes deux amis d'en-

fance VINCENT et MAX pour toutes ces soirées PES ou FIFA et pour la sincérité de leur amitié. Un clin d'œil va également à mon *"poto"* TOMATO (le "néo-frelon" rouge) pour m'avoir incité à faire le Master Crypto et m'avoir accordé son amitié.

Enfin, mon dernier remerciement, et sans doute le plus important, va à ma fiancée SÉGOLÈNE, qui a le mérite de me supporter (et ce n'est pas une mince affaire ...) depuis quelques années déjà. Je la remercie pour tout son amour et tout le bonheur que nous partageons dans notre vie quotidienne. J'espère qu'elle embrassera la réussite qu'elle mérite dans les travaux de thèse qu'elle s'apprête à entreprendre, et je ferai tout mon possible pour la combler dans notre vie future.

Je dédie cette thèse à
ma maman, mon frère et
mes grands-parents

Avant-Propos

La cryptographie classique s'efforce de construire des schémas en fournissant, dans la mesure du possible, des preuves relatives de sécurité basées sur la difficulté de certains problèmes mathématiques au sens de la théorie de la complexité. Dans un modèle de sécurité plus étendu, on considère également, depuis quelques années, des attaques qui prennent en compte la nature physique des calculs. Ces nouvelles attaques sont particulièrement menaçantes pour les systèmes embarqués tels que les cartes à microprocesseur, contre lesquels un adversaire peut mobiliser des moyens d'analyse de plus en plus sophistiqués. Parmi ces attaques, les techniques d'injection de fautes sont apparues à la fin des années 90, et n'ont cessé d'évoluer depuis. La popularité de cette classe d'attaque physique réside sur la possibilité de retrouver efficacement, et à moindre coût, les données secrètes pour certaines implantations d'algorithmes cryptographiques.

Cette thèse est le fruit des recherches que j'ai effectuées au sein du laboratoire d'évaluation du CEA-LETI : le CESTI [1]. Bien que le sujet principal de cette thèse concerne la conception et l'implantation d'attaques par perturbations, mes recherches m'ont amené sur des thématiques variées allant de la théorie des nombres aux algorithmes de réduction de réseaux. Cette vaste exploration m'a permis d'acquérir une connaissance générale des concepts liés à la cryptographie moderne. Cette connaissance a été complétée par l'étude, plus particulière, des attaques par perturbation qui m'a apporté une vision plus pragmatique des problématiques liées à l'implantation de fonctions cryptographiques.

Parmi les différentes attaques que nous avons imaginées, j'ai essayé de mettre en avant l'étude que nous avons menée sur les effets relatifs à la perturbation de clés publiques. Cette étude a représenté le fil rouge de la réflexion que j'ai menée durant cette thèse. Cette réflexion autour des attaques par perturbation a été complétée par l'étude des implantations de RSA-CRT ou d'algorithmes de chiffrement à flot. Ces différentes attaques nous ont permis d'identifier les parties critiques des implantations étudiées et, très souvent, a abouti à la proposition de contre-mesures.

Ce mémoire est divisé en quatre parties. Nous commencerons par introduire les différents thèmes abordés à savoir, principalement, la cryptographie et les attaques par injection de fautes. Ensuite, nous présenterons les attaques que nous avons imaginées pour exploiter les perturbations d'éléments publics. La troisième partie adressera les applications des attaques par perturbations contre les implantations d'algorithmes de chiffrement à flot émergents. Enfin, avant de conclure sur les attaques par perturbation, nous mettrons en évidence toute la difficulté d'élaborer des contre-mesures efficaces avec l'exemple du RSA-CRT.

1. Centre d'Évaluation de la Sécurité des Technologies de l'Information.

Glossaire

A5/1	Algorithme de chiffrement à flot utilisé dans le cadre des communications GSM.
CPU	***C**entral **P**rocessor **U**nit* Unité centrale de traitement de l'information qui constitue le cœur calculatoire d'un microprocesseur.
CRT	***C**hinese **R**emainder **T**heorem* Théorème d'arithmétique modulaire traitant de résolution de systèmes de congruences. Ce théorème est notamment utilisé pour rendre le calcul d'une signature RSA, en théorie, quatre fois plus rapide.
DFA	***D**ifferential **F**ault **A**nalysis* Technique d'attaque physique consistant à extraire de l'information à partir des différences observées entre un chiffré de référence et un chiffré obtenu en perturbant le composant cryptographique.
DPA	***D**ifferential **P**ower **A**nalysis* Technique d'attaque physique consistant à mettre en évidence une différence de consommation moyenne, en fonction de la valeur d'un bit d'une donnée manipulée.
DSA	***D**igital **S**ignature **A**lgorithm* Algorithme de signature numérique standardisé par le NIST aux États-Unis, du temps où le RSA était encore breveté. Cet algorithme fait partie de la spécification *DSS* pour *Digital Signature Standard* adoptée en 1993.
GSM	***G**lobal **S**ystem for **M**obile Communications* Norme numérique de deuxième génération pour la téléphonie mobile.
LFSR	***L**inear **F**eedback **S**hift **R**egister* Registre à décalage avec rétroaction linéaire. Les LFSR sont souvent utilisés dans les algorithmes de chiffrement à flot ou dans les générateurs de nombres pseudo-aléatoires.
LLL	**L**enstra **L**enstra **L**ovász LLL est un algorithme de réduction de réseau notamment utilisé pour cryptanalyser des schémas de chiffrement asymétrique.

LSB — ***L**east **S**ignificant **B**it*
En français, bit de poids faible. Bit ayant la plus petite valeur dans la représentation binaire. C'est le bit de droite dans la représentation binaire habituelle.

MSB — ***M**ost **S**ignificant **B**it*
En français, bit de poids fort. Bit ayant la plus grande valeur dans la représentation binaire. C'est le bit de gauche dans la représentation binaire habituelle.

NIST — ***N**ational **I**nstitute of **S**tandards and **T**echnology*
Agence du Département de Commerce des États-Unis remplaçant l'organisme de normalisation anciennement appelé NBS.

NSA — ***N**ational **S**ecurity **A**gency*
Organisme gouvernemental des États-Unis, responsable de la collecte et de l'analyse de toutes formes de communications, aussi bien militaires, gouvernementales, commerciales ou même personnelles, par radiodiffusion, par Internet ou par tout autre mode de transmission.

NVM — ***N**on-**V**olatile **M**emory*
En français, mémoire non-volatile. Type de mémoire qui conservant les données qu'elle contient en l'absence d'alimentation électrique. La ROM est un exemple de mémoire de ce type.

OAEP — ***O**ptimal **A**symmetric **E**ncryption **P**adding*
Cet algorithme fut introduit en 1994 par M. Bellare et P. Rogaway [Mui06]. Utilisé pour le RSA, RSA-OAEP est prouvé sûr dans le modèle de l'oracle aléatoire.

PSS — ***P**robalistic **S**ignature **S**cheme*
Méthode pour créer des signatures RSA proposée par M. Bellare et P. Rogaway [BR96] et prouvée sûre dans le modèle de l'oracle aléatoire.

RAM — ***R**andom **A**ccess **M**emory*
Mémoire rapide et volatile dans laquelle un ordinateur place les données lors de leur traitement.

RC4 — ***R**ivest **C**ipher 4*
Algorithme de chiffrement à flot conçu en 1987 par R. Rivest. Il est supporté par différentes normes, par exemple dans SSL.

RSA — **R**ivest **S**hamir **A**dleman
Du nom de ses inventeurs, le RSA est un crypto-système à clé publique dont la sécurité repose sur la difficulté supposée dz la factorisation de grands entiers.

ROM — ***R**ead-**O**nly **M**emory*
Mémoire qui ne s'efface pas lorsque l'appareil qui la contient n'est plus alimenté en électricité. Une fois programmées, les données stockées en ROM ne sont plus modifiables.

SPA — ***S**imple **P**ower **A**nalysis*
Technique d'attaque physique consistant à extraire de l'information secrète à partir de la mesure et de l'analyse du courant consommé lors de l'exécution d'un calcul cryptographique.

SSL — ***S**ecure **S**ocket **L**ayer*
Protocole de sécurisation des échanges sur Internet, développé à l'origine par Netscape. Ce protocole fournit les fonctions d'authentification, d'intégrité et de confidentialité. OpenSSL est une implantation libre du protocole SSL.

Wi-Fi — Contraction de ***Wi**reless **Fi**delity*
Technologie permettant de relier sans fil plusieurs appareils informatiques au sein d'un réseau informatique. Cette technologie, régie par le groupe de normes IEEE 802.11, est devenue moyen d'accès haut débit à Internet.

Table des matières

Partie V Conclusion & Perspectives

Table des figures

Liste des tableaux

Première partie

La cryptographie appliquée

Chapitre 1

Présentation générale

Sommaire

1.1 Introduction à la cryptologie

Avec l'apparition des premiers langages au début de l'antiquité, les communications sont au coeur de la transmission du savoir et, par la même, de l'essor de l'humanité. Le besoin de

sécuriser ces communications, dans le but de préserver un savoir-faire par exemple, est probablement aussi ancien que les communications elles-mêmes. C'est cette problématique qui a favorisé le développement de la cryptologie, du grec *Kruptos* (caché) et *Logos* (discours), où science du *secret*. Bien que l'on s'accorde à penser que la cryptographie est un art existant depuis l'antiquité [2], c'est seulement avec l'avènement de l'informatique que la cryptographie moderne est considérée comme une science à part entière. Plus particulièrement, l'étude de cette science s'articule autour des trois piliers suivants [Ste97] :

- la *confidentialité* permet d'assurer que l'information n'est seulement accessible qu'à ceux dont l'accès est autorisé,
- l'*authenticité* permet de prouver l'identité d'une personne ou l'origine d'une donnée,
- l'*intégrité* permet de s'assurer que des données n'ont pas été modifiées accidentellement ou volontairement par un tiers.

Classiquement, on divise la cryptologie en deux branches distinctes. La première est la *cryptographie* – où écriture secrète – qui consiste à élaborer des méthodes permettant d'assurer au moins un des trois piliers précédent. La cryptographie peut permettre, par exemple, de promulguer des certificats numériques. Dans ce cas, on souhaite connaître l'auteur du certificat et s'assurer que celui-ci n'a pas été modifié après sa promulgation, soit deux propriétés parmi les trois précités. Les algorithmes de *signature électronique* sont parfaitement adaptés pour assurer ces propriétés.

L'autre branche issue de la cryptologie est la *cryptanalyse*. Cette discipline consiste à analyser la résistance des méthodes cryptographiques face à différentes attaques. Lorsque l'étude de la résistance se limite à l'étude de l'algorithme, on parle d'*attaque mathématique*. Mais, lorsque l'on s'intéresse aux implantations matérielles de ces méthodes cryptographiques, on parle alors d'*attaques physiques*. Dans cette discipline, les mathématiques autant que l'intuition font partie intégrante de la panoplie de l'attaquant.

La cryptographie et la cryptanalyse sont deux domaines complémentaires de la cryptologie qui évoluent en parallèle. En effet, les évolutions des méthodes d'attaques permettent de concevoir des algorithmes cryptographiques plus résistants et, à l'inverse, la résistance de ces algorithmes motive l'imagination d'attaques encore plus tranchantes.

Dans le chapitre suivant, nous présenterons les algorithmes cryptographiques, parmi les plus utilisés de nos jours, que nous allons étudier dans ce mémoire. Nous laisserons le lecteur se référer aux ouvrages [MOVR97, Sti95, Sch96, DRTV07] pour plus de détails sur la cryptologie et ses applications.

1.2 La cryptographie symétrique

Historiquement, le concept de *cryptographie symétrique* est le premier à avoir emergé, permettant notamment d'assurer la sécurité des communications entre deux interlocuteurs. En cryptographie symétrique, une même clé sert à réaliser les opérations de chiffrement et de déchiffrement (*cf.* Figure 1.1). Cette clé doit être tenue secrète pour assurer la sécurité de la communication, c'est pourquoi on désigne aussi ce domaine par *cryptographie à clé secrète*. Le premier algorithme de cette famille, le plus connu, est sans doute le chiffre de César. Son principe consiste à appliquer, à chaque symbole du texte clair, une rotation de valeur fixe dans l'alphabet de référence. Dans ce cas, c'est la valeur de la rotation qui fait office de clé secrète.

De part sa construction, la cryptographie symétrique souffre de deux problèmes majeurs dans son application. Le premier concerne l'échange préalable de la clé secrète entre les deux

2. Les premières traces de cryptologie datent de 2000 ans avant J.-C. en Égypte et le premier texte chiffré est une recette secrète de poterie datant du XVI-ème siècle avant J.-C. découvert dans l'actuel Irak.

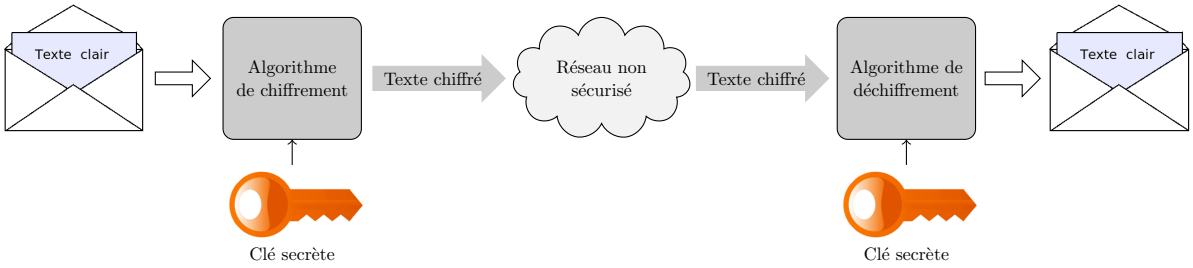


FIGURE 1.1 – Schéma de cryptographie symétrique

interlocuteurs. En effet, pour garantir la sécurité des communications futures, il faut que cette clé ne soit connue par personne d'autre. Cela nécessite de sécuriser un échange préalable de cette clé secrète. L'autre problème relatif à la cryptographie symétrique est la gestion de ces clés. Puisqu'une clé secrète ne permet de sécuriser qu'une seule communication entre deux individus, pour un réseau de n individus, il faudra alors gérer $n(n-1)/2$ clés. En conséquence, le nombre de clés à gérer évolue comme le carré du nombre d'individus sur le réseau. Ces deux problèmes ont très certainement motivé l'invention de la *cryptographie asymétrique* à la fin des années 1970.

Classiquement, on distingue deux types d'algorithmes de chiffrement symétrique : les algorithme de *chiffrement par blocs* et les algorithmes de *chiffrement à flot*. Comme leur nom le suggère, les algorithmes de chiffrement par blocs prennent en entrée des blocs de texte clair de taille fixe (typiquement 64 ou 128 bits) et retournent des chiffrés de même longueur. Cette catégorie d'algorithme de chiffrement symétrique est sans doute la plus utilisée en pratique avec, comme représentants, le *DES*[3] [oSN77] et le standard actuel *AES*[4] [oSN01c]. Les algorithmes de chiffrement à flot, quand à eux, permettent de réaliser un chiffrement bit à bit d'un flot de données en entrée. La clé secrète permet d'initialiser la génération d'une longue séquence de bits, *la suite chiffrante*, à la manière d'un générateur *pseudo-aléatoire*. Cette suite chiffrante est ensuite utilisée pour masquer les bits de texte clair suivant le principe du chiffrement à masque jetable de VERNAM.

Les algorithmes de chiffrement à flot sont plus rapides que ceux chiffrant par blocs mais étaient généralement considérés comme beaucoup plus faibles d'un point de vue cryptographique. À ASIACRYPT [Sha04], A. Shamir donnait, pour cette raison, un avis assez pessimiste sur l'avenir des algorithmes de chiffrement à flot, prévoyant même leur disparition à long terme. Quelques temps après cette intervention, le réseau européen ECRYPT monta le projet *eSTREAM* dans le but de promouvoir de nouveaux algorithmes de chiffrement à flot visant des applications matérielles et logicielles [5].

Dans la suite, nous présenterons trois algorithmes de chiffrement à flot, dont deux sélectionnés comme finalistes à l'issue de ce projet : RABBIT et GRAIN-128. Nous nous attarderons spécialement sur ces deux derniers algorithmes car nous avons étudié la sécurité de leurs implantations d'un point de vue des perturbations, dans le cadre du projet ANR Odyssée.

3. *Data Encryption Standard*
4. *Advanced Encryption Standard*
5. Portfolio des finalistes disponible à `http://www.ecrypt.eu.org/stream/index.html`

1.2.1 Présentation de RC4

L'algorithme de chiffrement à flot *RC4* (pour *Rivest Cipher 4*) est apparu en 1987. Son concepteur n'est autre que Ronald Rivest, l'un des 3 inventeurs de l'algorithme RSA (*cf.* Section 1.3.1). Cet algorithme se distingue par sa grande simplicité et sa vitesse de chiffrement. Les détails de l'algorithme ont été tenus secrets jusqu'à ce qu'une description, probablement obtenue par ingénierie inverse, ne soit posté sur la liste de diffusion *Cipherpunks* en 1994. Malgré cela, il a été utilisé dans des protocoles comme *WEP*[6], *WPA*[7] ou *TLS*[8]. D'autre part, cet algorithme a fait l'objet de nombreuses recherches qui ont révélé différentes vulnérabilités cryptographiques. S. Fluhrer et D. McGrew ont d'abord montré l'existence d'un biais dans la suite chiffrante générée par RC4. Ce biais leur permet de distinguer le flux généré par RC4 d'un flux aléatoire en analysant $2^{30.6}$ octets de suite chiffrante [FM01]. Ce résultat a ensuite été amélioré à 2^{25} octets par B. Preneel et S. Paul en 2004 [PP04]. Une autre attaque utilisant le biais de RC4, mais engendré cette fois par les tout premiers octets de suite chiffrante générés, fut présenté en 2001 par S. Fluhrer, I. Mantin et A. Shamir [FMS01]. C'est sur ce principe qu'ils ont proposé d'attaquer le protocole WEP dans les réseaux sans fils. Pour ces différentes raisons, l'algorithme RC4 n'offre plus un niveau de sécurité suffisant pour de futures applications.

Description générale. RC4 est algorithme de chiffrement à flot synchrone prenant en entrée une clé secrète pouvant varier de 40 à 1024 bits. En pratique, on choisit souvent une taille de clé de 128 bits. En revanche, cet algorithme ne prend pas de vecteur d'initialisation en entrée. L'état interne se compose de 258 octets répartis de la manière suivante :

- une permutation S de toutes les 256 valeurs possibles d'un octet,
- deux pointeurs i et j servant d'index dans le tableau de permutation.

Après la phase d'initialisation de la clé, l'algorithme génère un octet de suite chiffrante par itération de la mise à jour d'état interne. Les fonctions d'initialisation et de mise à jour d'état interne sont décrites dans les paragraphes suivants.

1.2.1.1 Initialisation

Avant de générer les premiers octets de suite chiffrante, l'algorithme exécute une phase d'initialisation. Cette étape consiste à utiliser la clé secrète comme une *graine* pour la génération d'une suite chiffrante aléatoire. Dans un premier temps, tous les octets de la permutation S sont initialisés avec leur indices respectifs. Ainsi, $S[0] = 0, S[1] = 1, \ldots, S[255] = 255$. Une fois ces opérations effectuées, l'algorithme utilise la clé secrète pour calculer des indices qui serviront à mélanger la permutation S de manière "aléatoire". Le principe de l'initialisation est décrit dans l'algorithme 1. De cette manière, l'ordre du mélange réalisé dans de la permutation S dépend de la clé et l'on est sûr que chacun des octets de S a bien été permuté au moins une fois. Une fois cette initialisation terminée, l'algorithme est prêt pour la génération de la suite chiffrante proprement dite.

1.2.1.2 Génération de la suite chiffrante

L'algorithme RC4 génère un octet de suite chiffrante par itération de la fonction de mise à jour de l'état interne. Contrairement à la phase d'initialisation, la clé n'est plus utilisée pour

6. Wired Equivalent Privacy
7. Wi-Fi Protected Access
8. Transport Layer Security

Algorithme 1 Initialisation de RC4 par la clé secrète

ENTRÉES : État interne S, clé secrète K, taille de clé n

Pour $i = 0$ **à** 255 **faire**
  $S[i] \leftarrow i$
Fin Pour
$j \leftarrow 0$
Pour $i = 0$ **à** 255 **faire**
  $j \leftarrow (j + S[i] + K[i \bmod n]) \bmod 256$
  Temp $\leftarrow S[i]$
  $S[i] \leftarrow S[j]$
  $S[j] \leftarrow$ Temp
Fin Pour

mettre à jour les registres d'état interne. Au début de l'exécution, les pointeurs de l'état interne i et j sont remis à zéro, puis leur valeur est incrémentée de la façon suivante :

$$\begin{aligned} i &\leftarrow i + 1 \bmod 256 \\ j &\leftarrow j + S[i] \bmod 256 \end{aligned}$$

Ces pointeurs servent ensuite à réaliser la mise à jour de l'état de la permutation S. En effet, ils définissent les octets de S qui seront additionnés modulo 256 puis permutés. Le résultat de cette addition définit l'octet de la permutation S qui sera retourné en guise d'octet de la suite chiffrante (*cf.* Figure 1.2). Ainsi, pour chaque itération de la mise à jour de l'état interne, un

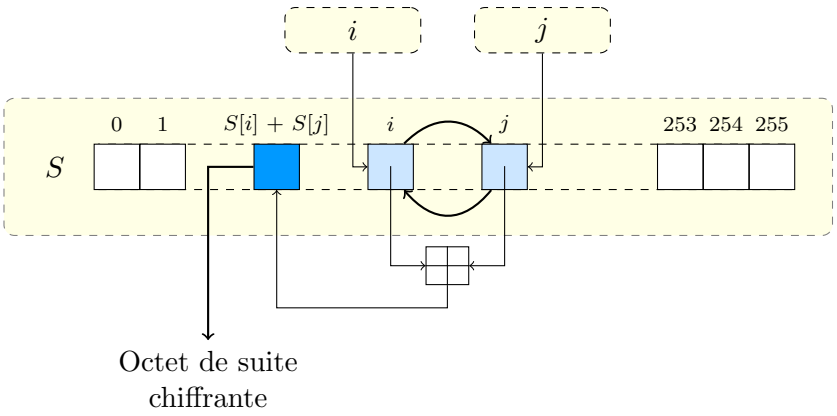


FIGURE 1.2 – Schéma de mise à jour de l'état interne de RC4

octet de suite chiffrante est généré. Cet octet est ensuite combiné avec un octet de message via un "ou exclusif" pour donner un octet du chiffré. On peut toutefois noter que pour éviter l'exploitation d'un biais par la méthode de S. Fluhrer, I. Mantin et A. Shamir [FMS01], il est préférable de ne pas retourner les 512 premiers octets de suite chiffrante.

1.2.2 Présentation de RABBIT

L'algorithme de chiffrement à flot RABBIT a été présenté pour la première fois à la conférence FSE en 2003 [BVP+03]. Cet algorithme a été ensuite proposé comme candidat au projet

eSTREAM en 2005 [BVCZ05] organisé par le réseau d'excellence européen ECRYPT. Ses performances, notamment en termes de vitesse de chiffrement [9] lui permettent d'apparaître parmi les quatre algorithmes de chiffrement à flot retenus à l'issue du projet, dans la catégorie logicielle. Bien que l'algorithme RABBIT ait été imaginé dans une optique de performances, les aspects de sécurité n'ont pas été laissés de côté par les concepteurs. En effet, ces derniers n'ont pas hésité à mener une étude approfondie pour démontrer la résistance de RABBIT face aux attaques classiques (*i.e.* algébrique, corrélation, *guess-and-determine*, différentielle). L'étude de la sécurité de cet algorithme de chiffrement à flot, par ses concepteurs, apparaît dans [BVCZ05] et a fait l'objet d'une série de rapports [A/S03a, A/S03b, A/S03c, A/S03d, A/S03e, A/S03f]. Par ailleurs, la fonction de mise à jour de l'état interne a aussi été étudié dans deux contributions externes [Aum07, YHL08]. Ces études ont mis en évidence un biais dans cette fonction, mais celui-ci n'est pas exploitable en pratique [10]. Aussi, c'est pour ces raisons que RABBIT est considéré, à ce jour, comme un algorithme de chiffrement à flot sûr et efficace.

Description générale. RABBIT est un algorithme de chiffrement à flot synchrone prenant en entrée une clé secrète de 128 bits ainsi qu'un vecteur d'initialisation (IV) de 64 bits. Bien qu'aléatoire ce vecteur d'initialisation n'est pas considéré comme une donnée secrète. À chaque itération, l'algorithme produit en sortie un bloc *pseudo-aléatoire* de 128 bits. Dans ce cas, l'opération de chiffrement (resp. déchiffrement) consiste à combiner ce bloc de suite chiffrante à un bloc de 128 bits de texte clair (resp. texte chiffré) par un "ou exclusif" (XOR).À partir d'une clé secrète de 128 bits, il est possible de chiffrer jusqu'à 2^{64} octets de texte clair.

L'état interne de cet algorithme de chiffrement à flot est composé de 513 bits répartis de la manière suivante :

- Huit registres d'état interne de taille 32 bits notés $(x_{j,i})_{0\leq j\leq 7}$ à l'itération i,
- Huit compteurs de taille 32 bits notés $(c_{j,i})_{0\leq j\leq 7}$,
- Un bit de retenue noté $\phi_{7,i}$.

À l'instant $i = 0$, les registres d'état interne et les compteurs sont initialisés suivant le schéma d'initialisation de clé et d'IV. Cette initialisation sert à rendre les bits d'état interne le moins dépendant possible des bits des données d'entrée, et en particulier des bits de clé sécrète. La génération des blocs de suite chiffrante commence seulement une fois l'initialisation achevée.

Notations. Afin de décrire le fonctionnement de RABBIT, nous utiliserons les même notations que celles utilisées lors de la description original de l'algorithme à FSE 2003 [BVP+03]. Ces notations seront aussi utilisées, plus loin dans cette thèse, pour décrire notre attaque par perturbation.

- La fonction logique `XOR` est notée $\oplus$,
- la fonction logique `AND` est notée $\wedge$,
- la fonction logique `OR` est notée $\vee$,
- la concaténation de deux variables est notée $\diamond$,
- les décalages de n bits vers la gauche et vers la droite sont notés respectivement $\ll n$ et $\gg n$,

9. Selon [BVCZ05], la vitesse de chiffrement/déchiffrement d'implantations matérielles optimisées de RABBIT peut atteindre $3,7$ bits par cycle d'horloge sur un processeur Pentium III, $9,6$ bits par cycle d'horloge sur un ARM7 et $10,9$ bits par cycle d'horloge sur un MIPS 4Kc.

10. L'article [YHL08] présente une amélioration de l'attaque de J.-P. Aumasson qui proposait initialement d'exploiter le biais de la fonction de mise à jour de RABBIT en collectant 2^{247} blocs de suite chiffrante [Aum07]. Grâce à l'utilisation de la transformée de Fourier rapide (*FFT*), cette attaque permet réduire le nombre nécessaire pour l'extraction de la clé secrète à 2^{158} blocs. Une variante plus efficace est aussi proposée dans [YHL08].

- les rotations de n bits vers la gauche et vers la droite sont notées respectivement $\lll n$ et $\ggg n$,
- la partie du vecteur A allant du bit g au bit h est noté $A^{[g..h]}$,
- A modulo k est noté $A_{[k]}$.

1.2.2.1 Initialisation

Initialisation par la clé secrète. L'état interne de l'algorithme est tout d'abord initialisé par les 128 bits de clé secrète. Cette clé $K^{[127..0]}$ est divisée en huit blocs de 16 bits : $k_0 = K^{[15..0]}$, $k_1 = K^{[31..16]}$, ..., $k_7 = K^{[127..112]}$. Puis, l'état interne de RABBIT est initialisé, à partir de ces sous-clés, de la manière suivante :

$$x_{j,0} = \begin{cases} k_{(j+1 \bmod 8)} \diamond k_j & \text{si } j \text{ est pair} \\ k_{(j+5 \bmod 8)} \diamond k_{(j+4 \bmod 8)} & \text{si } j \text{ est impair} \end{cases} \tag{1.1}$$

et

$$c_{j,0} = \begin{cases} k_{(j+4 \bmod 8)} \diamond k_{(j+5 \bmod 8)} & \text{si } j \text{ est pair} \\ k_j \diamond k_{(j+1 \bmod 8)} & \text{si } j \text{ est impair} \end{cases} \tag{1.2}$$

Cette initialisation est suivie de quatre itérations consécutives de la fonction de mise à jour de l'état interne décrite plus loin. Ensuite, tous les compteurs sont réinitialisés en utilisant les registres d'état de manière à rendre impossible une extraction de clé par inversion de la mise à jour des compteurs :

$$\forall j \in [\![0;7]\!],\ c_{j,4} = c_{j,4} \oplus x_{(j+4 \bmod 8),4} \tag{1.3}$$

Initialisation par l'IV. Contrairement au schéma d'initialisation par la clé secrète, l'initialisation par l'IV concerne exclusivement les compteurs d'état interne. Le principe consiste à mélanger les bits de l'IV aux bits de compteur puis d'exécuter quatre mises à jour successives de manière à retirer toute dépendance linéaire entre les bits de compteurs et les bits d'IV :

$$\begin{aligned} c_{0,4} &= c_{0,4} \oplus IV^{[31..0]} & \qquad c_{1,4} &= c_{1,4} \oplus (IV^{[63..48]} \diamond IV^{[31..16]}) \\ c_{2,4} &= c_{2,4} \oplus IV^{[63..32]} & \qquad c_{3,4} &= c_{3,4} \oplus (IV^{[47..32]} \diamond IV^{[15..0]}) \\ c_{4,4} &= c_{4,4} \oplus IV^{[31..0]} & \qquad c_{5,4} &= c_{5,4} \oplus (IV^{[63..48]} \diamond IV^{[31..16]}) \\ c_{6,4} &= c_{6,4} \oplus IV^{[63..32]} & \qquad c_{7,4} &= c_{7,4} \oplus (IV^{[47..32]} \diamond IV^{[15..0]}) \end{aligned}$$

Cette initialisation des compteurs d'état par l'IV garantie que pour chacun des 2^{64} IVs possibles, une suite chiffrante unique sera générée.

1.2.2.2 Génération de la suite chiffrante

La fonction principale de RABBIT est la mise à jour de l'état interne. C'est cette fonction qui assure la non-linéarité des relations entre les différents registres de l'état interne. Leur mise

à jour est définie par les équations suivantes :

$$\begin{aligned}
x_{0,i+1} &= g_{0,i} + (g_{7,i} \lll 16) + (g_{6,i} \lll 16) \\
x_{1,i+1} &= g_{1,i} + (g_{0,i} \lll 8) + g_{7,i} \\
x_{2,i+1} &= g_{2,i} + (g_{1,i} \lll 16) + (g_{0,i} \lll 16) \\
x_{3,i+1} &= g_{3,i} + (g_{2,i} \lll 8) + g_{1,i} \\
x_{4,i+1} &= g_{4,i} + (g_{3,i} \lll 16) + (g_{2,i} \lll 16) \\
x_{5,i+1} &= g_{5,i} + (g_{4,i} \lll 8) + g_{3,i} \\
x_{6,i+1} &= g_{6,i} + (g_{5,i} \lll 16) + (g_{4,i} \lll 16) \\
x_{7,i+1} &= g_{7,i} + (g_{6,i} \lll 8) + g_{5,i}
\end{aligned}$$

dont les valeurs $g_{j,i}$ sont définies par l'expression suivante :

$$g_{j,i} = (x_{j,i} + c_{j,i+1})^2 \oplus \Big((x_{j,i} + c_{j,i+1})^2 \gg 32\Big) \bmod 2^{32} \tag{1.4}$$

Les additions sont réalisées modulo 2^{32} et les élévations au carré modulo 2^{64}. Dans la littérature, cette expression est aussi formalisée en une fonction (*i.e.* fonction *"g"*). Comme souligné précédemment, l'étude statistique de cette fonction a révélé la présence d'un biais [Aum07, YHL08]. La figure 1.3 schématise la mise à jour des registres d'état interne de Rabbit. La mise

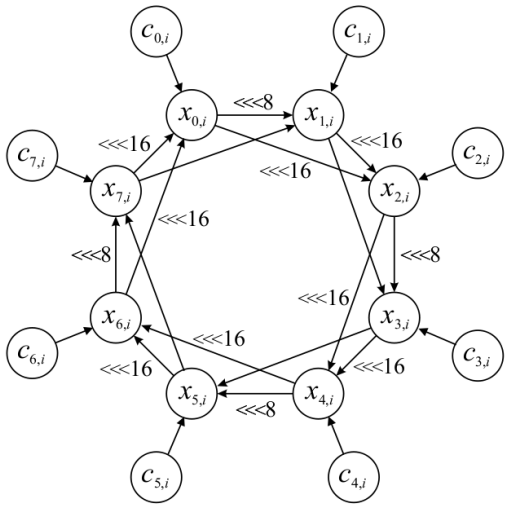


Figure 1.3 – Schéma de la mise à jour de l'état interne (tiré de [BVP+03])

à jour de l'état interne passe aussi par la mise à jour des huit compteurs d'état ainsi que par celle du bit de retenue $\phi_{7,i}$. Les équations suivantes détaillent la mise à jour de ces dernières variables d'état :

$$\begin{aligned}
c_{0,i+1} &= c_{0,i} + a_0 + \phi_{7,i} \bmod 2^{32}, \\
c_{j,i+1} &= c_{j,i} + a_j + \phi_{j-1,i+1} \bmod 2^{32}, \text{ pour } 0 < j < 8.
\end{aligned} \tag{1.5}$$

Le bit de retenue $\phi_{j,i+1}$ est obtenu comme suit :

$$\phi_{j,i+1} = \begin{cases} 1 & \text{si } c_{0,i} + a_0 + \phi_{7,i} \geq 2^{32} \wedge j = 0, \\ 1 & \text{si } c_{j,i} + a_j + \phi_{j-1,i+1} \geq 2^{32} \wedge j > 0, \\ 0 & \text{autrement} \end{cases}$$

Enfin, les constantes $(a_j)_{0 \leq j \leq 7}$ utilisées pour la mise à jour des compteurs valent :

$$\begin{cases} a_0 = & a_3 = a_6 = \texttt{0x4D34D34D}, \\ a_1 = & a_4 = a_7 = \texttt{0xD34D34D3}, \\ a_2 = & a_5 = \texttt{0x34D34D34} \end{cases}$$

Enfin, pour chaque itération i de la mise à jour de l'état interne, 128 bits de suite chiffrante ${s_i}^{[127..0]}$ sont générés à partir des registres d'état interne :

$$\begin{aligned} {s_i}^{[15..0]} &= {x_{0,i}}^{[15..0]} \oplus {x_{5,i}}^{[31..16]} & {s_i}^{[31..16]} &= {x_{0,i}}^{[31..16]} \oplus {x_{3,i}}^{[15..0]} \\ {s_i}^{[47..32]} &= {x_{2,i}}^{[15..0]} \oplus {x_{7,i}}^{[31..16]} & {s_i}^{[63..48]} &= {x_{2,i}}^{[31..16]} \oplus {x_{5,i}}^{[15..0]} \\ {s_i}^{[79..64]} &= {x_{4,i}}^{[15..0]} \oplus {x_{1,i}}^{[31..16]} & {s_i}^{[95..80]} &= {x_{4,i}}^{[31..16]} \oplus {x_{7,i}}^{[15..0]} \\ {s_i}^{[111..96]} &= {x_{6,i}}^{[15..0]} \oplus {x_{3,i}}^{[31..16]} & {s_i}^{[127..112]} &= {x_{6,i}}^{[31..16]} \oplus {x_{1,i}}^{[15..0]} \end{aligned}$$

1.2.3 Présentation de Grain-128

Un autre algorithme sélectionné comme finaliste du projet *eSTREAM*, mais dans la catégorie matérielle est l'algorithme Grainv1. Bien qu'il présente des performances intéressantes, la taille de la clé secrète est de 80 bits ce qui n'offre pas un niveau de sécurité suffisant [BS00]. L'algorithme de chiffrement à flot Grain-128 est une déclinaison de Grainv1 utilisant une clé secrète de 128 bits [HJMM06] tout en conservant l'efficacité du modèle original. Sa structure interne est composée de deux registres à décalage. Le premier est un registre à décalage avec rétroaction linéaire (LFSR) permettant de garantir une période minimale pour la suite chiffrante ainsi que de bonnes propriétés statistiques. Afin d'éviter une dépendance linéaire des bits de suite chiffrante avec les bits d'état interne, on combine le registre précédent avec un registre à décalage avec rétroaction non-linéaire (NFSR) ainsi qu'à une fonction de filtrage non-linéaire.

Cet algorithme étant destiné à être implanté en matériel, sa résistance face aux attaques physiques classiques a déjà été étudiée [AMMA08]. Nous présenterons, plus loin dans la thèse, une analyse des perturbations [BCC+09] développée en collaboration avec les membres du projet ANR Odyssée.

Description générale. Grain-128 est un algorithme de chiffrement à flot synchrone prenant en entrée une clé secrète de 128 bits ainsi qu'un vecteur d'initialisation de 96 bits. L'état interne de 256 bits est divisé en deux registres à décalage de 128 bits, l'un linéaire et l'autre non-linéaire. La suite chiffrante est générée en filtrant certains bits de l'état interne via une fonction de filtrage non-linéaire h. En régime établi (*i.e.* après la phase d'initialisation), le débit en sortie est d'un bit de suite chiffrante par coup d'horloge. Mais, en remarquant que les 31 derniers bits de chacun des deux registres ne sont pas utilisés pour générer la suite chiffrante et si le matériel le permet, il est possible de multiplier par 32 le débit en sortie en parallélisant l'exécution.

Afin de décrire le fonctionnement de l'algorithme, nous reprendrons les notations initialement définies par les concepteurs.À l'instant i, les bits de l'état interne contenus dans le LFSR seront notés $s_i, s_{i+1}, \ldots, s_{i+127}$ et ceux du NFSR $b_i, b_{i+1}, \ldots, b_{i+127}$.

1.2.3.1 Initialisation

La première phase consiste à initialiser les registres d'état interne avec les bits de clé secrète et de vecteur d'initialisation. Tous les 128 bits de la clé k sont copiés dans le NFSR : $\forall i, 0 \leq i \leq 127, b_i = k_i$. Ensuite, les 96 bits de l'IV sont copiés dans le LFSR tel que $\forall i, 0 \leq i \leq 95, s_i = IV_i$ et le reste des bits du LFSR sont initialisés à 1 : $\forall i, 96 \leq i \leq 127, s_i = 1$. L'initialisation se termine par 256 itérations de la mise à jour de l'état interne sans générer de bits de suite chiffrante (*cf.* Figure 1.4). La sortie étant `XOR`ée avec le résultat de la fonction de rétroaction de chacun des deux registres.

1.2.3.2 Génération de la suite chiffrante

Une fois la phase d'initialisation achevée, l'algorithme peut générer la suite chiffrante qui sera utilisée pour chiffrer. La figure 1.4 décrit la mise à jour de l'état interne de Grain-128 aboutissant à la génération d'un bit de suite chiffrante par itération. Le LFSR est mis à jour grâce à la

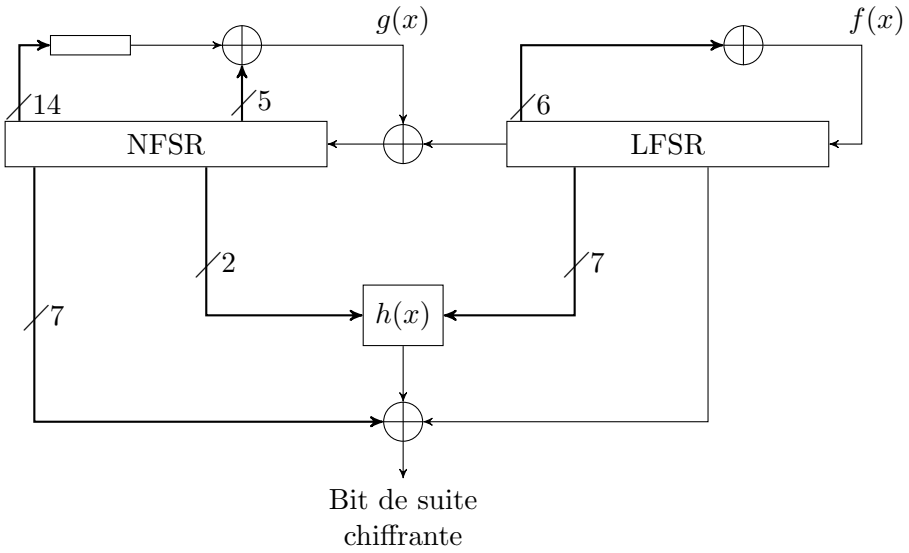


Figure 1.4 – Schéma de mise à jour de l'état interne de Grain-128

fonction de rétroaction linéaire f prenant en entrée 6 bits de l'état interne. Cette fonction est définie par le polynôme suivant :

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128} \tag{1.6}$$

où le + représente l'addition binaire (*i.e.* "ou exclusif"). Ainsi, les bits du LFSR sont mis à jour de la façon suivante :

$$s_{i+128} = s_i \oplus s_{i+7} \oplus s_{i+38} \oplus s_{i+70} \oplus s_{i+81} \oplus s_{i+96} \tag{1.7}$$

En revanche, la fonction de rétroaction du NFSR, g, est non-linéaire, car elle utilise l'opérateur binaire "et" noté "." :

$$\begin{aligned} g(x) &= 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} \\ &\quad + x^{44}.x^{60} + x^{61}.x^{125} + x^{63}.x^{67} \\ &\quad + x^{69}.x^{101} + x^{80}.x^{88} + x^{110}.x^{111} \\ &\quad + x^{115}.x^{117} \end{aligned} \tag{1.8}$$

La mise à jour du NFSR se réalise à partir de la fonction linéaire précédente dont la sortie est masquée par un bit du LFSR s_i :

$$\begin{aligned} b_{i+128} &= s_i \oplus b_i \oplus b_{i+26} \oplus b_{i+56} \oplus b_{i+91} \oplus s_{i+96} \\ &\quad \oplus b_{i+3}.b_{i+67} \oplus b_{i+11}.b_{i+13} \oplus b_{i+17}.b_{i+18} \\ &\quad \oplus b_{i+27}.b_{i+59} \oplus b_{i+40}.b_{i+48} \oplus b_{i+61}.b_{i+65} \\ &\quad \oplus b_{i+68}.b_{i+84} \end{aligned} \tag{1.9}$$

Enfin, pour chaque mise à jour des 256 bits d'état interne, un bit de suite chiffrante est généré. Cet bit est obtenu en filtrant simultanément 2 bits de NFSR et 7 bits de LFSR par la fonction non-linéaire h suivante :

$$h(x) = x_0.x_1 + x_2.x_3 + x_4.x_5 + x_6.x_7 + x_0.x_4.x_8 \tag{1.10}$$

Les bits $x_0, x_1, \ldots, x_8$ correspondent respectivement aux bits d'état interne b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79} et s_{i+95}. Enfin la sortie de la fonction h est différenciée avec 7 bits du NFSR et 1 bit du LFSR pour donner un bit de suite chiffrante z_i :

$$\begin{aligned} z_i &= b_{i+2} \oplus b_{i+15} \oplus b_{i+36} \oplus b_{i+45} \oplus b_{i+64} \oplus b_{i+73} \oplus b_{i+89} \\ &\quad \oplus h(x) \oplus s_{i+93} \end{aligned} \tag{1.11}$$

1.3 La cryptographie asymétrique

Le concept de *cryptographie asymétrique*, ou *cryptographie à clé publique* a été introduit en 1976 par Whitfield Diffie et Martin Hellman dans l'article fondateur [DH76a].À l'époque, les auteurs ont présenté le concept sans toutefois en proposer d'instance concrète. Il a fallu attendre l'invention du RSA [RSA78] et du protocole d'échange de clé Diffie-Hellman [DH76b] pour voir les premières applications de cryptographie asymétrique.

Contrairement à la cryptographie symétrique, ce nouveau système repose sur l'utilisation de deux clés distinctes servant à réaliser respectivement les opérations de chiffrement et de déchiffrement. Classiquement, le destinataire choisit la clé de déchiffrement, la *clé privée*, dont il dissimule la valeur. En utilisant une fonction *à trappe* et la clé privée, il calcule ensuite la clé qui sera utilisée pour le chiffrement : la *clé publique*. L'utilisation de fonctions à trappe est particulièrement intéressante ici car ces fonctions sont faciles à calculer mais difficiles à inverser si l'on ne connaît pas *la trappe*. Aussi, en cryptographie asymétrique, c'est la clé privée qui fait office de trappe. La clé publique est ensuite diffusée ce qui permet aux autres individus d'envoyer des messages chiffrés que seul le destinataire légitime (*i.e.* le détenteur de la clé privée associée) pourra déchiffrer (*cf.* Figure 1.5). Ainsi, dans le cadre de la cryptographie asymétrique, il n'est pas nécessaire d'échanger préalablement un secret pour communiquer de manière sécurisée.

Toutefois, l'utilisation de clés publiques peut poser des problèmes. En effet, en l'état, il est difficile d'identifier réellement la personne qui diffuse sa clé publique. Il se pourrait qu'un attaquant diffuse une clé publique en usurpant l'identité d'un individu. Il serait alors capable de déchiffrer les messages adressés à la victime. Ce problème peut être réglé par le déploiement d'infrastructures de gestion de clés publiques. [11]

En pratique, les méthodes de chiffrement asymétrique sont nettement plus lentes que celles

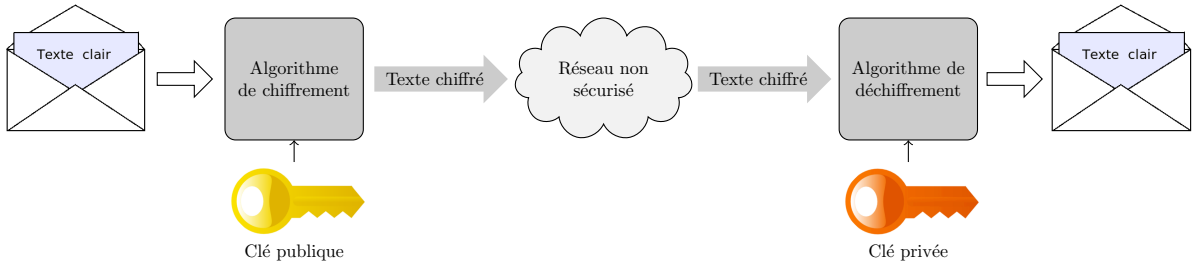


FIGURE 1.5 – Schéma de cryptographie asymétrique

de chiffrement symétrique. Cela s'explique notamment par la complexité des opérations mathématiques qu'elles doivent réaliser ainsi que par la longueur des clés recommandées pour garantir un niveau de sécurité acceptable (par exemple, la recommandation minimale de l'ANSSI [12] pour la taille des clés RSA est de 2048 bits [ANS10]). Par conséquent, les méthodes de chiffrement asymétrique ne sont pas intéressantes pour sécuriser de canaux de communications sur lesquels transitent un important volume de données. En revanche, elle sont très pratiques pour sécuriser l'échange d'une clé de chiffrement symétrique qui permettra ensuite d'utiliser un algorithme de chiffrement par blocs, plus rapide, pour échanger des données.

L'apparition de la cryptographie asymétrique a permis l'émergence du concept de *signature électronique*. Comme pour une signature manuscrite apposée au bas d'un document, sa version électronique peut garantir les deux propriétés cryptographiques que sont l'authentification (identifier l'individu qui a signé le document) et l'intégrité (vérifier que le document n'a pas été modifié après sa signature). La figure 1.6 décrit un schéma général de signature numérique. L'utilisation de la clé privée pour le calcul de la signature permet aux autres individus, connais-

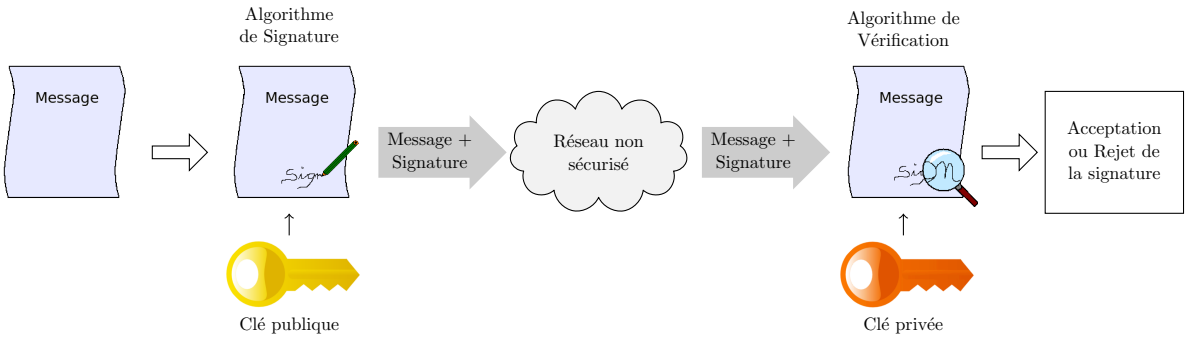


FIGURE 1.6 – Schéma de signature électronique

11. *Public Key Infrastructure* dans la littérature.
12. Agence Nationale de la Sécurité des Systèmes d'Information

sant la clé publique associée et le document original, de vérifier facilement sa validité. La validité de la signature prouve alors que c'est bien le possesseur de la clé privée qui a signé le document et que le document n'a pas été modifié après sa signature. Ainsi, la signature électronique présente bien les propriétés cryptographiques attendues. Dans la partie suivante nous présenterons trois des algorithmes les plus utilisés en cryptographie asymétrique à savoir les algorithmes *RSA*, El Gamal et *DSA*.

1.3.1 Présentation de RSA

L'algorithme *RSA* a été inventé en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman [RSA78]. C'est la première instance de méthodes de cryptographie à clé publique, dont le principe a été introduit à l'état de concept, par Whitfield Diffie et Martin Hellman [DH76a].À l'heure actuelle, il s'agit sans doute du crypto-système asymétrique le plus connu et le plus utilisé à travers le monde. Sa popularité en fait aussi un des algorithmes les plus étudiés, que ce soit d'un point de vue théorique [HS09, Cop96, May03, JM06] ou pratique [Koc96, FKJM$^+$06, Len96].

La sécurité du RSA repose sur la difficulté supposée du problème de la factorisation d'entiers.À ce jour, il n'existe pas de méthodes de résolution dont la complexité est meilleure que sous-exponentielle (*i.e.* Méthode du crible algébrique ou *Number Field Sieve* [LHL93]). Le plus grand module RSA factorisé par ce moyen, en utilisant une méthode de calculs distribués, est long de 768 bits [13]. Afin de rendre impossible la factorisation du module, on choisit en pratique des tailles de module au moins égales à 1024 bits, voire à 2048 bits.

En pratique, le RSA se décline en deux versions. La version classique, ou version RSA *standard*, décrite ci-après, et la version *CRT* [14]. Cette dernière version est quatre fois plus rapide que la version standard, c'est pourquoi elle est très utilisée pour implanter le RSA dans les systèmes embarqués. Cette déclinaison est décrite plus loin dans la thèse (*cf.* Section 11.1).

1.3.1.1 Génération des clés

Le module public N (ou module RSA), est défini comme le produit de deux grands nombres premiers tirés aléatoirement p et q. La taille de N, au sens binaire, sera noté n. On a alors, $\varphi(N) = (p-1)\cdot(q-1)$, l'image du module RSA par l'indicatrice d'EULER $\varphi(\cdot)$. On choisit alors l'*exposant public* e de telle sorte que le $pgcd(e, \varphi(N)) = 1$. Le couple $K_p = (N, e)$ forme la partie publique de la clé. La partie privée K_s est l'*exposant privé* d, calculé comme l'inverse de e dans le groupe multiplicatif $(\mathbb{Z}/N\mathbb{Z})^*$, soit :

$$d \cdot e \equiv 1 \bmod \varphi(N) \tag{1.12}$$

1.3.1.2 Chiffrement RSA

Le chiffrement RSA d'un message m consiste en une simple exponentiation modulaire avec l'exposant public : $C = m^e \bmod N$. L'opération de déchiffrement est réalisée de manière similaire mais, en utilisant l'exposant privé d : $\tilde{m} = C^d \bmod N$. Si aucune erreur n'intervient pendant le chiffrement, la transmission du chiffré ou le déchiffrement, alors $\tilde{m} = m$. Une méthode alternative, basée sur le Théorème des restes chinois, a été proposée par J.-J. Quisquater et C. Couvreur

13. Le 12 décembre 2009, une équipe internationale de chercheurs annonce la factorisation d'un module RSA de 768 bits grâce à l'implantation de l'algorithme du crible algébrique sur un système distribué. En terme de complexité temporel effective, ce calcul nécessite de réaliser 10^{20} opérations soit une année de calcul sur un réseau de 2000 processeurs AMD Opteron cadencés à 2,2GHz [KAF$^+$10]

14. CRT pour *Chinese Remainder Theorem* ou Théorème des restes chinois dans la langue de Molière

pour réaliser efficacement l'opéraion de déchiffrement [QC82]. Cette méthode est référencée dans la norme PKCS #1.

1.3.1.3 Signature RSA

Comme pour l'opération de déchiffrement, la signature RSA S d'un message m consiste en une exponentiation modulaire avec l'exposant privé : $S = m^d \bmod N$. La validité de cette signature est vérifiée en utilisant la clé publique du signataire : $m \stackrel{?}{=} S^e \bmod N$.

Dans la pratique, on préfère, signer une empreinte du message m plutôt que de signer directement sa valeur. Pour ce faire, on utilise une fonction hachage $\mathcal{H}$ pour calculer l'empreinte du document à signer $\dot{m} = \mathcal{H}(m)$. Dans ce cas, la signature devient $S = \dot{m}^d \bmod N$. De même, le calcul de l'empreinte est aussi nécessaire pour la vérification de signature. Par ailleurs, on conjugue généralement l'utilisation d'une fonction de hachage à celle d'un schéma de remplissage (*e.g.* RSA-OAEP [BR94] pour le chiffrement et RSA-PSS [BR96] pour la signature). Pour les différentes attaques détaillées dans cette thèse, nous considérerons exclusivement les signatures RSA réalisées à partir d'une fonction de hachage et/ou de remplissage déterministe.

1.3.2 Présentation d'El Gamal

L'algorithme El Gamal appartient aussi à la famille des crypto-systèmes asymétriques. Il a été inventé au milieu des années 80 par T. El Gamal [ElG85], cryptologue américain d'origine égyptienne. Contrairement au RSA, sa sécurité ne repose pas sur le problème de la factorisation mais sur la difficulté supposée du *problème du logarithme discret*, qui semble comparable à celle du problème de factorisation. En effet, le meilleur algorithme connu pour résoudre ce problème est la méthode du calcul d'indice (ou *Index calculus algorithm*) dont la complexité est aussi sous-exponentielle [Adl79, SWD96]. Cependant, il est possible qu'il existe des moyens de casser l'algorithme El Gamal sans résoudre le problème du logarithme discret. Enfin, comme pour le RSA, cet algorithme peut aussi bien être utilisé pour le chiffrement de messages ou la signature de documents électroniques.

1.3.2.1 Génération des clés

Les clés El Gamal comportent une partie publique et une partie privée. Le premier élément de la clé publique est le module p, grand nombre premier tiré aléatoirement. Le second est $g \in \mathbb{Z}_p^*$, un générateur du groupe multipicatif. Enfin, le dernier élément de la clé publique est y, calculé à partir de la clé privée $1 \leq x \leq p-1$ telle que $y \equiv g^x \bmod p$. Ainsi, la clé publique est formée du triplet (p, g, y) et la clé privée est x.

1.3.2.2 Chiffrement El Gamal

Afin de chiffrer un message m, il faut préalablement tirer un aléa k tel que $1 \leq k \leq p-1$ et calculer :

$$c_1 \equiv g^k \bmod p \quad \text{et} \quad c_2 \equiv m \cdot y^k \bmod p \tag{1.13}$$

Le chiffré du message m transmis est alors le couple (c_1, c_2). L'opération de déchiffrement consiste à utiliser la clé privée x pour calculer :

$$\begin{aligned} \frac{c_2}{c_1^x} \bmod p &\equiv \frac{m \cdot y^k}{g^{xk}} \bmod p \\ &\equiv \frac{m \cdot g^{xk}}{g^{xk}} \bmod p \end{aligned} \tag{1.14}$$

Si aucune erreur n'est intervenue dans le chiffrement, le déchiffrement ou la transmission du chiffré, alors le résultat de la fraction est égale au message clair initial m.

1.3.2.3 Signature El Gamal

Comme pour la méthode de chiffrement, la première étape de la signature El Gamal consiste à tirer un aléa k tel que $1 \leq k \leq p-1$ mais il faut aussi que k et $p-1$ soient premiers entre eux. Le calcul de la signature revient alors à :

$$u \equiv g^k \bmod p \quad \text{et} \quad v \equiv \frac{\dot{m} - xu}{k} \bmod p-1. \tag{1.15}$$

ou $\dot{m}$ représente l'empreinte, ou l'image de m par une fonction de hachage. La signature du message m correspond alors au couple $S = (u, v)$.À la réception de la signature, le destinataire utilise la clé publique du signataire pour vérifier si :

$$g^{\dot{m}} \stackrel{?}{\equiv} y^u u^v \bmod p \tag{1.16}$$

1.3.3 Présentation de DSA

Le sigle *DSA* est l'acronyme américain de *Digital Signature Algorithm*. Cet algorithme de signature numérique a été standardisé en 1994 par le NIST aux États-Unis dans la norme FIPS PUB 186 [oSN94], légèrement modifiée en 2000 [oSN00], puis 2001 [oSN01b]. L'utilisation de clés plus longues ainsi que de la fonction de hachage SHA-2 [oSN01a, oSN07] a été approuvée en Juin 2009 [oSN09]. Le DSA est basé sur l'algorithme El Gamal et, de fait, sa sécurité repose sur la difficulté du calcul du logarithme discret.

1.3.3.1 Génération des clés

La clé publique se compose d'un module aléatoire p de taille 1024 bits [15], d'un autre module q de taille 160 bits et tel que $q \mid (p-1)$, et de $g \in \mathbb{Z}_p^*$, un générateur du groupe multiplicatif d'ordre q. La dernière composante de la partie publique y est calculée à partir de la clé privée $x \in \mathbb{Z}_q^*$ telle que $y \equiv g^x \mod p$. Ainsi, la clé publique est formée du quadruplet (p, q, g, y) et la clé privée est x.

1.3.3.2 Signature DSA

Pour calculer la signature d'un message d'entrée m, il faut d'abord tirer un aléa $k < q$, puis, en utilisant une fonction de hachage (*i.e.* SHA-2), on calcule :

$$u \equiv \left(g^k \mod p\right) \mod q \quad \text{et} \quad v \equiv \frac{\dot{m} + xu}{k} \mod q. \tag{1.17}$$

15. En réalité, la norme impose que la taille de p soit comprise entre 1024 et 3072 bits [oSN09].

La signature du message m correspond alors au couple $S = (u, v)$. Afin de s'assurer qu'une signature DSA est valide, il faut d'abord calculer $w \equiv v^{-1} \mod q$ puis vérifier que :

$$u \stackrel{?}{\equiv} \left(g^{w\dot{m}} y^{wu} \mod p\right) \mod q, \tag{1.18}$$

1.3.4 Méthodes d'exponentiation modulaire

L'*exponentiation modulaire* est une opération de base pour l'utilisation des méthodes de cryptographie asymétrique. En effet, que ce soit pour réaliser une signature DSA ou un chiffrement RSA, il faut pouvoir calculer une puissance modulaire du message d'entrée aussi efficacement que possible. Si cette opération est réalisée naïvement :

$$m^d \mod N \equiv \underbrace{m \cdot m \cdot \ldots \cdot m}_{\text{d fois}} \mod N \tag{1.19}$$

alors la complexité calculatoire croît exponentiellement avec la taille de l'exposant ce qui n'est pas acceptable pour implanter efficacement l'exponentiation modulaire. Une méthode tout aussi intuitive, mais bien plus élégante, consiste à exprimer l'exposant en base 2, $d = \sum_{i=0}^{n-1} 2^i \cdot d_i$ et réaliser l'exponentiation modulaire en parcourant les bits d'exposant :

$$m^d \mod N \equiv \prod_{i=0}^{n-1} m^{2^i d_i} \mod N \tag{1.20}$$

Ainsi, l'élévation à la puissance ne se calcule plus en multipliant d fois le message m par lui même mais en multipliant, au plus $log_2(d)$ fois, le message par une puissance carrée de celui-ci. Ces méthodes d'exponentiation, appelées méthodes d'*exponentiation binaire* présentent une complexité qui croît linéairement avec la taille de l'exposant ce qui est nettement plus efficace. Les parties suivantes présentent les implantations classiques d'exponentiation modulaire binaire ainsi que les optimisations ou les variantes les plus répandues.

1.3.4.1 Méthode *"Left-To-Right"*

Le premier moyen de réaliser l'exponentiation modulaire binaire est de parcourir les bits d'exposant des poids forts vers les poids faibles. De ce fait, cette méthode est couramment appelée méthode d'exponentiation modulaire *"Left-To-Right"*. L'exponentiation est réalisée en répétant des carrés modulaires, suivis, selon la valeur du bit courant d'exposant d_i, par des multiplications modulaires. Cette méthode est la plus utilisée en pratique car elle permet d'économiser une variable d'accumulation par rapport à la méthode duale (*i.e.* la variable B cf. Algorithme 3).

1.3.4.2 Méthode *"Right-To-Left"*

L'autre méthode consiste à parcourir les bits de l'exposant en sens inverse, c'est-à-dire des poids faibles vers les poids forts. Cette méthode est aussi appelée exponentiation binaire *"Right-To-Left"*. Elle se différencie de la méthode précédente par l'enchaînement des carrés et des multiplications modulaires ainsi que par l'utilisation d'une variable qui accumule les puissances carrées du message calculées pour chaque itération.

Algorithme 2 Exponentiation binaire *"Left-To-Right"*

ENTRÉES : $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
SORTIE : $S = m^d \bmod N$
1: $A \leftarrow 1$
2: **Pour** $i = (n-1)$ **à** 0 **faire**
3: $A \leftarrow A^2 \bmod N$
4: **Si** $d_i = 1$ **alors**
5: $A \leftarrow A \cdot m \bmod N$
6: **Fin Si**
7: **Fin Pour**
8: **Retourner** A

Algorithme 3 Exponentiation binaire *"Right-To-Left"*

ENTRÉES : $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
SORTIE : $S = m^d \bmod N$
1: $A \leftarrow 1$
2: $B \leftarrow m$
3: **Pour** $i = 0$ **à** $n-1$ **faire**
4: **Si** $d_i = 1$ **alors**
5: $A \leftarrow A \cdot B \bmod N$
6: **Fin Si**
7: $B \leftarrow B^2 \bmod N$
8: **Fin Pour**
9: **Retourner** A

1.3.4.3 Les variantes

Bien que les algorithmes précédents permettent de calculer efficacement l'exponentiation modulaire, leur implantation peut laisser fuir de l'information sur l'exposant utilisé, qui peut être la clé privée dans le cas d'une signature RSA. En effet, dans les algorithmes précédents, l'exécution d'une multiplication est directement liée à la valeur du bit d'exposant. Dans le cadre d'une implantation sur carte à puce, cette dépendance peut permettre à un attaquant potentiel de déduire la valeur de l'exposant en analysant le comportement physique de la puce (*e.g.* consommation électrique [KJJ99], temps d'exécution [Koc96], etc ...). Une solution pour colmater cette faille de sécurité potentielle consiste à rendre l'exécution indépendante de la valeur de l'exposant. Cette variante, aussi connue comme l'exponentiation binaire *Square & Multiply Always* a été proposée par J.-S. Coron à CHES 1999 [Cor99]. L'algorithme suivant détaille cette variante dans le cas *"Left-To-Right"*.

Comme il est possible d'améliorer la sécurité des algorithmes classiques d'exponentiation, il est possible aussi d'améliorer leurs performances. L'algorithme suivant présente l'exponentiation modulaire implanté dans la librairie `OpenSSL`, à savoir l'exponentiation modulaire *par fenêtres glissantes*[16]. Ici, le principe consiste à traiter les bits d'exposant par fenêtres de tailles maximales k bits. Ainsi, en calculant préalablement des puissances modulaires du message m, on peut réduire le nombre d'itérations de l'algorithme d'exponentiation de manière significative.

Enfin, certaines variantes permettent à la fois d'améliorer les performances des méthodes classiques tout améliorant la sécurité. L'exemple ci-dessous présente l'algorithme d'exponenti-

16. Sliding-Window Exponentiation dans la littérature

Algorithme 4 Exponentiation binaire Square & Multiply Always

ENTRÉES : $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
SORTIE : $S = m^d \bmod N$

1: $A_0 \leftarrow 1$
2: **Pour** $i = (n-1)$ **à** 0 **faire**
3: $\quad A_0 \leftarrow {A_0}^2 \bmod N$
4: $\quad A_1 \leftarrow m \cdot A_0 \bmod N$
5: $\quad A_0 \leftarrow A_{d_i}$
6: **Fin Pour**
7: **Retourner** A_0

ation binaire *de Montgomery* [Mon87].À noter que la multiplication modulaire est implantée selon la méthode optimisée du même Montgomery [Mon85] . Évidemment, d'autres implantations efficaces de l'exponentiation modulaire existent mais nous laisserons les lecteurs intéressés se référer à l'article [cKRSP94] pour plus de détails.

Algorithme 5 Exponentiation modulaire par fenêtres glissantes

ENTRÉES : $m, d = \sum_{i=0}^{n-1} 2^i d_i, N, k \geq 1$
SORTIE : $S = m^d \bmod N$

{On précalcule les puissances du message}
$m_1 \leftarrow m$
$m_2 \leftarrow m^2 \bmod N$
Pour $i = 1$ **à** $2^{k-1} - 1$ **faire**
  $m_{2i+1} = m_{2i-1} \cdot m_2 \bmod N$
Fin Pour
{Début de l'exponentiation modulaire}
$A \leftarrow 1$
$i \leftarrow n - 1$
Tant que $i \geq 0$ **faire**
  Si $d_i == 0$ **alors**
    $A \leftarrow A^2 \bmod N$
    $i \leftarrow i - 1$
  sinon
    {Trouver la plus longue fenêtre de bits $d_i d_{i-1} \dots d_l$
    de telle sorte que $i - l + 1 \leq k$ and $d_l = 1$}
    $A \leftarrow A^{2^{i-l+1}} \cdot m_{d_i d_{i-1} \dots d_l} \bmod N$
    $i \leftarrow l - 1$
  Fin Si
Fin Tant que
Retourner A

Algorithme 6 Exponentiation binaire de Montgomery

ENTRÉES : $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
SORTIE : $S = m^d \bmod N$

$A_0 \leftarrow 1$
$A_1 \leftarrow m$
Pour $i = (n - 1)$ **à** 0 **faire**
  $A_{\bar{d_i}} \leftarrow A_0 \cdot A_1 \bmod N$
  $A_{d_i} \leftarrow {A_{d_i}}^2 \bmod N$
Fin Pour
Retourner A_0

Chapitre 2

Les attaques par perturbation

Sommaire

2.1 Introduction

Les premières perturbations de composants électroniques ont été constatées dans les années 1970, lorsqu'il a été observé que les émissions de particules radioactives pouvaient perturber le bon fonctionnement de puces [MW78]. Les rayons cosmiques ont également été identifiés comme une source d'erreur potentielle. Cette observation a même motivé des recherches, dans le domaine de l'aéronautique, visant à modéliser les effets de ces rayons sur des semi-conducteurs [Zie79]. Cependant, de telles perturbations ne sont pas contrôlables et ne pouvaient donc pas être exploitées.

Parallèlement à l'élaboration de méthodes visant à protéger les composants contre les perturbations, la production volontaire de fautes a aussi concentré une partie des recherches [Hab92].

Mais, c'est seulement à la fin des années 90 que les premières exploitations concrètes de perturbations volontairement induites sur des puces ont été révélées. Dans l'article fondateur [BDL97], trois chercheurs du laboratoire *Bellcore* montrent que la perturbation volontaire de composants embarquant des méthodes cryptographiques peut engendrer une fuite d'information critique. Outre ce nouveau concept d'*attaque par perturbation* , les auteurs de l'article ont proposé différentes applications contre les implantations d'algorithmes de chiffrement asymétrique, dont une assez spectaculaire contre l'implantation CRT de la signature RSA [BDL97, BDL01].

Quelques temps plus tard, E. Biham et A. Shamir proposèrent d'utiliser les perturbations pour attaquer les implantations d'algorithmes de chiffrement symétrique, même partiellement inconnus [BS97, BS96]. Ces attaques par perturbations, exploitant les différences entre des sorties correctes et erronées, furent alors baptisées *Differential Fault Analysis* . La popularité grandissante qu'a connu cette classe d'attaque, lors de ces dix dernières années, peut sans doute s'expliquer par la facilité avec laquelle il est possible d'exploiter les perturbations d'un composant [BDL97, BECN$^+$04] comme par la difficulté d'élaborer des contre-mesures efficaces (*cf.* Chapitre 11).

De nos jours, les attaques par perturbations sont considérées comme une menace importante par l'industrie de la carte à puce. D'ailleurs, les concepteurs de composants sécurisés n'hésitent pas à inclure des contre-mesures spécifiques, logicielles et/ou matérielles, pour protéger leur produits. La résistance des puces face à ce type d'attaque physique fait même partie des points vérifiés par les CESTI.

Dans la suite de ce chapitre, nous commencerons par présenter les différentes méthodes pratiques élaborées pour perturber le bon fonctionnement de puces. Nous développerons ensuite les différents modèles de perturbation considérés dans la littérature. Enfin, nous détaillerons les attaques de référence publiées contre chacun des types d'algorithmes que nous avons plus particulièrement étudiés pendant la thèse.

2.2 Les perturbations en pratique

Depuis l'introduction des attaques par perturbation, à la fin des années 90, de nombreuses méthodes pour injecter des fautes ont été développées. Dans cette section, nous allons détailler les méthodes parmi les plus utilisées, mais aussi apparaissant le plus souvent dans la littérature.

2.2.1 Attaques par impulsions électriques

Le principe consiste à appliquer une variation brève sur la tension d'alimentation de la carte à microprocesseur ou sur la fréquence d'horloge. Ces attaques sont communément appelées *Glitch Attack* dans la littérature [AK96, ABF$^+$02, BECN$^+$04].

Les attaques sur la tension d'alimentation peuvent être réalisées en provoquant soit un pic d'alimentation (supérieur à la tension Vcc), soit une micro-coupure (tension comprise entre 0V et Vcc). De manière générale, la perturbation de la tension d'alimentation peut provoquer une mauvaise interprétation, voire le saut d'une instruction de la part du microprocesseur.
Les variations de la fréquence d'horloge, quant à elles, peuvent provoquer un cadencement d'une partie de la logique sans que l'entrée ne soit encore stabilisée ou définie. La sortie sera donc faussée. Ainsi, les attaques sur la fréquence d'horloge peuvent perturber des lectures de données en mémoire comme l'exécution de micro-instructions.

La mise en oeuvre de ce type d'attaque est peu coûteuse. En effet, les attaques par impulsions électriques peuvent être réalisées avec un simple générateur de signal. Cependant, il est

relativement difficile d'utiliser cette méthode pour ne perturber qu'une partie spécifique du composant ou ne cibler que certaines opérations pendant l'exécution d'une fonction cryptographique. D'autre part les cartes actuelles embarquent des contre-mesures matérielles, telles que des filtres ou des détecteurs de surtension, permettant de les protéger contre des signaux pouvant provoquer des erreurs. Notons enfin que les composants modernes fonctionnent sur une horloge interne décorrélée de l'horloge externe, ce qui rend les *Glitch Attacks* sur l'horloge quasiment inexploitables.

2.2.2 Attaques par illumination

Une autre méthode d'attaque consiste à tirer profit de la sensibilité à la lumière des circuits électroniques. Une exposition brève d'un circuit électronique à une source lumineuse puissante peut induire des courants additifs provoquant des erreurs dans les portes logiques illuminées. Ces attaques furent notamment introduites en 2002 par S. Skorobogatov et R. Anderson [SA02] en utilisant le flash d'un appareil photo pour basculer la valeur d'un bit d'une cellule de mémoire. Cependant, les attaques par illumination ont été mises en pratique et utilisées concrètement dès le début des années 1990, dans le cadre d'évaluations ITSEC, par Raphaël Bauduin au CNET de Caen.

De nos jours, les attaques par illuminations sont sans doute les plus utilisées pour perturber les composants. Mais, bien que, comme dans l'exemple précédent, ces attaques puissent être réalisées avec des moyens peu onéreux, on préfère utiliser en pratique des lasers [BECN$^+$04]. D'une part car la lumière d'un flash peut être facilement détectée par des détecteurs intégrés à la puce. Mais surtout car l'utilisation de tirs lasers permet une grande précision spatiale (zone de la puce à perturber) et temporelle (instant et durée d'une impulsion). L'utilisation de sources lumineuses offre un large spectre de perturbations possibles puisque les principaux effets associés sont la modification de données en mémoire (ou d'adresses, ou de codes d'opérations) ainsi que le déroutement de processus. De récents travaux ont montré que la réalisation de tirs lasers sur la mémoire RAM pouvaient permettre de modifier la configuration de certains FPGAs, induisant les effets pré-cités [CMV$^+$09, Can09].

Bien que ce type d'attaque soit très efficace, certains inconvénients peuvent la rendre inutilisable pour un novice. D'une part, les attaques par illumination nécessitent de mettre la puce à nu afin que le rayon lumineux émis puisse atteindre le composant et provoquer les perturbations escomptées. Par ailleurs le coût d'un banc laser peut représenter un obstacle financier pour certains attaquants. Enfin, si la puissance émise par le laser est mal dosée, l'attaquant peut tout simplement détruire le composant cryptographique, ou le rendre inutilisable.

2.2.3 Attaques par impulsions magnétiques

Cette attaque imaginée par J.-J. Quisquater et D. Samyde consiste à émettre un champ magnétique intense à proximité du composant dont on souhaite perturber le bon fonctionnement [QS02]. Le champ magnétique crée des courants locaux (connus sous le nom de courants de Foucault) sur la surface du composant pouvant provoquer des fautes. De tels effets peuvent être obtenus en approchant une sonde chargée (*i.e.* un fil électrique alimenté et enroulé autour d'une aiguille pourra faire l'affaire) à proximité de la surface du circuit.

L'efficacité de cette attaque pourra être améliorée si le composant est débarrassé de la couche plastique le recouvrant. En effet, plus le champ magnétique est appliqué près du silicium et plus les courants induits seront intenses. Cette préparation rend l'attaque semi-invasive. Enfin, la puissance du champ magnétique comme sa durée et la localisation de ces effets semblent être

relativement difficiles à maîtriser en pratique.

D'autres méthodes d'attaques sont listées dans la littérature. C'est le cas de la variation de la température du composant [BECN+04]. Le principe consiste à réchauffer/refroidir le composant de façon à l'utiliser au-delà des températures de fonctionnement préconisées. Dans ce cas, deux effets sont principalement observés : modification aléatoire des données en RAM et perturbations des opérations de lecture/écriture. Une autre méthode consiste à émettre un faisceau de rayons X ou d'ions lourds sur le composant [BECN+04].

2.3 Les modèles de perturbation

Dans la section précédente, nous avons décrit les méthodes de perturbation les plus utilisés en pratique et nous avons détaillé les effets induits pour chacune d'entre elles. À partir de cette description, il est possible d'élaborer des modèles de perturbation qui décriront, plus ou moins précisément, les effets que l'on peut espérer obtenir en perturbant un composant cryptographique.

2.3.1 Types de perturbation

Classiquement, on distingue deux grands types de perturbations. Comme leur nom l'indique, les perturbations *permanentes* consistent à réaliser une modification définitive d'une cellule mémoire ou d'un circuit. Cette modification peut concerner des variables en mémoire et du code contenus dans de la mémoire non volatile, ou encore le cablage du circuit (découpage d'un fil). Un exemple d'utilisation de ce modèle est présenté dans une attaque contre l'implantation de la signature RSA (*cf.* Section 2.4.2).

Cependant, les perturbations peuvent aussi avoir des effets éphémères. On parle alors de perturbations *transitoires* . Ces perturbations ont pour effets de modifier temporairement l'exécution de code ou d'une opération. En particulier, le composant retrouve son comportement original après une réinitialisation, ou lorsque la source de la perturbation n'émet plus de signaux compromettants. La perturbation de données en RAM, persistant tant que la donnée n'est pas écrasée ou effacée (*e.g.* par une réinitialisation du composant), est un exemple typique de perturbation transitoire. Ce type de faute est certainement le plus étudié et le plus utilisé pour attaquer les implantations d'algorithmes cryptographiques [BDJ+98, BDL97, Gir05a].

2.3.2 Modélisation des effets de perturbations

Suivant les moyens choisis par l'attaquant pour injecter des fautes sur un composant cryptographique, différents modèles de perturbations pourront être considérés. La modélisation formalise les effets qu'il est possible d'obtenir sur un composant cryptographique (*i.e.* perturbation aléatoire ou écrasement d'une donnée, déroutement) et permet d'exploiter les sorties erronées. Le modèle doit, en outre, tenir compte des aptitudes de l'attaquant à contrôler les perturbations (localisation spatiale et temporelle, nombre de bits perturbés). Plusieurs modèles apparaissent dans la littérature [Gir05c, BO06, Ott04], en nous inspirant de ces derniers, avons donc choisi d'opter pour la classification suivante. Tous les modèles de perturbation que nous allons présenter formalisent des effets observés expérimentalement.

2.3.2.1 Modification de la valeur d'un bit

Dans ce modèle, on suppose qu'un attaquant a la possibilité de modifier la valeur d'un bit. Cette modification peut indépendamment *basculer* la valeur du bit [17] ou *écraser* la valeur pour la fixer à 0 ou 1 [18]. Dans le modèle le plus contraint, la localisation du bit perturbé peut être maîtrisée par l'attaquant. Ce modèle de perturbation est le plus ancien et a été considéré pour attaquer de nombreuses implantations d'algorithmes cryptographiques [BDL97, BMM00, Gir05a], et particulièrement dans le cadre des algorithmes de chiffrement à flot [BS97, BCC+09, KY09].

Cependant, ce modèle semble aujourd'hui dépassé par rapport à l'évolution des méthodes de protections incluses dans les composants récents (chiffrement des mémoires, détection/corrections d'erreurs). Il est donc relativement difficile à mettre en pratique sur les composants actuels. Malgré tout, ce modèle reste intéressant car il permet de décrire assez simplement les principes d'attaques qui pourront être ensuite étendus à des modèles moins contraints.

2.3.2.2 Modification de quelques bits

Un autre modèle consiste à considérer que plusieurs bits peuvent être modifiés par l'injection d'une seule perturbation. Ce modèle prend en compte le fait que les composants actuels sauvegardent et chargent les données par blocs, dont la taille correspond aux spécificités de leur architecture. Typiquement, on considère que les perturbations altèrent la valeur d'un octet entier mais ceci pourra tout à fait être étendu à des composants basés sur des architectures 16 bits, voire 32 bits. Le spectre de ce modèle couvre également les fautes visant un bit spécifique de la mémoire mais dont les effets se propagent sur les bits voisins. Ce phénomène est particulièrement courant dans la pratique. Le nombre de bits perturbés devra être borné pour permettre à un attaquant de retrouver l'erreur induite par une recherche exhaustive.

Contrairement au modèle précédent, celui-ci est considéré comme réaliste car il reflète véritablement des effets de perturbations observés et permet d'exploiter des cas réels de perturbations [DLV03, Gir05a, GK04]. Comme nous l'avons dit précédemment, ce modèle est particulièrement adapté aux perturbations intervenant pendant la lecture ou le chargement de données. D'autre part, lorsque la mémoire est chiffrée ou codée par blocs, la modification d'un ou de quelques bits du bloc se propagera sur l'ensemble du bloc au moment du déchiffrement/décodage. Par ailleurs, suivant les contre-mesures utilisées pour protéger les mémoires (brouillage des adresses), la localisation spatiale de la faute sera plus ou moins difficile.

Pour ces différentes raisons, nous avons choisi ce modèle, pour réaliser différentes attaques contre les clés publiques d'implantations d'algorithmes de signature électronique.

2.3.2.3 Modification aléatoire

Dans le dernier modèle considéré, les fautes sont injectées de telle sorte que l'attaquant ne puisse pas identifier la nature de la perturbation induite. Dans ce cas, ni la localisation spatio-temporelle de la perturbation, ni le nombre de bits modifiés ne peuvent être identifiés par l'attaquant. Évidemment, ce modèle de perturbation est le plus facile à réaliser en pratique mais, bien souvent, il est aussi le plus difficile à exploiter. Cependant, il a déjà été suffisant pour démontrer la vulnérabilité de certaines implantations [YKLM01].

L'utilisation de ce modèle est motivée par la multitude de méthodes de protections qui peuvent être déployées sur des composants sécurisés modernes. En effet, le chiffrement de la mémoire,

17. *bit-flip fault model* dans la littérature
18. *Stuck-at fault model* dans la littérature

le brouillage d'adresse ou encore l'utilisation d'horloges asynchrones et aléatoires sont autant de facteurs limitant le contrôle des fautes injectées. Bien qu'il soit très intéressant d'imaginer des attaques suivant ce modèle, les concepteurs de solutions sécurisées devront, en revanche se méfier d'un attaquant aussi faible. En effet, la sécurité de l'implantation d'un algorithme ne devra pas seulement reposer sur des mesures empêchant le contrôle des perturbations.

2.3.2.4 Perturbations multiples

L'idée de perturber plusieurs fois l'exécution d'un composant a été publiée pour la première fois par C.H. Kim et J.-J. Quisquater [KQ07]. Dans cette application contre une implantation protégée de RSA-CRT, le principe consiste à répéter une attaque par impulsion électrique de manière à sauter des instructions à deux moments distincts de l'exécution. Dans ce cas précis, on peut parler d'analyse de perturbations de *second-ordre* [DGRS09]. Bien qu'il soit admis qu'un composant puisse être perturbé plusieurs fois de la même manière (*e.g.* attaque par impulsion électrique dans l'exemple précédent), il est, en revanche, plus difficile d'imaginer un attaquant panachant différentes méthodes pour perturber plusieurs fois la même exécution. D'autre part, même si l'attaquant choisit d'utiliser la même méthode pour perturber un composant à différents instants, les effets induits sur le composant à chaque injection de faute seront différents en général. Les perturbations multiples de composant, que ce soit spatiales ou temporelles, ont déjà été réalisées avec succès dans certains laboratoires de tests et d'évaluations.

Dans la suite de la thèse, nous nous limiterons à l'étude des perturbations au premier ordre. Cependant, la possibilité de réaliser des fautes multiples devra être considérée pour la conception de composants sécurisés.

2.4 Attaques classiques et contre-mesures

Dans la partie précédente, nous avons présenté les modèles de perturbation utilisés pour décrire au mieux le comportement de composants soumis à des injections de fautes. En combinant ces modèles avec une connaissance, plus ou moins détaillée, de l'implantation d'algorithmes cryptographiques, il est possible d'évaluer le degré de dépendance entre la sortie erronée et les informations secrètes, ou mieux, de retrouver ces informations secrètes.

Dans le paragraphe suivant, nous illustrerons l'exploitation malicieuse de perturbations via différentes applications contre chacun des deux types d'algorithmes de chiffrement. Dans le cadre des applications contre des algorithmes de chiffrement symétrique, nous avons choisi RC4 car nous nous sommes intéressés, dans cette thèse, aux perturbations d'algorithmes de chiffrement à flot. Par ailleurs, la seconde attaque décrite utilise un modèle de faute original, basé sur les caractéristiques de RC4, qui pourra inspirer de futurs travaux dans ce domaine. Pour les algorithmes de chiffrement asymétrique, nous avons pris l'exemple du célébrissime RSA car ces implantations font parties des premières victimes des attaques par perturbations.

2.4.1 Application au chiffrement symétrique : Exemple de RC4

2.4.1.1 Perturbation classique de l'état interne

Depuis leur apparition à la fin des années 90, les attaques par perturbation ont fait l'objet de multiples applications. Cependant, on dénombre peu de travaux ayant attrait à la sécurité des implantations d'algorithmes de chiffrement à flot face aux perturbations.À CHES 2004, J. J. Hoch & A. Shamir ont comblé ce vide en proposant une méthode générique pour analyser les perturbations d'algorithmes de chiffrement à flot basés sur des registres à décalage linéaire [HS04].

Afin d'illustrer leur attaque, ils ont proposé différentes applications contre des algorithmes de chiffrement à flot parmi les plus utilisés. En particulier, l'article présente une méthode pour exploiter des perturbations affectant l'état interne de RC4. Depuis, les perturbations sur l'état interne sont considérés comme un modèle classique pour évaluer la résistance d'implantations d'algorithmes de chiffrement à flot.

Dans le cadre de leur application sur RC4, les auteurs ont considéré un attaquant capable de modifier la valeur d'un octet de la permutation S et ce, juste après la phase d'initialisation (*cf.* Section 1.2.1.1). Or, les registres de S contiennent toutes les valeurs possibles que peut prendre un octet (*i.e.* de 0 à 255). En modifiant la valeur d'un registre, on remplace donc une valeur possible par une autre, déjà stockée dans la permutation S. Par conséquent, la valeur retirée n'apparaîtra jamais dans la suite chiffrante alors que la valeur erronée apparaîtra, en moyenne, deux fois plus souvent que les autres. Les auteurs estiment qu'il faut analyser environ 10000 octets de suite chiffrante pour identifier la valeur écrasée et la valeur résultant de la perturbation. Ceci permet donc de déduire un octet de l'état interne après l'initialisation. Pour retrouver le reste de l'état interne, concentrons nous sur la perturbation de l'octet $S[1]$. En comparant les suites chiffrantes correctes et erronées, on peut distinguer trois manifestations différentes de la perturbation, correspondant respectivement à trois cas distincts. Le cas le plus intéressant est illustré par la figure 2.1. Il concerne le cas où $S[1]$ est perturbé de telle sorte que l'octet erroné correspondant soit $\hat{S}[1] = \varepsilon$ avec $\varepsilon \in [\![0; 255]\!]$. Cette valeur perturbée se propage alors jusque dans la suite chiffrante. Comme nous le remarquions précédemment, il faut analyser plusieurs

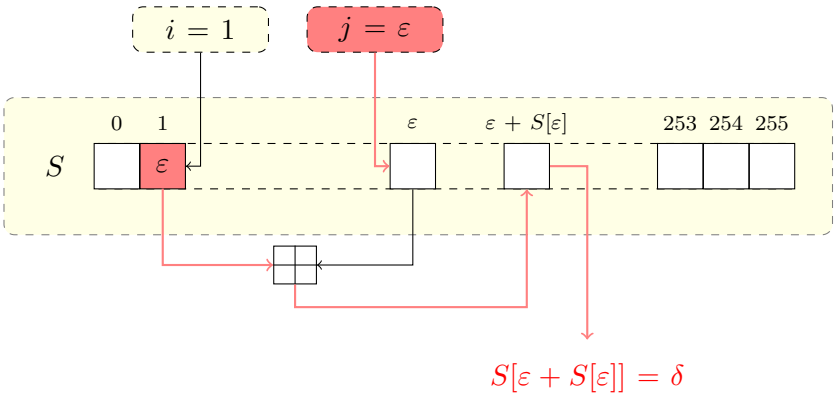


FIGURE 2.1 – Illustration de la perturbation de $S[1]$ dans RC4

milliers d'octets de la suite chiffrante erronée pour identifier la valeur initiale de $S[1]$ ainsi que la valeur perturbée ε. Par ailleurs, la lecture d'un octet erroné de suite chiffrante δ permet de déduire directement un octet de l'état interne car : $\delta = S[\varepsilon + S[\varepsilon]]$. Ainsi, chaque perturbation différente de $S[1]$ nous donnera une nouvelle relation qui sera utilisée pour retrouver un octet de la permutation S, après initialisation. Nous laisserons le lecteur se référer à l'article original [HS04] pour les détails de l'analyse des autres cas d'erreurs.

Pour obtenir la totalité de l'état interne de RC4, après l'initialisation, les auteurs ont estimé qu'il fallait analyser près de 2^{26} octets de suite chiffrante résultant de 2^{16} perturbations différentes sur la permutation S.

2.4.1.2 Analyse des états "impossibles" de RC4

À FSE 2005, E. Biham *et al.* ont proposé d'améliorer les attaques par perturbation sur RC4 [BGN05]. Parmi les deux méthodes proposées dans l'article, l'exploitation des *états impossibles* de RC4 a particulièrement retenu notre attention. Le principe de cette attaque consiste à perturber l'état interne de l'algorithme de façon à ce qu'il se mette dans un état impossible à atteindre normalement. Bien que [BGN05] propose une attaque par perturbation encore meilleure, nous avons choisi de décrire celle-ci car elle utilise un modèle de faute original et, qui plus est, basé sur des caractéristiques propres à RC4.

Les états impossibles ont été initialement décrits par H. Finney [Fin94] et sont aussi appelés *états de Finney*. Leur caractérisation est décrite dans la figure 2.2. Ces états impossibles sont

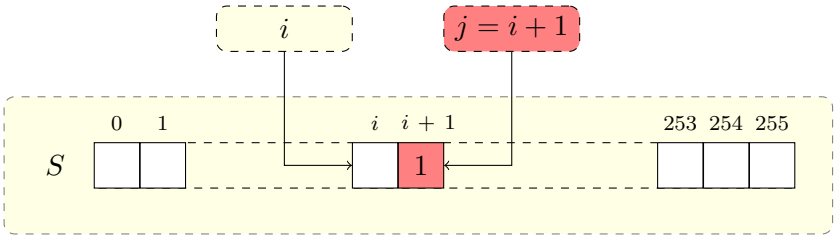


FIGURE 2.2 – Perturbation "impossible" de RC4

atteint lorsque $j = i + 1$ et que simultanément $S[j] = 1$. Il est intéressant de noter que lorsque l'état interne est dans un état impossible, les états suivants seront aussi des états impossibles, l'octet $S[j] = 1$ étant échangé avec l'octet $S[j + 1]$ et les registres i et j étant incrémentés de 1. Ces états sont normalement impossible à atteindre à cause de la mise à zéro des pointeurs i et j réalisée avant la génération de la suite chiffrante (*cf.* Section 1.2.1.2). Cependant, la perturbation des registres d'état interne de RC4 peut *"rendre l'impossible possible"* [BGN05]. Aussi, les auteurs ont considéré un attaquant capable de perturber aléatoirement la valeur de chacun des pointeurs i et j. Sous ce modèle, la probabilité qu'une faute provoque un état impossible est d'une chance sur 2^{16} (*i.e.* $j = i + 1$ avec une probabilité 2^{-8} et $S[j] = 1$ avec une probabilité 2^{-8} également pour une perturbation aléatoire de j).

Le premier intérêt de ce modèle est qu'il est relativement simple de savoir si la perturbation a mis l'algorithme dans un état impossible ou pas. Pour cela, il suffit d'analyser les octets de suite chiffrante postérieurs à l'injection de la faute. Les auteurs ont remarqué que, dans le cas où la faute n'a pas provoqué un état de Finney, on peut s'attendre à observer une collision, soit deux octets de même valeur, dans les 20 octets de suite chiffrante retournés après la perturbation. Par conséquent, si aucune collision n'est observée sur les 30 octets de la suite chiffrante postérieurs à la perturbation, alors l'état interne est très probablement dans un état de Finney.

L'analyse de la suite chiffrante permet, non seulement, de savoir si la faute injectée est en accord avec le modèle mais, elle permet en outre de retrouver la valeur de l'état interne de RC4 juste avant la perturbation. Pour cela, il suffit de noter les registres de S qui subissent une permutation circulaire d'un octet toutes les 255 itérations à cause de l'injection de faute. De plus, comme l'indice de l'octet renvoyé dans la suite chiffrante se répète toutes les 255 itérations également, alors, en observant $255 \cdot 256$ octets de suite chiffrante on obtient un entrelacement de tout l'état interne postérieur à la faute. Ainsi, en déduisant les valeurs correctes de i et j, il est possible remonter jusque l'état interne de RC4 juste avant la perturbation. Cette attaque nécessite de réaliser autant de perturbations que pour l'attaque de Hoch & Shamir [HS04] (*i.e.*

2^{16}), en revanche, elle nécessite d'analyser 2^{21} octets de suite chiffrante erronée ce qui est moins que pour l'attaque précédente.

2.4.2 Application au chiffrement asymétrique : Exemple de RSA

2.4.2.1 Perturbation de la signature RSA

Les chercheurs de Bellcore ont été des précurseurs dans le domaine des attaques par perturbation. Ils décrivent dans l'article de référence [BDL97], puis dans [BDL01], comment exploiter avantageusement la perturbation de signatures RSA, en mode standard, pour retrouver la clé privée. Cette attaque a été mise au point contre l'algorithme d'exponentiation binaire *"Right-To-Left"* (*cf.* Section 1.3.4.2). Le principe général consiste à perturber la zone mémoire contenant la valeur temporaire du calcul, en basculant volontairement la valeur d'un bit. Afin de clarifier la description de l'attaque, nous traiterons le cas où la perturbation est permanente, mais une perturbation transiante peut aussi être exploitée.

L'attaque se déroule alors en deux phases distinctes. La première phase est *"en-ligne"* car elle consiste à collecter un nombre suffisant de couples $(m_i, \hat{S}_i)$ en introduisant une faute permanente sur la valeur du calcul intermédiaire. La deuxième phase est complètement *"hors-ligne"* et consiste à analyser les erreurs introduites. Soit S_i la signature correcte et modélisons par $\varepsilon(\dot{m}) = \pm 2^b$ le basculement volontaire de la valeur d'un bit avec $0 \leq b < n$. Cette erreur sur un calcul intermédiaire est supposée permanente et, peut donc être formalisée ainsi :

$$\hat{S}_i \equiv \left(\prod_{j=0}^{t-1} \dot{m}_i^{2^j d_j} \pm 2^b \right) \cdot \prod_{j=t}^{n-1} \dot{m}_i^{2^j d_j} \mod N \tag{2.1}$$

$$\equiv S_i \pm 2^b \cdot \prod_{j=t}^{n-1} \dot{m}_i^{2^j d_j} \mod N \tag{2.2}$$

$$\text{ou encore} \quad S_i \equiv \hat{S}_i \pm 2^b \cdot \dot{m}_i^{d_{[t]}} \mod N \tag{2.3}$$

avec $d_{[t]} = \sum_{j=t}^{n-1} 2^j d_j$. En utilisant la propriété de vérification de signature, on obtient l'équation suivante :

$$\dot{m}_i \equiv (\hat{S}_i \pm 2^b . \dot{m}_i^{d_{[t]}})^e \mod N \tag{2.4}$$

Cette équation est remarquable car elle ne dépend plus que de l'empreinte du message $\dot{m}_i$ et de sa signature erronée $\hat{S}_i$. Par ailleurs, on peut remarquer que la perturbation a permis d'isoler une partie $d_{[t]}$ de l'exposant privé d. On retrouve alors cette partie en faisant des hypothèses sur sa valeur et en vérifiant cette hypothèse par un calcul (*i.e.* approche *guess-and-determine*).

Ensuite, l'analyse proprement dite consiste à retrouver l'intégralité des bits de d par fenêtres de w bits, en commençant par les poids forts (*i.e.* d_{n-1}). Pour ce faire, on cherche simultanément les valeurs de $d_{[t]}$ et $\pm 2^b$ qui vérifient l'équation (2.4). La valeur de $d_{[t]}$ trouvée contient alors certains bits connus de d et au plus l bits inconnus. Ces l bits peuvent être retrouvés en cherchant les valeurs dans $[\![0; 2^l - 1]\!]$ qui vérifient (2.4). *A priori*, l'attaquant ne sait pas, pour une signature erronée $\hat{S}_i$ donnée, à quel moment l'algorithme a été perturbé. Donc, pour chaque valeur $d_{[t]}$ cherchée, l'attaquant réalise les tests pour tous les couples $(m_i, \hat{S}_i)$. Selon les auteurs, il est possible de retrouver la clé privée d de cette manière, avec une probabilité supérieure à $\frac{1}{2}$, à partir de (n/w) *log* $(2n)$ couples messages/signatures erronées. et au prix de $O((2^w n^3 log^2(n))/w^3)$ exponentiations. Cette attaque été améliorée et généralisée [19] beaucoup plus tard par J. Blömer et M. Otto [Ott04].

19. Généralisation à l'exponentiation modulaire de type *"Left-To-Right"*

2.4.2.2 Perturbation du déchiffrement RSA

Quelques semaines après la publication de l'attaque précédente F. Bao *et al.* font état dans [BDJ+96, BDJ+98, BECN+04] d'une attaque contre le déchiffrement RSA. Ainsi cette attaque a permis de renforcer l'impact des attaques par perturbation et d'augmenter leur portée dans le domaine de la cryptographie asymétrique.

Le principe de cette attaque repose sur l'introduction d'une faute pendant le déchiffrement RSA, en mode standard, de telle sorte que cette faute puisse être interprétée comme une modification d'un bit d'exposant privé. Dans ce cas, l'opération de déchiffrement d'un chiffré C pourra être caractérisée ainsi :

$$\tilde{m} \equiv C^d \mod N \equiv C^{\sum_{i=0}^{n-1} 2^i d_i} \mod N \tag{2.5}$$

où d_i désigne le i-ème bit de d. Dans le cas d'un déchiffrement perturbé, le message déchiffré $\hat{m}$ sera noté :

$$\hat{m} \equiv C^{\hat{d}} \mod N \tag{2.6}$$

L'erreur introduite pouvant être interprétée comme un basculement de la valeur du j-ème bit de d, on a alors :

$$\hat{m} \equiv C^{\sum_{i=0, i\neq j}^{n-1} 2^i d_i + 2^j \bar{d}_j} \mod N \tag{2.7}$$

Pour réaliser l'analyse de la faute, on divise le déchiffré erroné $\hat{m}$ par le déchiffré correct $\tilde{m}$. On déduit alors :

- Soit $\frac{\hat{m}}{\tilde{m}} \equiv C^{-2^j} \mod N \Rightarrow d_j = 1$,
- Soit $\frac{\hat{m}}{\tilde{m}} \equiv C^{2^j} \mod N \Rightarrow d_j = 0$.

De cette manière, on arrive à déduire le j-ème bit d'exposant privé. Il suffit alors de répéter ce procédé jusqu'à obtenir suffisamment d'informations sur l'exposant privé pour pouvoir retrouver les bits manquants par des méthodes classiques (*i.e. Baby-Step Giant-Step* de Shanks). Notons que cette attaque peut aussi être exploitée si l'on ne connaît que le déchiffré erroné [JQBD97] ou en cas d'erreurs multiples [BDJ+98]. Par ailleurs, le principe de l'attaque pourra être transposé pour attaquer des crypto-systèmes basés sur le problème du logarithme discret (*i.e.* déchiffrement El Gamal ou signature DSA).

Chapitre 3

Contributions et Résultats

Au moment de commencer la rédaction de ce mémoire, j'ai recherché les raisons initiales qui m'ont poussé à m'intéresser aux attaques par perturbations. Pour avoir un début d'explication, il faut remonter jusqu'au stage de fin d'études que j'ai effectué à SAGEM Sécurité. C'est donc en 2007 que j'ai découvert les attaques par canaux auxiliaires via l'attaque de J.-P. Seifert exploitant les mécanismes de branchements prédictifs des microprocesseurs [AGS07]. La possibilité d'utiliser des moyens alternatifs pour retrouver des informations critiques m'avait complètement séduit. En parcourant plus profondément la littérature attenante, j'ai découvert d'autres méthodes alternatives et l'une d'entre elles a particulièrement retenu mon attention : les *attaques par perturbation*. En effet, savoir qu'une exécution erronée puisse révéler du secret est à la fois fascinant et effrayant. Fascinant car il faut souvent ruser ou aiguiser sa curiosité pour extraire le secret des résultats d'exécutions erronées. Effrayant car, bien souvent, l'efficacité de ce type d'attaque est redoutable. C'est pourquoi, lorsque l'opportunité de préparer une thèse dans ce domaine s'est présentée, je n'ai pas hésité à postuler.

Dans ce mémoire, je présenterai les différentes problématiques que j'ai adressées durant ces trois années de thèse. Afin de faciliter la lecture de ce mémoire, j'ai choisi d'organiser ma réflexion autour de trois grands thèmes : les attaques par perturbations des éléments publics, l'étude du comportement de nouveaux algorithmes de chiffrement à flot vis-à-vis des perturbations et l'analyse de contre-mesures via l'exemple de RSA-CRT. Ce mémoire est donc organisé de la façon suivante.

Dans la partie II nous nous intéresserons à l'exploitation des perturbations d'éléments publics pendant l'exécution de primitives cryptographiques utilisant les données secrètes. Nous commencerons par montrer le principe de notre attaque contre des implantations classiques de RSA. Puis, nous généraliserons la méthode d'attaque aux implantations de RSA utilisant la contre-mesure de masquage d'exposant. Nous finirons l'étude en proposant une attaque contre les éléments publics de DSA améliorant largement l'état de l'art. Notons que chacune des attaques contre les implantations de RSA ont fait l'objet de publications respectivement à CHES 2008 [BCG08b], CT-RSA 2009 [BCDG09], CHES 2010 [BDG10] (avec le titre honorifique de *"Best Paper Award"*) ainsi que dans le livre *"Fault Analysis in Cryptography"* [BCDGar].

Ensuite, nous étudierons la sécurité des implantations d'algorithmes de chiffrement à flot vis-à-vis des perturbations. En particulier, nous détaillerons, dans la partie III, les attaques que nous avons imaginées contre les implantations de deux des finalistes du projet eSTREAM. Dans un premier temps, nous montrerons que malgré la présence de non-linéarité dans l'état interne de Grain-128 il est possible d'exploiter les perturbations de la partie linéaire pour retrouver la clé privée. Puis, sous un modèle peu étudié dans la littérature, nous montrerons que les perturbations

d'implantations de RABBIT peuvent engendrer une fuite d'information critique. Ces résultats ont été publiés respectivement à HOST 2009 [BCC+09] et INDOCRYPT 2009 [BCG09].

Nous terminerons ce mémoire en soulignant toute la difficulté pour élaborer des contre-mesures efficaces contre les perturbations malicieuses. Dans cette optique, la partie IV présentera un état de l'art complet de l'évolution des implantations du RSA-CRT. Nous montrerons comment l'évolution des attaques a motivé la conception de contre-mesures de plus en plus résistantes. Cette étude sera complétée par l'attaque que nous avons imaginée contre la contre-mesure M. Ciet et M. Joye [JC05]. Cette attaque a été présentée à la conférence FDTC 2008 [BCG08a].

Deuxième partie

Attaques par perturbation des éléments publics

Chapitre 4

Analyse de la perturbation d'un module RSA

Sommaire

4.1 État de l'art

L'apparition des attaques par faute à la fin des années 90 a ouvert de nouvelles perspectives pour analyser la sécurité des implantations d'algorithmes cryptographiques. Dans le cas du crypto-système asymétrique RSA, en mode standard, les premières attaques par faute étaient

basées sur la perturbation des paramètres privés (*i.e.* l'exposant privé d, *cf.* Section 2.4.2). Aussi, les paramètres privés étant amenés à être protégés contre les perturbations, certains se sont demandés si la perturbation des éléments publics pouvait aussi entraîner une fuite d'information compromettante. Un début de réponse à cette question est apparu avec l'attaque de J.-P. Seifert [Sei05] contre le mécanisme de vérification de signature RSA. Contrairement aux attaques précédentes, le but ici n'est pas de retrouver la clé privée d mais de passer le mécanisme de vérification de signatures en perturbant le module public N. L'attaque de J.-P. Seifert a ensuite été généralisée par J. Muir [Mui06].

Bien que l'attaque précédente ait révélé que la perturbation d'éléments publics pouvait être compromettante, il a fallu attendre l'attaque de E. Brier *et al.* [BCMCC06] pour véritablement statuer sur la nécessité de protéger aussi les éléments publics de RSA. En effet, cette attaque permet d'extraire l'exposant privé d, et donc d'usurper l'identité du signataire, en analysant des messages signés avec des modules publics perturbés. Cette attaque est d'autant plus efficace que, dans un de ses modes, il n'est pas nécessaire de choisir un modèle de perturbation pour réaliser l'analyse. La section suivante décrit le principe de ces deux attaques sur les éléments publics de RSA et rappelle leur performances.

4.1.1 Attaque du mécanisme de signature RSA

L'attaque de J.-P. Seifert sur la signature RSA [Sei05] est la première attaque par perturbation des paramètres publics de RSA. Le but consiste à corrompre le mécanisme de vérification de signature, en perturbant le module public N, de telle sorte qu'il accepte une signature falsifiée par un attaquant. Cette attaque se déroule en deux phases distinctes.

La première phase est *"hors-ligne"*, car elle ne nécessite pas d'interagir avec le mécanisme de vérification. Elle consiste à choisir un modèle de faute reproductible pour produire des modules fautés $\hat{N}$ probables, et calculer les fausses signatures correspondantes. Concrètement, l'attaquant génère des modules fautés $\hat{N}$, en suivant le modèle de faute choisi, et de telle sorte que e soit premier avec $\varphi(\hat{N})$ [20]. Aussi, l'attaquant doit modéliser précisément le type de faute qu'il devra reproduire juste avant la vérification de signature. J.-P. Seifert impose que $\hat{N}$ soit premier pour calculer facilement $\hat{d} \equiv e^{-1} \bmod \varphi(\hat{N})$. J. Muir généralisera l'algorithme en relaxant cette condition à toutes les valeurs de $\hat{N}$ friables [Mui06]. Une fois $\hat{d}$ calculé, l'attaquant signe un message m et garde en mémoire le couple message/signature perturbée $(m, \hat{S})$.

Dans la deuxième phase en *"en-ligne"*, l'attaquant donne le couple $(m, \hat{S})$ en entrée du mécanisme de vérification de signature. Il tente alors de modifier la valeur du module N au moment de son chargement, de manière à ce que tous les calculs soient réalisés avec le module perturbé. La faute produite devra correspondre au modèle de faute choisi, de telle sorte que la valeur de N perturbée soit égale à la valeur de $\hat{N}$ calculée préalablement. Si tel est le cas, la fausse signature donnée en entrée par l'attaquant sera validée par le mécanisme de vérification.

La probabilité de succès de cette attaque dépend de la précision du modèle de faute choisi ainsi que de la capacité de l'attaquant à injecter une faute ayant les effets attendus. Les résultats de l'implantation de cette attaque, ainsi qu'une optimisation, sont proposés dans [Mui06]. La version optimisée appliquée à un RSA de taille 1024 bits permet à un attaquant de forcer l'acceptation d'une signature falsifiée avec une probabilité supérieure à 50%, en modifiant aléatoirement 4 bits du module.

20. Condition d'inversibilité de e dans le groupe multiplicatif $\left(\mathbb{Z}/\varphi(\hat{N})\mathbb{Z}\right)^*$

4.1.2 Attaque de Brier *et al.*

Cette attaque, directement inspirée de l'attaque de J.-P. Seifert, propose d'étendre les précédents résultats à l'extraction de l'exposant privé d. Cette fois, l'idée est d'analyser des signatures dont le module a été modifié pour extraire des morceaux d'exposant privé en calculant des logarithmes discrets de taille réduite (*i.e.* de l'ordre de 20 à 30 bits). Ces morceaux sont ensuite recombinés par le Théorème des Restes Chinois pour reconstruire l'exposant d. Cette attaque est applicable seulement dans le cadre de signatures RSA avec remplissage déterministe ou aléa joint. Les schémas de signature basés sur une récupération d'aléa (*i.e.* RSA-PSS [BR96]) ou sur une randomisation d'exposant (*cf.* Section 5.2) ne paraissent pas être vulnérables. Mais, la preuve de l'immunité de ces implantations demeure une question ouverte.

Principe général. Comme pour l'attaque de J.-P. Seifert, cette attaque se décline en deux phases distinctes. La première est *"en ligne"* car elle consiste à collecter K couples message/signature erronée $(m_i, \hat{S}_i)_{1 \leqslant i \leqslant K}$, obtenus avec des module perturbés $\hat{N}_i \neq N$:

$$\hat{S}_i = \dot{m}_i{}^d \bmod \hat{N}_i \tag{4.1}$$

La valeur des différents modules perturbés $(\hat{N}_i)_{1 \leqslant i \leqslant K}$ est inconnue de l'attaquant. En revanche, les auteurs ont supposé que ces valeurs sont uniformément distribuées sur les entiers de taille n [BCMCC06, Cla07]. Ensuite, l'attaquant analyse les signatures erronées pour des résidus de l'exposant privé d en calculant des logarithmes discrets de petite taille. Toute cette partie d'analyse est réalisée *"hors-ligne"* et se décline en plusieurs variantes. Dans ces différentes variantes, l'attaquant peut choisir de générer ou pas, une table contenant les valeurs de modules fautés possibles. Cette table est aussi appelée *"dictionnaire de modules"*. On peut enfin noter que le nombre K de signatures erronées à collecter, durant la première phase, varie en fonction de la méthode d'analyse choisie.

Analyse sans dictionnaire. Cette méthode d'analyse correspond au cas où l'attaquant n'est pas capable d'identifier un modèle de perturbation à partir des signatures erronées, ou quand le dictionnaire des modules est trop grand. Dans ce cas, il n'est pas possible de savoir avec quel module fauté la signature a été réalisée. Afin de passer outre cette difficulté, les auteurs ont noté que pour une puissance d'un nombre premier p^a telle que $p \nmid \dot{m}_i$ et $p \nmid \hat{S}_i$, et si l'ordre multiplicatif de $\dot{m}_i$ est divisible par un petit premier r, alors le résidu de l'exposant privé à retrouver $d \mod r$ satisfait (4.2) avec une probabilité égale à 1 si $p^a \mid N_i$ et $\frac{1}{r}$ dans le cas contraire.

$$\hat{S}_i \equiv \dot{m}_i{}^{d \mod \varphi(p^a)} \mod p^a \tag{4.2}$$

Ainsi, lorsque $\varphi(p^a)$ est divisible par un petit premier r_k, l'attaquant peut utiliser ce biais en mettant en place une méthode de comptage permettant de déterminer des morceaux d'exposant privé $d_k = d \bmod r_k$ par résolution de petits logarithmes discrets sur les signatures erronées. Lorsque $R = \prod_k r_k$ est plus grand que N (et par conséquent que $\varphi(N)$), on applique le Théorème des Restes Chinois pour reconstruire la clé privée d. Nous laisserons le lecteur se référer à [BCMCC06] et [Cla07] pour plus de détails sur la méthode de comptage mise au point à partir de la proposition [BCMCC06, Proposition 1]. On peut enfin noter que cette méthode est avantageuse car elle ne nécessite pas de modéliser l'erreur introduite. D'après [BCMCC06, Cla07], l'implantation a montré que l'on peut retrouver un exposant privé d de 512 bits (1024 bits pour

un petit exposant public [21]) en collectant environ 25000 signatures erronées. En revanche, 60000 signatures erronées sont nécessaires pour retrouver 1024 bits d'exposant privé (2048 bits pour un petit exposant public).

Analyse avec dictionnaire. Les performances d'attaque peuvent être améliorées si l'attaquant peut choisir et valider un modèle de faute adéquat pour les perturbations réalisées lors de la phase *"en-ligne"*. À partir du modèle choisi et de N, il dresse une liste des valeurs possibles pour les modules perturbés. Cette liste est aussi appelée dictionnaire des modules. Une fois le dictionnaire généré, l'attaquant tente d'identifier les couples $(m_i, \hat{S}_i)$ dont la signature a été réalisée avec une des entrées du dictionnaire (*i.e.* un des modules fautés possibles). Chaque correspondance, ou *"touche"*, apporte une certaine quantité d'information sur l'exposant d. En terme de performances, l'utilisation d'un dictionnaire est avantageuse puisque, d'après les expériences menées dans [BCMCC06, Cla07], 28 *"touches"* et 1100 signatures erronées suffisent pour retrouver une clé RSA de 1024 bits. Ainsi, l'utilisation d'un dictionnaire permet d'améliorer l'attaque initiale d'un facteur 20. En combinant cette méthode avec une analyse statistique il est possible d'extraire la clé privée à partir d'un nombre de fautes égale aux nombres de touches (ce qui est optimal). Dans ce cas 28 signatures erronées peuvent suffire pour retrouver une clé RSA de 1024 bits.

4.1.3 Conclusion & Motivations

Cet état de l'art sur l'analyse des perturbations d'éléments publics de RSA a permis de montrer que même les éléments non-critiques devait retenir l'attention des concepteurs de solutions sécurisées. Bien que, par construction, les éléments publics ne doivent révéler aucune information privée, leur perturbation peut, quand à elle, entraîner une fuite d'information critique pouvant rendre caduque un schéma de vérification de signature [Sei05, Mui06]. Pire, elle peut mener jusqu'à l'extraction totale de l'exposant privé [BCMCC06, Cla07]. Par conséquent, ces attaques soulignent la nécessité de protéger les éléments publics contre les perturbations.

Par l'intermédiaire de nos travaux, nous avons essayé d'aller plus loin dans l'analyse de la vulnérabilité des éléments publics aux perturbations. En effet, les deux attaques précédentes exploitent des perturbations du module public intervenant *avant* le début de l'opération clé de la signature RSA : l'exponentiation modulaire. C'est pourquoi nous nous sommes intéressés aux perturbations de modules publics intervenant *pendant* l'exécution de l'exponentiation. Dans cette optique, nous avons mis en place différentes attaques visant des implantations classiques de l'exponentiation modulaire (*cf.* Section 1.3.4). Par ailleurs, nous avons tenté d'apporter un élément de réponse au problème ouvert posé dans [BCMCC06], à savoir si l'exploitation de la perturbation d'éléments publics est possible dans le cas d'une implantation protégée par la randomisation de l'exposant. Les attaques que nous avons imaginées dans le cadre de notre étude sont décrites dans les paragraphes suivants.

21. Lorsque e est petit, les auteurs utilisent la relation $e \cdot d \equiv 1 \mod \varphi N$ pour déterminer les bits de poids fort de l'exposant privé. En effet, sachant que $\varphi N \approx N$ sur les bits de poids fort, alors $d \approx \frac{1+k\cdot(N+1)}{e}$ avec $k < e$. Ainsi, si e est petit (*e.g.* $e = 2^16 + 1$), la moitié des bits de d peuvent être déduis de la relation précédente complétée par une recherche exhaustive sur k.

4.2 Attaque des implantations type "Right-To-Left"

4.2.1 Perturbation d'un module public RSA

4.2.1.1 Modèle de faute

Dans le cadre de notre analyse, nous avons choisi un modèle de faute dans lequel l'attaquant est capable de modifier, de manière transitoire, la valeur du module RSA pendant l'exponentiation modulaire d'une signature. Afin de rendre notre attaque la plus réaliste possible, nous considérerons que la perturbation infecte aléatoirement la valeur d'un octet du module. L'attaquant ne connaît alors ni la position de l'octet perturbé, ni sa nouvelle valeur. D'un point de vue mathématique, on peut modéliser cette faute de la manière suivante :

$$\hat{N} = N \oplus \varepsilon \tag{4.3}$$

où $\varepsilon = R_8 \cdot 2^{8i}, i \in [\![0; \frac{n}{8} - 1]\!]$ et R_8 est une valeur aléatoire, non nulle, sur un octet. La figure 4.1 illustre notre hypothèse de perturbation. Dans le cadre de cette étude, nous considérerons

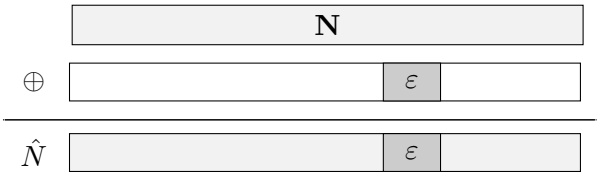


FIGURE 4.1 – Perturbation d'un octet aléatoire d'un module public RSA

que l'algorithme d'exponentiation modulaire utilisé est l'algorithme *"Right-To-Left"* (*cf.* Section 1.3.4.2). et que la première opération modifiée est un carré modulaire. Par ailleurs, la faute est injectée à la t-ième itération de l'exponentiation, de telle sorte que toute la fin de l'exponentiation soit réalisée avec un module faux. Enfin, l'instant de l'injection de faute (*i.e.* le paramètre t) est précisément connu par l'attaquant. Nous verrons plus tard comment étendre ce modèle à la perturbation d'une unique opération ou à la perturbation initiale d'une multiplication modulaire (*cf.* Section 4.2.3). Enfin, on peut noter que notre attaque peut s'appliquer quelle que soit la méthode de carré ou de multiplication modulaire implanté.

Discussion. Ce modèle de perturbation a été choisi pour la facilité avec laquelle on peut le reproduire, et notamment dans le domaine des cartes à puce [DLV03, BO06]. En effet, une telle perturbation peut être obtenue en effectuant un tir laser sur la zone de mémoire RAM contenant la valeur du module N. Ce modèle pourra être étendu à toutes les perturbations pouvant produire un nombre raisonnables d'effets sur N. En ce qui concerne le contrôle temporel de l'injection de faute par l'attaquant, cela peut-être obtenu en menant préalablement une analyse élémentaire [22] de la consommation de la puce (pour identifier la répétition de motifs) et en synchronisant un tir laser à l'itération choisie. On peut toutefois noter que notre analyse peut aussi s'appliquer si l'attaquant a une connaissance approximative de l'instant de la perturbation.

22. Traduction de *Simple Power Analysis*

4.2.1.2 Exemple d'exécution perturbée

Soit $d = \sum_{i=0}^{n-1} 2^i \cdot d_i$ l'exposant privé exprimé dans sa représentation binaire. Avec cette notation, la signature RSA d'un message m peut s'exprimer :

$$S \equiv \dot{m}^{\sum_{i=0}^{n-1} 2^i \cdot d_i} \mod N \tag{4.4}$$

Comme décrit dans le modèle de faute, nous considérons qu'une faute est injectée à la t-ième itération de l'exponentiation modulaire *"Right-To-Left"*, de telle sorte que la première opération perturbée soit un carré modulaire :

$$\hat{B} \equiv \left(\dot{m}^{2^{t-1}} \mod N\right)^2 \mod \hat{N} \tag{4.5}$$

L'algorithme d'exponentiation finit ensuite son exécution en réalisant les opérations qui suivent avec un module erroné. Ainsi, si l'on note $A \equiv \dot{m}^{\sum_{i=0}^{t-1} 2^i \cdot d_i} \mod N$ la variable d'accumulation contenant le résultat de la première partie de l'exécution, la signature erronée peut aussi s'exprimer :

$$\hat{S} \equiv ((A \cdot \hat{B})...)\hat{B}^{2^{n-t-1}} \mod \hat{N} \tag{4.6}$$

$$\equiv A \cdot \hat{B}^{\frac{d_{[t]}}{2^t}} \mod \hat{N} \tag{4.7}$$

où $d_{[t]} = \sum_{i=t}^{n-1} 2^i \cdot d_i$. L'injection de faute pendant l'exponentiation a divisé l'exécution en deux parties. En effet, alors que les premières opérations mathématiques sont réalisées avec le module RSA correct, la fin de l'exécution est réalisée avec un module erroné. Ces effets sont directement observables si l'on analyse la signature erronée. En regardant l'équation (4.7) de plus près, on remarque qu'une partie de l'exposant $d_{[t]}$ (*i.e.* les bits de poids fort) a été isolée par l'injection de faute. Dans la partie suivante, nous allons montrer comment exploiter cette observation pour extraire tous les bits d'exposant privé.

4.2.2 Analyse différentielle de la perturbation

4.2.2.1 Extraction d'un morceau de d'exposant privé

Le but de l'analyse que nous proposons ici est de retrouver la partie d'exposant isolée par l'injection d'une faute sur le module N pendant une signature. Le principe général consiste à utiliser la connaissance d'une signature correcte pour générer une signature erronée probable. Cette signature erronée probable est calculée en spéculant simultanément sur les valeurs du morceaux d'exposant privé et du module erroné. Enfin, on compare cette signature erronée probable à $\hat{S}$ pour valider nos hypothèses et obtenir une partie d'exposant privée. Aussi, la méthode d'analyse que nous proposons nécessite de connaître un couple signature correct/signature erronée obtenu à partir du même message m, connu par l'attaquant. Le choix du message n'influence pas les performances de l'attaque.

De manière plus formelle, soient $d_{[t]} = \sum_{i=t}^{n-1} 2^i \cdot d_i$ la partie d'exposant privé isolée par la faute et $\hat{N}$ le module perturbé. Si la taille de $d_{[t]}$ est *raisonnable*, c'est à dire que son nombre de bits $w = n - t$ est borné. Cette taille devra être suffisemment faible pour que l'attaquant puisse retrouver $d_{[t]}$ par une recherche exhaustive rapide. L'attaquant peut alors utiliser le modèle de faute et choisir un couple de valeurs candidates $(d'_{[t]}, \hat{N}')$ pour calculer :

$$A_{d'_{[t]}} \equiv S \cdot \dot{m}^{-d'_{[t]}} \mod N \tag{4.8}$$

Le but de ce calcul est de combiner la signature correcte à l'hypothèse sur la partie de d isolée pour retrouver la valeur de la variable d'accumulation A au moment de la perturbation du module. Ensuite, l'attaquant utilise le couple de valeurs candidates $(d'_{[t]}, \hat{N}')$ pour simuler un calcul de signature erronée, en accord avec le modèle :

$$\hat{S}_{(d'_{[t]},\hat{N}')} \equiv A_{d'_{[t]}} \cdot \left(\dot{m}^{2^{t-1}} \mod N\right)^{2 \cdot \frac{d'_{[t]}}{2^t}} \mod \hat{N}' \tag{4.9}$$

Enfin, il compare cette signature erronée probable avec celle retournée par le composant perturbé :

$$\hat{S}_{(d'_{[t]},\hat{N}')} \stackrel{?}{\equiv} \hat{S} \mod \hat{N}' \tag{4.10}$$

Si l'équation précédente est vérifiée alors le couple de valeurs candidates $(d'_{[t]}, \hat{N}')$ est très probablement égal à celui recherché (*cf.* Théorème 4.2.1). Ce résultat est d'autant plus intéressant que, dans ce cas, une seule équation permet de valider la recherche exhaustive de deux variables. D'un point de vue qualitatif, cela s'explique par le fait que la taille de l'ensemble des valeurs possibles pour le couple recherché est bien plus petite que celle de l'ensemble dans lequel l'équation (4.10) est satisfaite.

En revanche, dans le cas où l'équation n'est pas satisfaite, l'attaquant doit répéter le calcul pour un autre couple de valeurs candidates. Si aucun couple ne satisfait l'équation (4.10), alors il se peut que ce soit la multiplication modulaire qui ait été infectée initialement par la perturbation et l'attaquant pourra essayer de mener l'analyse complémentaire décrite dans la Section 4.2.3.

Théorème 4.2.1 (Probabilité de faux-acceptant)**.** *Considérons un algorithme de signature RSA standard de taille n implanté sur un composant avec la méthode "Right-To-Left". Soit $\hat{S}$ le résultat de la perturbation d'un octet aléatoire du module pendant une signature. On note $\hat{N}$ le module perturbé et $d_{[t]}$ la partie d'exposant isolée. La probabilité que l'équation (4.10) soit satisfaite pour un couple de valeurs candidates $(d'_{[t]}, \hat{N}') \neq (d_{[t]}, \hat{N})$ est négligeable pour des tailles de RSA usuelles ($n \gg 16$).*

Démonstration. On souhaite évaluer la probabilité qu'un faux couple passe le test de l'équation (4.10) soit :

$$\Pr\left[\text{Équation (4.10) est satisfaite } \mid (d'_{[t]}, \hat{N}') \neq (d_{[t]}, \hat{N})\right]$$

Pour simplifier le calcul de cette probabilité conditionnelle, nous allons évaluer séparément les évènements suivants :

- Évènement A : Équation (4.10) est satisfaite,
- Évènement B : $(d'_{[t]}, \hat{N}') \neq (d_{[t]}, \hat{N})$.

On recombinera ensuite la probabilité conditionnelle par la formule :

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}$$

Commençons par calculer la probabilité que l'évènement B se produise, en regardant la probabilité complémentaire :

$$\begin{aligned} & \Pr[(d'_{[t]}, \hat{N}') \neq (d_{[t]}, \hat{N})] \\ = \; & 1 - \Pr[(d'_{[t]} = d_{[t]}) \text{ et } (\hat{N}' = \hat{N})] \\ = \; & 1 - \Pr[d'_{[t]} = d_{[t]}] \cdot \Pr[\hat{N}' = \hat{N}] \end{aligned}$$

Or, par définition, $d'_{[t]}$ peut prendre $2^{n-t} \leqslant 2^w$ valeurs et, puisque le module doit être perturbé aléatoirement sur un octet, $\hat{N}'$ peut prendre $(2^8-1)\cdot\frac{n}{8}$ valeurs possibles. Dans les deux cas, on recherche la probabilité d'une égalité, soit :

$$\Pr[(d'_{[t]}, \hat{N}') \neq (d_{[t]}, \hat{N})] \quad = \quad 1 - \frac{8}{n\cdot(2^8-1)\cdot 2^w}$$

Maintenant, intéressons nous à l'évènement $A \cap B$. Évaluer la probabilité de cet évènement revient à calculer le nombre de fois où l'équation (4.10) est satisfaite quand on réalise le test avec un mauvais couple de valeurs candidates. En fait, cette probabilité n'est pas évidente à évaluer puisqu'elle dépend de plusieurs inconnues. Aussi, nous allons essayer de la borner suffisement finement dans un modèle aléatoire. Dans ce modèle, la probabilité que l'équation (4.10) soit vérifiée, pour des valeurs aléatoires, est $1/\hat{N}$. Dans le cas, le calcul de la borne inférieure est rapide puisque, si l'on retire le bon couple pour réaliser le test, il se peut que l'évènement ne se réalise pas. À l'opposé, l'évènement peut se réaliser pour tous les mauvais couples testés. Par conséquent, en retirant le bon couple de l'ensemble des valeurs possible, on peut borner notre probabilité par :

$$0 < \Pr[A \cap B] < min\left(\frac{\hat{N}-1}{\hat{N}}, \frac{2^w\cdot n\cdot(2^8-1)-1}{8\cdot\hat{N}}\right)$$

Par reconstruction de la probabilité conditionnelle, on obtient alors :

$$0 < \Pr[A|B] < min\left(\frac{n\cdot(\hat{N}-1)\cdot(2^8-1)\cdot 2^w}{n\cdot\hat{N}\cdot(2^8-1)\cdot 2^w - 8}, \frac{(n\cdot(2^8-1)\cdot 2^w-1)\cdot n\cdot(2^8-1)\cdot 2^w}{\hat{N}\cdot(n\cdot(2^8-1)\cdot 2^w - 8))}\right) \tag{4.11}$$

Bien que cette expression soit un peu barbare, on peut noter que la probabilité est maximisée par une value proche de 1 quand $n < 16$ bits ce qui n'est pas très intéressant. Toutefois, on peut encore noter que la probabilité décroît rapidement quand la valeur de $\hat{N}$ augmente, soit une décroissance exponentielle avec n. En conclusion, lorsque $n \gg 16$ (*i.e.* $n = 1024$ bits), cette probabilité est complètement négligeable. Ce résultat a été confirmé par notre implantation l'attaque (en utilisant la librairie GMP [23]). □

Remarque 1. *Il n'est pas nécessaire de répéter le calcul de l'équation (4.8) pour chacun des couples de valeurs candidates. En effet, ce calcul ne dépend pas du module erroné, il pourra être effectué seulement pour chaque valeur probable d'exposant* $d'_{[t]}$.

Remarque 2. *Afin de réduire le nombre de couples candidats, et ainsi réduire le nombre de tests à effectuer, on peut noter que seules les valeurs probables de module* $\hat{N}' > \hat{S}$ *conviennent. En effet, la signature erronée* $\hat{S}$ *est le reste de la division par* $\hat{N}$*, et par conséquent* $\hat{N} > \hat{S}$*. Cette simple observation peut réduire significativement le nombre de couples à tester.*

4.2.2.2 Généralisation au reste de l'exposant privé

Le paragraphe précédent décrit un moyen d'exploiter les perturbations du module public pour retrouver des bits de poids fort de l'exposant privé. Le reste des bits de d peut être obtenu progressivement en répétant l'analyse précédente sur des signatures perturbées de plus en plus tôt dans leur exécution, et en utilisant les bits de d déjà retrouvés. Comme pour les attaques précédentes sur le module RSA (*cf.* Section 4.1), la première étape consiste à collecter plusieurs

23. GNU Multiple Precision Arithmetic Library disponible à `http://gmplib.org`

signatures $(\hat{S}_k)_{1\leqslant k\leqslant K}$ dont le module a été perturbé à différentes itérations de l'exponentiation. Pour chacune des signatures $\hat{S}_k$ la partie d'exposant isolée par la faute est :

$$\begin{aligned} d_{[t_k]} &= \sum_{i=n-t_k}^{n-1} 2^i d_i \\ &= \sum_{i=n-t_k}^{n-t_{k-1}-1} 2^i d_i + \sum_{i=n-t_{k-1}}^{n-1} 2^i d_i \\ &= \sum_{i=n-t_k}^{n-t_{k-1}-1} 2^i d_i + d_{[t_{k-1}]} \end{aligned} \tag{4.12}$$

Cette relation de récurrence met en évidence la résolution progressive de l'exposant privé par l'analyse de signatures perturbées de plus en plus tôt pendant leur exécution respective. En effet, dans le cas général, la partie d'exposant isolée $d_{[t_k]}$ se divise en une partie connue $d_{[t_{k-1}]}$ (obtenue par une analyse précédente) et en $(t_k - t_{k-1})$ bits inconnus. Ces bits inconnus de $d_{[t_k]}$ sont alors retrouvés en utilisant la connaissance de $d_{[t_{k-1}]}$ et en se ramenant à l'analyse décrite précédemment. En d'autres termes, rechercher un couple $(d_{[t_k]}, \hat{N}_k)$ satisfaisant (4.10) revient alors à faire des hypothèses seulement sur la valeur du module fauté $\hat{N}_k$ et sur les $(t_k - t_{k-1})$ bits inconnus de $d_{[t_k]}$.

Comme précédemment l'attaquant défini alors une taille de fenêtre w de telle sorte que $\forall k \in [\![1; K]\!]$, $t_k - t_{k-1} \leqslant w$ avec $t_0 = 0$ et $t_K = n - 1$. Ce paramètre w est choisi par l'attaquant de manière à limiter la taille de la recherche des bits d'exposant privé pour l'analyse de chacune des signatures erronées. Par conséquent, ce paramètre a aussi une influence sur le nombre K de signature à collecter pour une résolution complète de l'exposant privé d. Enfin, l'attaquant peut aussi utiliser des méthodes algébriques classiques, comme les *pas de bébé/pas de géant*, pour retrouver les derniers bits manquants d'exposant.

4.2.2.3 Performances

Estimations théoriques. Afin de comparer les performances de notre nouvelle attaque à celles précédemment publiées, nous avons estimé théoriquement le nombre de perturbations à réaliser ainsi que le temps nécessaire à l'extraction de l'exposant privé. Le résultat de cette estimation est résumé dans le théorème 4.2.2.

Théorème 4.2.2. *Considérons un algorithme de signature RSA standard de taille n implanté sur un composant avec la méthode "Right-To-Left". Alors il est possible de retrouver l'intégralité de l'exposant privé d, par fenêtres de w bits, en perturbant aléatoirement un octet du module de* $\mathcal{O}(n/w)$ *signatures et au prix de* $\mathcal{O}\left(2^w \cdot n^2/w\right)$ *exponentiations modulaires.*

Démonstration. En ce qui concerne le nombre de signatures perturbées à récolter, ce résultat est évident. En effet, puisque l'analyse d'une signature peut permettre d'extraire w bits d'un exposant privé de taille n bits, il faut donc analyser au moins n/w signatures erronées selon le modèle pour trouver l'intégralité de l'exposant. En revanche il faut regarder l'attaque plus en détail pour déterminer la complexité calculatoire. Commençons par détailler le coût d'une seule analyse. Pour identifier un couple de valeurs candidates qui vérifie l'équation (4.10), dans le pire des cas, il faut tester tous les couples possibles. Or, la première valeur est une partie d'exposant de w bits et la seconde est un module fauté selon le modèle de faute choisi (modification aléatoire d'un octet de N). Par conséquent le nombre de couples candidats est : $2^w \cdot 255 \cdot \frac{n}{8} \approx 2^{5+w} \cdot n$.

Or, un test de couple nécessite 2 exponentiations (*cf.* Section 4.2.2). Par ailleurs, il faut analyser les n/w signatures erronées pour obtenir entièrement l'exposant. Aussi, la complexité globale de l'attaque est donc de $2 \cdot n/w \cdot 2^{5+w} \cdot n = \mathcal{O}\left(2^w \cdot n^2/w\right)$ exponentiations modulaires. □

À titre de comparaison, l'attaque de Bellcore [BDL97, BDL01] nécessite de réaliser $\mathcal{O}(n^3 \cdot 2^w \cdot \log^2(n)/w^2)$ exponentiations modulaires ce qui est plus complexe. Par ailleurs, pour un RSA de 1024 bits et une taille de fenêtre $w \leq 20$, notre attaque présente des performances sensiblement comparable à celles de Brier *et al.* [BCMCC06] (si l'algorithme *pas de bébé/pas de géant* est utilisé pour calculer les petits logarithmes discret). Enfin, notre attaque est aussi compétitive en terme de nombre de fautes puisque l'attaque de Bellcore [BDL97, BDL01] requiert l'analyse de $\mathcal{O}((n/w) \cdot \log(2n))$ résultats erronées. En revanche, nous n'avons pas pu atteindre, dans nos simulations, les performances de l'attaque de Brier *et al.* [BCMCC06] en termes de nombres de fautes (*i.e.* 28 fautes pour un RSA 1024 bits pour l'analyse statistique avec dictionnaire). Toutefois, ces performances soulignent le caractère réaliste de notre attaque.

Résultats expérimentaux. Afin de valider nos résultats, nous avons implanté cette attaque sur un PC Linux (Red Hat Scientifique et `gcc 4.1.2`) embarquant un processeur Intel Core 2 Duo cadencé à 2.66 GHz. Le crypto-système RSA a été implanté en utilisant la librairie GMP. Nous avons généré les signatures erronées en simulant des perturbations aléatoires d'un octet du module, pendant le calcul de signature. Dans ce cas précis et en choisissant $w = 4$ bits, il suffit de 257 signatures erronées et de 40 minutes d'analyse pour retrouver une clé privée RSA de 1024 bits. Ce temps d'analyse tient compte de la distribution de la recherche sur chacun des deux cœurs du PC et de quelques optimisations. Cependant, ces performances pourraient être améliorées en utilisant la méthode *pas-de-bébé/pas-de-géant* pour retrouver les derniers bits manquants de d.

4.2.3 Extension du modèle de faute

Afin d'illustrer notre attaque, nous avons proposé d'étudier le cas où la première opération infectée par la faute est un carré modulaire. Or, nous montrons dans ce paragraphe qu'il est possible d'analyser efficacement une signature erronée même si c'est une multiplication modulaire qui a été perturbée en premier. Par ailleurs, nous montrons qu'il en va de même quand la faute n'infecte qu'une seule opération pendant la signature. Cette étude supplémentaire permet de relâcher des contraintes sur le modèle de faute et de généraliser un peu plus notre attaque sur le module public N.

4.2.3.1 Perturbation initiale d'une multiplication

Dans ce paragraphe, nous nous plaçons dans le cadre du modèle de faute que nous avons choisi (*cf.* Section 4.2.1.1) à la différence près que nous allons, cette fois, traiter le cas où la première opération infectée est une multiplication modulaire. En se remémorant le principe de l'exponentiation *"Right-To-Left"*, on peut déjà remarquer que ce cas n'est possible que si $d_{t-1} = 1$. En effet, l'exécution d'une multiplication modulaire dépend de la valeur courante du bit d'exposant. Dans ce cas, juste après la perturbation du module N, la variable d'accumulation A peut être exprimée ainsi :

$$\hat{A} \equiv (\dot{m}^{\sum_{i=0}^{t-2} 2^i \cdot d_i} \mod N) \cdot B \mod \hat{N} \tag{4.13}$$

$$\equiv \left[(\dot{m}^{\sum_{i=0}^{t-2} 2^i \cdot d_i} \mod N) \cdot (\dot{m}^{2^{t-1}} \mod N)\right] \mod \hat{N} \tag{4.14}$$

Ensuite, quelle que soit la valeur de l'exposant, l'exécution se poursuit avec un carré modulaire :

$$\hat{B} \equiv (\dot{m}^{2^{t-1}} \mod N)^2 \mod \hat{N} \tag{4.15}$$

Enfin la signature erronée peut être exprimée de la manière suivante :

$$\begin{aligned} \hat{S} &\equiv ((\hat{A} \cdot \hat{B})...) \cdot \hat{B}^{2^{n-t-1}} \mod \hat{N} && (4.16)\\ &\equiv \hat{A} \cdot \hat{B}^{\sum_{i=t}^{n-1} 2^{i-t} \cdot d_i} \mod \hat{N} && (4.17)\\ &\equiv [(\dot{m}^{\sum_{i=0}^{t-1} 2^i \cdot d_i} \mod N) \cdot (\dot{m}^{2^{t-1}} \mod N)^{\frac{d_{[t-1]}}{2^{t-1}}}] \mod \hat{N} && (4.18) \end{aligned}$$

où $d_{[t-1]} = \sum_{i=t-1}^{n-1} 2^i \cdot d_i$. D'après l'équation précédente, on peut noter que l'injection de la faute permet d'isoler cette fois $(n-1)-(t-1)+1 = n-t+1$ bits d'exposant. Mais, en sachant que la première opération perturbée est une multiplication, on peut déduire que $d_{t-1} = 1$. Par conséquent, seuls $n-t$ bits d'exposant privé restent inconnus et il suffit de légèrement adapter l'analyse décrite précédemment pour les retrouver avec une forte probabilité.

4.2.3.2 Perturbation d'une seule opération

Maintenant que nous savons comment exploiter des perturbations infectant initialement un carré ou une multiplication, intéressons nous au cas où une seule opération est infectée. Une telle perturbation transitoire peut être obtenue en perturbant la lecture en mémoire d'un octet du module, au moment de réaliser l'opération modulaire adéquate. Puisque la valeur de N stockée en mémoire n'a pas été modifié, les opérations suivantes de l'exponentiation seront réalisées avec la valeur correcte du module public. Comme précédemment, nous considérerons une perturbation aléatoire d'un octet de N. Cette perturbation peut intervenir indépendemment avant un carré ou une multiplication modulaire à la t-ième itération de l'exécution. Mais, puisque la multiplication modulaire n'est pas toujours exécutée, il faudra scinder l'analyse en deux cas, suivant la valeur du bit courant d'exposant d_{t-1} :

1. $d_{t-1} = 0$ ou 1 et le carré modulaire est perturbé. Dans ce cas, c'est la variable d'accumulation B qui est modifiée en premier : $\hat{B} \equiv (\dot{m}^{2^{t-1}} \mod N)^2 \mod \hat{N}$. Ensuite, la faute se propage ainsi dans l'exécution :

$$\begin{aligned} \hat{S} &\equiv ((A \cdot \hat{B})...) \cdot \hat{B}^{2^{n-t-1}} \mod N && (4.19)\\ &\equiv A \cdot \hat{B}^{\sum_{i=t}^{n-1} 2^{i-t} \cdot d_i} \mod N && (4.20)\\ &\equiv (\dot{m}^{\sum_{i=0}^{t-1} 2^i \cdot d_i} \mod N) && (4.21)\\ &\quad \cdot [(\dot{m}^{2^{t-1}} \mod N)^2 \mod \hat{N}]^{\frac{d_{[t]}}{2^t}} \mod N \end{aligned}$$

   Comme on peut le remarquer, la perturbation d'une seule opération suffit à isoler une partie d'exposant privé. Cet effet est obtenu par la réalisation d'un carré dans un groupe différent des autres opération (*i.e.* $\mathbb{Z}/\hat{N}\mathbb{Z}$ au lieu de $\mathbb{Z}/N\mathbb{Z}$). Pour extraire les bits de d isolés, il suffit de modifier légèrement la reconstruction d'une signature fautée possible et de vérifier que l'équation (4.10) est satisfaite modulo N.

2. $d_{t-1} = 1$ et la multiplication est perturbée. Contrairement au cas précédent, cette situation n'est possible que si $d_{t-1} = 1$. Ici, c'est la variable d'accumulation A qui est infectée en premier :

$$\hat{A} \equiv [(\dot{m}^{\sum_{i=0}^{t-2} 2^i \cdot d_i} \mod N) \cdot (\dot{m}^{2^{t-1}} \mod N)] \mod \hat{N} \tag{4.22}$$

Dans ce modèle, toutes les opérations suivantes sont réalisées avec le bon module. Ainsi, $B \equiv \dot{m}^{2^t} \mod N$ mais l'erreur se propage quand même de la manière suivante :

$$\hat{S} \equiv ((\hat{A} \cdot B)...)B^{2^{n-t-1}} \mod N \quad (4.23)$$

$$\equiv \hat{A} \cdot B^{\sum_{i=t}^{n-1} 2^{i-t} \cdot d_i} \mod N \quad (4.24)$$

$$\equiv \hat{A} \cdot B^{\frac{d_{[t]}}{2^t}} \mod N \quad (4.25)$$

Une fois encore, on remarque que $n - t$ bits de d sont isolés par la perturbation. Par conséquent, cette signature erronée peut encore être analysée en adaptant la méthode décrite précédemment (*cf.* Section 4.2.2)

Par le biais de cette analyse complémentaire, nous avons montré que notre attaque par perturbation ne se limite pas à un modèle de faute unique. Afin de rendre cette étude aussi générale que possible, nous avons étudié les cas où une faute sur le module public vient perturber une ou plusieurs opérations pendant l'exécution d'une signature. Enfin, notre attaque par perturbation pourra être avantageusement panachée avec d'autres méthodes d'analyse (analyse algébrique, attaques par canaux auxiliaires) pour atteindre de meilleurs performances.

4.3 Attaque des implantations type "Left-To-Right"

4.3.1 Limitation de l'analyse R2L

Bien que les algorithmes d'exponentiation binaire *"Right-To-Left"* et *"Left-To-Right"* présentent une structure similaire (*cf.* Section 1.3.4), leur résistance vis à vis des attaques physiques peut diverger. Cette dissymétrie a déjà été constatée par P.-A. Fouque et F. Valette avec leur *"Doubling Attack"* [FV03]. En effet, cette attaque par analyse élémentaire permet de retrouver un exposant privé RSA masqué seulement si l'implantation de l'exponentiation est de type *"Left-To-Right"*. Par conséquent, nous avons poussé notre étude pour savoir si seules les implantations de type *"Right-To-Left"* sont vulnérables à notre attaque par perturbation du module public. La partie suivante présente les difficultés que nous avons rencontrées pour étendre l'attaque aux implantations *"Left-To-Right"* ainsi que les solutions que nous avons choisies pour les contourner.

Application Naïve. Dans un premier temps, nous avons essayé de porter naïvement notre attaque sur une implantation de l'exponentiation binaire de type *"Left-To-Right"*. Pour ce faire, nous avons utilisé le modèle de faute décrit précédemment (*cf.* Section 4.2.1.1), à savoir une modification aléatoire de la valeur d'un octet du module N à t itérations de la fin de l'exponentiation. Soit A la variable d'accumulation juste avant l'injection de la perturbation du module N :

$$A \equiv \dot{m}^{\sum_{i=t}^{n-1} 2^{i-t} \cdot d_i} \mod N \quad (4.26)$$

En considérant que la première opération perturbée est un carré modulaire, la signature erronée $\hat{S}$ peut être exprimée de la manière suivante :

$$\hat{S} \equiv \left(\left(\left(A^2 \cdot \dot{m}^{d_{t-1}}\right)^2 \cdot \dot{m}^{d_{t-2}}\right)^2 \dots\right)^2 \cdot \dot{m}^{d_0} \mod \hat{N} \quad (4.27)$$

$$\equiv A^{2^t} \cdot \dot{m}^{d_{[t]}} \mod \hat{N} \quad (4.28)$$

Contrairement à l'attaque contre l'implantation "Right-To-Left", nous noterons ici par $d_{[t]} = \sum_{i=0}^{t-1} 2^i \cdot d_i$, les t bits de poids faible de d isolés par la perturbation du module public. En

étudiant l'équation précédente (4.28), on peut faire deux remarques importantes. Premièrement, l'injection de la faute permet toujours d'isoler t bits de l'exposant privé. Mais la perturbation provoque un effet non désiré. En effet, on remarque que la variable d'accumulation A subit une série de carrés modulaires, t exactement, pendant l'exécution de la signature. Cela signifie que pour extraire la partie d'exposant isolé avec la méthode décrite pour l'algorithme *"Right-To-Left"*, il faut pouvoir inverser ces carrés modulaires. Or, le calcul de racines carrés modulaires quand le module est composite, est un problème aussi difficile que la factorisation. Aussi, ce calcul de racines carrées est d'autant plus difficile dans un groupe RSA, où le module N est impossible à factoriser en un temps raisonnable (*i.e.* N est composite car produit de deux grands nombres premiers p et q). Mais, en regardant plus attentivement l'équation (4.28), on se rend compte que la variable A est mise au carré modulo $\hat{N}$ et non modulo N. Or, si $\hat{N}$ est premier ou a une factorisation évidente, le calcul de racine carré modulaire se réalise en temps polynomial. Par conséquent, si l'on souhaite analyser les signatures erronées, il faut que les modules perturbés soit premiers (ou de factorisation évidente) mais il faut aussi que le calcul de racines carrées modulaires ne soit pas trop coûteux.

4.3.2 Problème des racines carrées modulaires

Dans cette partie, nous détaillerons l'algorithme que nous avons choisi pour implanter le calcul de racines carrées modulaires. Nous montrerons ensuite, par une étude statistique détaillée, que le fait de considérer seulement les signatures obtenues avec un module fauté premier est une hypothèse réaliste.

4.3.2.1 Algorithme de Tonelli & Shanks

L'algorithme de Tonelli & Shanks [Coh93] est une méthode probabiliste mais relativement efficace pour calculer des racines carrées dans un groupe muni d'une multiplication $(\mathbb{Z}/P\mathbb{Z})^*$, où P est un nombre premier.À partir de cet algorithme, le calcul de racines carrées modulaires peut être généralisé à tout groupe cyclique. Pour ce faire commencer par identifier l'existence d'un isomorphisme entre le groupe multiplicatif $(\mathbb{Z}/P\mathbb{Z})^*$ et le groupe additif $\mathbb{Z}/(P-1)\mathbb{Z}$. En effet, supposons que $P-1$ puisse aussi s'écrire :

$$P - 1 = 2^e \cdot r, \qquad \text{avec } r \text{ impair.} \tag{4.29}$$

Alors, le groupe cyclique G d'ordre 2^e est un sous-groupe de $\mathbb{Z}/(P-1)\mathbb{Z}$. Soit z un générateur de G, si a est un *résidu quadratique*[24] modulo P, alors :

$$a^{(P-1)/2} \equiv (a^r)^{2^{e-1}} \equiv 1 \bmod P \tag{4.30}$$

On peut alors noter que $a^r \bmod P$ est aussi un résidu quadratique dans G, soit $\exists k \in [\![0 : 2^e - 1]\!]$ de telle sorte que :

$$a^r \cdot z^k = 1 \text{ dans le groupe } G \tag{4.31}$$

Ou encore, $a^{r+1} \cdot z^k = a$ dans le groupe G. Par conséquent, la racine carrée modulaire de a modulo P peut s'exprimer ainsi :

$$a^{1/2} \equiv a^{(r+1)/2} \cdot z^{k/2} \bmod P \tag{4.32}$$

Les principales étapes de cet algorithme de calcul de racines carrées modulaires sont donc :

24. C'est à dire $\exists x \in (\mathbb{Z}/P\mathbb{Z})^*$ tel que $a \equiv x^2 \bmod P$

– Trouver un générateur z du sous-groupe multiplication G,
– Calculer l'exposant k.

Afin de nous convaincre de l'efficacité de cet algorithme pour calculer des racines carrées modulaires, nous l'avons implanté sur PC et mesuré ses performances. Ces résultats sont résumés dans la partie suivante.

Implantation et Résultats. À partir du principe du calcul de racines carrées modulaires proposé par Tonelli & Shanks nous avons implanté l'algorithme correspondant sur un PC [Coh93]. En pratique, la partie la plus coûteuse de l'algorithme correspond à la recherche de l'exposant k. Par conséquent, la complexité calculatoire de l'algorithme est dominée par la complexité de cette recherche :

$$\mathcal{C}_{Tonelli\ \&\ Shanks} = \mathcal{O}\left(\ln^4 P\right) \text{ opérations binaires} \tag{4.33}$$
$$= \mathcal{O}\left(\ln P\right) \text{ exponentiations} \tag{4.34}$$

Pour mieux se rendre compte, sur un Pentium IV 3.2GHz, notre implantation GIVARO [25] de l'algorithme nécessitait en moyenne 7/1000 de secondes pour trouver une racine carrée modulaire, pour des groupes de taille 1024 bits. Par conséquent, les performances réelles de l'implantation de cet algorithme ont retenu notre attention, c'est pourquoi nous l'avons choisi pour réaliser l'exploitation de signatures erronées.

Remarque 3. *Comme dans [Mui06], la contrainte sur la primalité du module perturbé peut être étendue au cas où celui ci est nombre friable. En d'autres termes, on peut aussi considérer les cas où le module fauté est factorisable par de "petits" nombres premiers. En effet, si la factorisation du module est connue, il est possible de calculer facilement des racines carrées modulaires. Dans ce cas, le principe consiste à chercher les racines carrées modulo chacun des facteurs de $\hat{N}$ puis à combiner les racines obtenues grâce au Théorème des Restes Chinois (cf. [Sho05, §13.3.3] pour une explication plus détaillée). Dans un souci de clarté, nous considérerons seulement les cas où les modules perturbés sont premiers pour calculer les racines carrées.*

4.3.2.2 Analyse statistique du nouveau modèle de faute

Dans le paragraphe précédent, nous avons retenu une méthode pour calculer des racines carrées modulaires. Or, dans le cadre de notre analyse de perturbation, ce calcul de racines carrées n'est possible que si le modulo fauté $\hat{N}$ est un nombre premier. Il est alors légitime de se demander si cette condition n'est pas rédhibitoire pour que l'attaque soit applicable en pratique. Afin de répondre à cette interrogation, nous proposons ici une étude statistique de ce nouveau modèle de faute.

Pour commencer cette étude, nous proposons d'estimer le nombre de premiers pour une taille en bits fixée. D'après [Dus98, Théorème 1.10], le nombre de premiers π inférieurs à un entier x peut être borné par :

$$\pi(x) \geq \frac{x}{\ln(x)}\left(1+\frac{1}{\ln(x)}+\frac{1.8}{\ln^2(x)}\right), \text{ pour } x \geq 32299. \tag{4.35}$$
$$\pi(x) \leq \frac{x}{\ln(x)}\left(1+\frac{1}{\ln(x)}+\frac{2.51}{\ln^2(x)}\right), \text{ pour } x \geq 355991.$$

25. GIVARO est une librairie C++ open source pour l'arithmétique et le calcul algébrique. Cette librairie, basée sur GMP, est disponible à l'adresse `http://packages.debian.org/fr/sid/libgivaro-dev`

Par conséquent, pour des nombres dont la taille est exactement τ bits tel que $\tau \geq 19$ bits, le nombre de premiers est :

$$\pi_\tau = \pi(2^\tau) - \pi(2^{\tau-1}) \tag{4.36}$$

En utilisant l'expression des bornes détaillées par l'équation (4.35), on peut approximer finement la probabilité qu'un nombre de τ bits choisi aléatoirement soit premier $\frac{\pi_\tau}{2^{\tau-1}}$:

$$\begin{aligned} \frac{\pi_\tau}{2^{\tau-1}} > \quad Inf(t) &= \frac{0.480t^5 - 1.229t^4 + 0.0265t^3 - 7.602t^2 + 9.414t - 3.600}{t^3(t-1)^3\ln^3(2)} \\ \frac{\pi_\tau}{2^{\tau-1}} < \quad Sup(t) &= \frac{0.480t^5 - 1.229t^4 + 2.157t^3 - 11.862t^2 + 13.674t - 5.02}{t^3(t-1)^3\ln^3(2)} \end{aligned} \tag{4.37}$$

Ainsi, pour des modules de taille $\tau = 1024$ bits :

$$Inf(1024) = \frac{1}{709.477} \text{ and } Sup(1024) = \frac{1}{709.474}$$

Par conséquent, cela signifie que si l'on considère des nombres de 1024 bits, alors ils ont une chance sur 709 d'être premier. De la même manière, si l'on tire aléatoirement un nombre de 2048 bits, ses chances d'être premier sont d'un peu plus d'un sur 1419. Cependant, cette densité est évaluée pour des nombres pouvant être indépendamment pairs ou impairs. Or, on sait qu'un module RSA résulte du produit de deux grands nombres premiers (et donc, au moins impairs). On peut donc affirmer que N est toujours impair. Par conséquent, la probabilité qu'un module fauté RSA soit premier est :

$$\Pr\left[\hat{N} \text{ premier} \mid N \text{ impair}\right] = pr_\tau \tag{4.38}$$

$$= 2 \times \frac{\pi_\tau}{2^{\tau-1}} \tag{4.39}$$

$$= \frac{2}{709} \tag{4.40}$$

Ainsi, en partant d'un module RSA de 1024 bits les chances de tomber sur un module perturbé premier sont d'un sur 305. Pour un RSA de 2048, elles sont donc d'un sur 709. Maintenant que nous avons réussi à évaluer finement la densité de nombres premiers pour des tailles usuelles de RSA, nous allons nous intéresser plus particulièrement à notre modèle de faute.

Pour rappeler rapidement le contexte de notre étude, nous considérons des perturbations venant affecter un octet aléatoire d'un module RSA N, pendant l'exécution d'une signature implanté à la manière *"Left-To-Right"*. Pour que les signatures erronées puissent être exploitables, il faut que le module perturbé $\hat{N}$ soit premier. Afin de montrer que ce modèle soit réaliste, nous voulons évaluer, de manière théorique, la probabilité que cet évènement se réalise. Pour ce faire, considérons, dans un premier temps, un ensemble de k nombres aléatoires de taille τ bits exactement et soit P_N la variable aléatoire représentant le nombre de premiers que l'on peut s'attendre à trouver dans l'ensemble considéré. On peut alors affirmer que la variable discrète P_N suit une loi binomiale $\mathcal{B}(k, pr_\tau)$. Nous pouvons alors calculer un intervalle de confiance pour le dénombrement de premier :

$$\Pr[a \leqslant P_N \leqslant b] = \sum_{i=a}^{b} \binom{k}{i} {pr_\tau}^i \cdot (1 - pr_\tau)^{k-i} \tag{4.41}$$

où a et b sont des bornes entières pour la variable aléatoire P_N. Maintenant que nous avons posé toutes les briques nécessaires à l'étude, plaçons nous dans le cas qui nous intéresse. Soit $\mathcal{N}$

l'ensemble des valeurs obtenues en considérant toutes les modifications possibles d'un module RSA N. En d'autres termes, si l'on note par $\oplus$ le "ou exclusif" binaire, alors [1] :

$$\mathcal{N} = \{N \oplus R_8 \cdot 2^{8i},\ R_8 = 0 \,..\, 255,\ i = 0 \,..\, (\frac{n}{8} - 1)\} \tag{4.42}$$

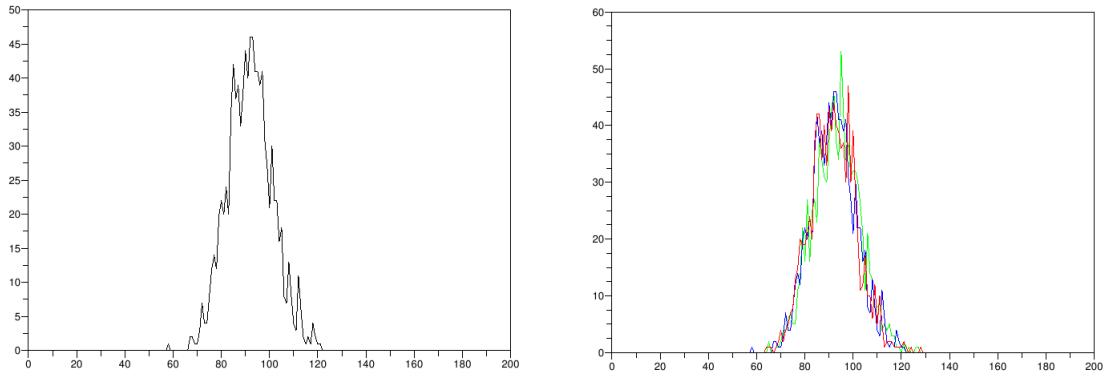
La cardinalité de cet ensemble $\mathcal{N}$ est donc :

$$|\mathcal{N}| = 256 \cdot \frac{n}{8} = 32 \cdot n \tag{4.43}$$

Si l'on considère que $\mathcal{N}$ est composé de valeurs tirées aléatoirement, alors sa proportion de valeurs premières correspond à ce qui est exprimé par l'équation (4.40). Dans ce cas, on peut fixer $k = |\mathcal{N}|$ et calculer le nombre moyen de premiers en approximant pr_τ. Dans le cas usuel d'un RSA de taille $n = \tau = 1024$ bits, alors on peut calculer une valeur approximative pour pr_{1024} et ensuite déduire le nombre moyen de modules perturbés suivant le modèle de faute dont la valeur est première : $k \cdot pr_\tau$. Dans notre cas, ce nombre moyen vaut $32 \cdot 1024 \cdot 2/709 = 92$. Par ailleurs, on peut aussi combiner l'estimation de pr_{1024} à l'équation (4.41), pour estimer que, dans plus de 99.99% des cas, le nombre de modules fautés premiers que l'on peut espérer dénombrer est au minimum de 56 et au maximum de 129. Il est intéressant de noter que pour un RSA de taille 2048 bits, la répartition est quasi similaire. En effet, parmi tous les modules perturbés obtenus suivant le modèle, environ 92 seront premiers en moyenne et ce nombre varie entre 56 et 129 avec une probabilité supérieure à 99.99%. En pratique, $\mathcal{N}$ n'est évidemment pas composé d'éléments aléatoires. Cependant les résultats que nous avons obtenus empiriquement démontrent que le biais engendré par cette hypothèse est négligeable.

Résultats empiriques. Afin de valider notre analyse théorique, nous avons mené une étude statistique complémentaire. Nous avons généré aléatoirement des modules RSA puis nous avons utilisé notre modèle de perturbation pour dériver toutes les valeurs de modules perturbés possibles. Enfin, nous avons recensé, dans cet ensemble, tous les modules perturbés premiers. Les résultats obtenus sont illustrés par la figure 4.2. La figure 4.2 (a) illustre la répartition du nombre de premiers que l'on peut espérer trouver parmis tous les modules perturbés aléatoirement sur un octet, dérivés de modules RSA de taille 1024. Bien que nous ayons pris en compte le modèle de perturbation dans notre expérience, la répartition semble suivre une loi binomiale, comme ce que l'on prédisait dans notre étude théorique (*i.e.* en considérant des modules perturbés aléatoires). En guise de comparaison, nous avons réalisé une étude similaire en remplaçant le module RSA par d'un nombre premier et un nombre impair (*cf.* Figure 4.2 (b)). Ces courbes sont le fruit d'une étude complémentaire de notre article de CT-RSA 2009 [BCDG09] menée en stage par des élèves en 2ème année de l'ENSIMAG. Ces courbes mettent en évidence la faible dépendance existant entre la nature du nombre de base et la probabilité que les nombres fautés dérivés soient premiers. En observant la table 4.1 résumant tous nos résultats, on peut noter que nous ne nous sommes jamais trouvés dans le cas où aucun nombre premier ne figurait parmi les ensembles de modules fautés. Les valeurs extrémales du nombre de premiers recensés, dans le cas de perturbation d'un octet, correspondent quasi parfaitement à ce que l'on a obtenu par l'analyse théorique. Par conséquent, ces résultats empiriques viennent confirmer nos estimations théoriques et clore notre étude du modèle de faute.

1. Dans un souci de simplification, nous supposons qu'un octet perturbé peut prendre 2^8 valeurs. En réalité, il ne peut en prendre que $2^8 - 1$ car l'erreur ne peut être nulle, sinon le module resterait inchangé et la signature serait inexploitable.



(a) Nombre de premiers à une distance de Hamming de 8 bits d'un module RSA

(b) Nombre de premiers à une distance de Hamming de 8 bits d'un entier impair (vert)/ module RSA (bleu)/ nombre premier (rouge)

FIGURE 4.2 – Distribution expérimentale du nombre de premiers parmi des nombres de 1024 bits de nature différente

Remarque 4. *Bien que la figure 4.2 et la table 4.1 présentent les résultats obtenus de manière globale, nous avons, en pratique, étudié encore plus finement la répartition des nombre premiers dans l'ensemble des valeurs possibles de modules fautés. Pour un module donné, nous avons remarqué que la répartition du nombre de premiers n'était pas uniforme suivant les octets perturbés. Cela signifie qu'en focalisant la perturbation sur un octet spécifique du module, un attaquant peut considérablement augmenter ses chances de tomber sur un module fauté premier (i.e. jusque d'un facteur* 5 *suivant l'octet étudié). Cette remarque peut donc permettre à un attaquant de réduire le nombre de signatures qu'il devra perturber pour espérer en obtenir une exploitable. En revanche, la position de ces octets "faibles" varie suivant le module étudié. Par conséquent, l'attaquant devra faire une analyse statistique préalable du module pour identifier ses octets "faibles" et ainsi augmenter ses chances d'obtenir un module fauté premier.*

TABLE 4.1 – Estimation empirique du nombre de premiers dans $\mathcal{N}$.

Architecture	n bits	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{N}\rvert \cdot pr_n$	# essais.	# de premiers		
					Min.	Moyen.	Max.
8-bit	1024	2^{15}	92.37	300	58	91.78	121
8-bit	2048	2^{16}	92.35	200	63	91.12	112
16-bit	1024	2^{22}	$11823,66$	100	11282	$11838,16$	12424
32-bit	1024	2^{37}	$\approx 3,88 \cdot 10^8$				

Conséquences de l'étude. Bien qu'a priori, considérer seulement des modifications primales d'un module RSA semble rédhibitoire, nos estimations théoriques confirmées par nos expérimentations montrent que ces cas ne sont pas si rares. En effet, pour un RSA de 1024 bits, il suffit de collecter 305 signatures perturbées suivant le modèle pour espérer en obtenir une exploitable. De plus ce nombre peut être considérablement réduit si l'attaquant dispose d'un matériel lui permettant de choisir assez précisément un octet du module à perturber. Par ailleurs, notre

étude vient renforcer les résultats déjà obtenus par J.-P. Seifert's [Sei05, Mui06] avec son attaque visant à corrompre le mécanisme de vérification de signatures RSA (*cf.* Section 4.1). Par conséquent, dans la suite de notre analyse de perturbations du module public pour des implantations *"Left-To-Right"* de la signature RSA, nous considérerons qu'il est possible d'obtenir des modules fautés premiers. Nous prendrons en compte la probabilité associée à cet évènement pour évaluer les performances de notre attaque.

Par l'intermédiaire de l'étude précédente, nous avons trouvé une solution pour résoudre le problème de calcul de racines carrées modulo un module fauté $\hat{N}$. Notre implantation de l'algorithme de Tonelli & Shanks ainsi que notre étude statistique montrent respectivement que cette solution est aussi réaliste en termes de complexité calculatoire qu'en termes de nombres de perturbations.

4.3.3 Analyse différentielle des perturbations

4.3.3.1 Extraction d'un morceau d'exposant

Maintenant que nous avons proposé une solution réaliste pour calculer des racines carrées modulaires, il est possible de mener une analyse menant à la restitution d'une partie de l'exposant privé. Le principe générale de cette analyse reprend les grandes lignes de l'attaque sur les implantations *"Right-To-Left"* (*cf.* Section 4.2.2). Dans un premier temps, nous décrirons comment extraire les bits de poids faible, puis nous généraliserons la méthode pour obtenir la totalité de l'exposant.

Collecte de signatures perturbées. La première phase de l'attaque est une phase *"en ligne"* durant laquelle l'attaquant essai de collecter plusieurs signatures perturbées, suivant le modèle, au même instant de leur exécution respective. Nous noterons $\left(\hat{S}_{(t,f)}\right)_{0 \leqslant f \leqslant F}$ le jeu de signatures perturbée à t itération de la fin de l'exponentiation. La partie d'exposant isolée sera notée $d_{[t]} = \sum_{i=0}^{t-1} 2^i d_i$ et la signature correcte S_t. Selon notre analyse statistique, l'attaquant doit récolter au moins $F = 305$ signatures perturbées, au même instant, pour espérer en avoir une exploitable.

Dictionnaire de modules premiers. Avant de commencer l'analyse, l'attaquant génère un *dictionnaire de modules premiers*. Ce dictionnaire correspond à la liste des nombres premiers appartenant à l'ensemble des valeurs de modules fautés possibles. D'un point de vue pratique, générer un tel dictionnaire revient à calculer tous les modules fautés possibles, à partir de N et du modèle de faute, puis de tester leur primalité. Afin de réaliser ces tests, nous avons choisi d'implanter l'algorithme probabiliste de Miller-Rabin [Rab80]. Par ailleurs, suivant le modèle de faute retenu, il est possible de prévoir la taille du dictionnaire en utilisant l'étude statistique précédente. Par exemple, si l'on considère toutes les modifications possibles, sur un octet, d'un module public RSA de 1024 ou 2048 bits, on peut estimer obtenir un dictionnaire de 46 entrées (*cf.* Table 4.1).

Calcul de racines carrées modulaires. Une fois le dictionnaire de modules premiers généré, l'attaquant passe à l'analyse de signatures proprement dite. Le principe consiste à utiliser les signatures erronées, et une signature correcte, pour retrouver la partie d'exposant isolée par la perturbation du module. Comme on est capable de calculer des racines carrées modulo $\hat{N}$, quand celui-ci est premier, nous essayerons d'annuler les effets de la faute sur une des signatures erronées pour retrouver la signature correcte. Plus précisément, pour chaque entrée $\hat{N}_i$ du dictionnaire,

l'attaquant choisit une valeur candidate pour le morceau d'exposant isolé par la perturbation $d'_{[t]}$. Ainsi, il peut calculer [3] :

$$R_{(d'_{[t]},\hat{N}_i)} \equiv \hat{S}_{(t,f)} \cdot \dot{m}^{-d'_{[t]}} \bmod \hat{N}_i \tag{4.44}$$

On peut remarquer que pour la bonne paire $(d_{[t]}, \hat{N})$, $R_{(d_{[t]},\hat{N})}$ doit être un résidu quadratique multiple (*i.e :* un résidu quadratique t-ième, *cf.* Section 4.3.1). Par conséquent, si $R_{(d'_{[t]},\hat{N}_i)}$ n'est pas un résidu quadratique, l'attaquant peut directement déduire que la paire de valeurs candidates $(d'_{[t]}, \hat{N}_i)$ qu'il a choisie n'est pas la bonne. Il est aussi intéressant de noter, que le test de résiduosité quadratique peut être utilisé pour éliminer prématurément les mauvaises paires. Dans notre cas, ce test peut toujours être réalisé car les valeurs candidates de modules fautés issues du dictionnaire sont primales. Le test est basé sur le théorème de Fermat :

$$\text{Si } \left(R_{(d'_{[t]},\hat{N}_i)}\right)^{(\hat{N}_i-1)/2} \equiv 1 \bmod \hat{N}_i, \tag{4.45}$$

$$\text{alors } R_{(d'_{[t]},\hat{N}_i)} \text{ est un résidu quadratique modulo } \hat{N}_i.$$

Comme tous les $\hat{N}_i$ sont premiers, ce test échoue avec une probabilité égale à 1/2 ce qui permet de filtrer, en moyenne, la moitié des $R_{(d'_{[t]},\hat{N}_i)}$. En revanche, si le test réussit, l'attaquant peut utiliser l'algorithme de Tonelli & Shanks pour calculer une racine carrée modulaire de $R_{(d'_{[t]},\hat{N}_i)}$ (*cf.* Section 4.3.2.1). L'autre racine carrée modulaire de $R_{(d'_{[t]},\hat{N}_i)}$ est obtenue en retranchant la racine retournée par le calcul précédent à $\hat{N}_i$. Ainsi, pour calculer une t-ième racine carrée de $R_{(d'_{[t]},\hat{N}_i)}$, l'attaquant doit exécuter t fois l'algorithme de Tonelli & Shanks sur chacune des racines carrées. Contrairement à ce que l'on pourrait penser, le nombre de racines carrées t-ième de $R_{(d'_{[t]},\hat{N}_i)}$ que l'on peut trouver n'explose pas exponentiellement. En pratique, nous avons trouvé que le nombre de racines carrées t-ième d'un nombre tiré aléatoirement dans un groupe multiplicatif cyclique est borné par $2^{min(t,s)}$ où s désigne la plus grande puissance de 2 divisant $\hat{N}_i - 1$. En général, cette puissance s dépasse rarement la valeur 4. Cette conjecture permet ainsi de borner le nombre de racines possibles retournées par cette étape. Ensuite, pour chacune des racines carrées t-ième de $R_{(d'_{[t]},\hat{N}_i)}$, l'attaquant poursuit sa résolution par une analyse différentielle.

Analyse différentielle. La suite de l'analyse s'inspire directement de ce que nous avons fait précédemment dans le cadre des exponentiations type *"Right-To-Left"*. A partir de chacune des racines carrées modulaires t-ième de $R_{(d'_{[t]},\hat{N}_i)}$, l'attaquant simule une fin d'exécution sans erreur en calculant :

$$S_{(d'_{[t]},\hat{N}_i)} \equiv \left(\left(R_{(d'_{[t]},\hat{N}_i)}\right)^{2^{-t}} \bmod \hat{N}_i\right)^{2^t} \cdot \dot{m}^{d'_{[t]}} \mod N \tag{4.46}$$

Pour finir, il compare le résultat de son calcul à la signature correcte :

$$S_{(d'_{[t]},\hat{N}_i)} \equiv S \mod N \tag{4.47}$$

3. Ce calcul n'est possible que si $d'_{[t]}$ est inversible $\mathbb{Z}/\mathbb{Z}\hat{N}_i$ mais, dans tous les cas que nous considérerons, les $\hat{N}_i$ sont premiers donc il est possible d'utiliser l'algorithme d'Euclide pour calculer les inverses.

Comme pour l'analyse *"Right-To-Left"* (*cf.* Section 4.2.2), lorsque cette dernière condition est remplie, pour l'une des racines carrées modulaires t-ième, cela signifie que la paire de valeurs candidates est très probablement la bonne (*cf.* Théorème 4.2.1). De cette manière l'attaquant arrive à extraire les t bits de poids faible de l'exposant privé.

4.3.3.2 Généralisation au reste de l'exposant privé

En appliquant la méthode précédente, un attaquant est capable de retrouver une partie des bits de poids faible de l'exposant privé. Le reste des bits de d peut être extrait en analysant des signatures perturbées plus tôt dans leur exécution respectives. Par conséquent, l'attaquant doit commencer par collecter plusieurs signatures perturbées $\hat{S}_{t_k,f}$ à différentes itérations t_k avant la fin de leur exécution. Le paramètre f rappelle que l'attaquant doit obtenir plusieurs signatures perturbées, au même instant t_k, pour tenir compte de la probabilité d'en obtenir une exploitable (*cf.* Section 4.3.2.2). Pour chaque ensemble de signature erronée $\left(\hat{S}_{t_k,f}\right)_{0\leqslant f\leqslant F}$, on notera $d_{[t_k]}$ la partie de d isolée par la faute :

$$\begin{aligned} d_{[t_k]} &= \sum_{i=0}^{t_k-1} 2^i d_i \\ &= \sum_{i=t_{k-1}}^{t_k-1} 2^i d_i + d_{[t_{k-1}]} \end{aligned} \tag{4.48}$$

Cette relation montre que les bits d'exposant privé peuvent être obtenus en cascade, analyse permettant de retrouver $(t_k - t_{k-1})$ bits de d. Afin d'utiliser au mieux le matériel à disposition, l'attaquant définira une taille de fenêtre w de telle sorte que $\forall k \in [\![1; K]\!]$, $t_k - t_{k-1} \leqslant w$ avec $t_0 = 0$. Ce paramètre permet de limiter l'espace de recherche pour les bits de d et donc, la complexité de l'analyse des signatures erronées.

4.3.3.3 Performances

Estimation théorique. Afin de parfaire l'étude de notre extension d'attaque par perturbation, nous proposons une analyse de ces performances. Cette estimation prend en compte à la fois la probabilité d'obtenir des signatures erronées exploitables, ainsi que la complexité du calcul de racines carrées modulaires. Le résultat de notre estimation de performances est exprimé par le théorème 4.3.1.

Théorème 4.3.1. *Considérons un algorithme de signature RSA standard de taille n implanté sur un composant avec la méthode "Left-To-Right". Alors il est possible de retrouver l'intégralité de l'exposant privé d, par fenêtres de w bits, en perturbant aléatoirement un octet du module de $\mathcal{O}\left(\frac{n^2}{3\cdot w}\right)$ signatures et au prix de $\mathcal{O}\left(2^{s+w}\cdot n^3(n-w)\right)$ exponentiations modulaires.*

Démonstration. Commençons par démontrer le résultat sur le nombre de fautes que requiert l'attaque. Comme précédemment, nous avons considérés des modifications aléatoires d'un octet du module public N. Mais, à cause du calcul de racines carrées modulaires, nous ne pouvons exploiter que les $\hat{N}$ prenant des valeurs primales. Dans la section 4.3.2.2, nous avons évalué la probabilité qu'un nombre d'exactement τ bits soit premier pr_τ. Soit F_τ la variable aléatoire représentant le nombre de fautes à provoquer pour obtenir un $\hat{N}$, alors le nombre moyen de

fautes $\mu(F_\tau)$ peut être encadré par :

$$\frac{1}{2 \cdot Sup(\tau)} < \mu\left(F_\tau\right) = \frac{1}{pr_\tau} < \frac{1}{2 \cdot Inf(\tau)}$$

Pour des tailles de RSA usuelles (*i.e.* $\tau = n = 1024$ ou 2048 bits), on utilise le théorème des gendarmes pour donner une approximation asymptotique de ce nombre moyen de fautes :

$$\mu\left(F_\tau\right) \approx \frac{\tau \cdot \ln^3(2)}{0.96} \approx \frac{\tau}{3}$$

Ainsi, ce résultat nous donne le nombre de fautes à provoquer pour obtenir une signature exploitable. Or, en analysant une signature erronée exploitable, un attaquant peut espérer extraire w bits d'exposant privé. Aussi, pour un RSA de taille n le nombre total de fautes à provoquer pour une extraction totale de l'exposant privé est :

$$\mathcal{F} = \frac{n^2}{3 \cdot w}$$

Maintenant que nous avons montré le résultat sur le nombre de fautes, intéressons nous de plus près à la complexité calculatoire. Comme pour l'attaque *"Right-To-Left"*, l'attaquant analyse les signatures erronées en testant toutes les valeurs possibles pour le couple $(d_{[t_k]}, \hat{N})$. Dans notre cas, le nombre couples possible dépend de la taille du dictionnaire de modules premiers D ainsi que de la taille de fenêtres w :

$$|(d'_{[t_k]}, \hat{N}_i)| = 2^w \cdot D \tag{4.49}$$

Afin d'identifier le bon couple parmi tous ceux qui sont testés, l'attaquant doit commencer par calculer $R_{(d'_{[t_k]}, \hat{N}_i)}$ (*cf.* Équation (4.44)). Ce calcul nécessite une exponentiation modulaire du message et une multiplication. Ensuite, si $R_{(d'_{[t_k]}, \hat{N}_i)}$ est un carré, l'attaquant calcule ces racines carrés modulaires t_k-ème. Comme nous l'indiquions dans notre conjecture, le nombre de racines trouvées est bornée par $2^{min(s,t_k)}$ où s représente la plus grande puissance de 2 divisant $\hat{N}_i - 1$. Très rapidement durant l'attaque, $t_k > s$, aussi, dans notre estimation de la complexité, nous bornerons par 2^s le nombre de racines carrés modulaires t_k-ème trouvées pour un $R_{(d'_{[t_k]}, \hat{N}_i)}$ donné.

$$\begin{aligned} C_{Racines\ carrees}(k) &= \sum_{i=0}^{s-1} 2^i \cdot C_{Tonelli\ \&\ Shanks} + 2^s \cdot \sum_{i=s}^{t_k-1} i \cdot C_{Tonelli\ \&\ Shanks} \\ &= \left(\frac{1-2^s}{1-2} + 2^s \cdot (t_k - s)\right) \cdot C_{Tonelli\ \&\ Shanks} \\ &\approx \mathcal{O}\left(2^s \cdot n \cdot t_k\right) \text{ exponentiations} \end{aligned}$$

N'oublions pas de noter que cette étape n'est réalisée que lorsque $R_{(d'_{(k)}, \hat{N}_i)}$ est un résidu quadratique ce qui n'arrive, en moyenne, que dans la moitié des cas car tous les $\hat{N}_i$ sont premiers. La dernière partie de l'analyse consiste à simuler, pour chacune des racines carrées modulaires t_k-ème obtenues, une fin d'exécution correcte puis à comparer la valeur obtenue avec la signature correcte (*cf.* Équation 4.46). Cette étape requiert le calcul de t_k carrés modulaires suivis d'une exponentiation modulaire et d'une multiplication. Ainsi la complexité de cette étape est aussi $\mathcal{O}(t_k \cdot n)$ exponentiations. D'autre part, pour extraire w bits de d avec un dictionnaire de taille D, le nombre de signatures fautées à récolter pour en obtenir une exploitable est

$N_{fautes\ par\ fenetres} = \frac{2^8 n/8}{D}$. Enfin, l'attaque nécessite d'analyser toutes les signatures collectées pour extraire la totalité de l'exposant privé. Si $t_k \leqslant k \cdot w$, alors la complexité totale est :

$$\begin{aligned} C &= \sum_{k=0}^{n/w} N_{fautes\ par\ fenetres} \cdot \frac{C_{Racines\ carrees}(k)}{2} \cdot 2^w \cdot D \qquad (4.50) \\ &= 2^{4+w} \cdot \sum_{k=0}^{n/w} 2^s \cdot n^2 \cdot k \cdot w \text{ exponentiations} \\ &= \mathcal{O}\left(2^{s+w} \cdot n^3 (n-w)\right) \text{ exponentiations} \qquad (4.51) \end{aligned}$$

□

Cette évaluation de la complexité calculatoire est utile pour dimensionner la taille de fenêtre et avoir une idée du temps effectif de l'attaque sur une machine standard. Par ailleurs, elle met en évidence quelques autres informations utiles. La première est que notre attaque appliquée contre des implantations *"Left-To-Right"* de l'exponentiation modulaire est plus complexe que celle proposée contre les implantations duales. Ceci n'est pas très surprenant puisque cette attaque requiert en plus de calculer des racines carrées modulaires. Par ailleurs, ces calculs additionnels requièrent que le module fauté soit premier, ce qui a une incidence sur le nombre de signatures perturbées à collecter.

Chapitre 5

Extension de l'analyse à un RSA avec exposant masqué

Sommaire

5.1 État de l'art

5.1.1 Algorithme de masquage d'exposant

La méthode de *masquage d'exposant* a été introduite par P. Kocher [Koc96] pour contrer les attaques par canaux auxiliaires, telles que la *Differential Power Analysis* (DPA), qui exploitent les signaux transpirants de l'exécution d'une exponentiation. Le principe de cette contre-mesure est basé sur le théorème de Fermat. En effet, $\forall m \in (\mathbb{Z}/N\mathbb{Z})^*$ et $\lambda \in \mathbb{Z}$, $m^{\lambda \cdot \varphi(N)} \equiv 1 \bmod N$. La méthode de masquage d'exposant directement inspiré du résultat précédent est détaillée dans l'algorithme 7.

Comme nous l'avons écrit précédemment, la complexité de l'algorithme d'exponentiation modulaire est polynomiale (linéaire) en la taille de l'exposant. Ainsi, pour préserver un temps d'exécution raisonnable, la taille l de l'aléa λ doit être relativement petite par rapport à la taille de l'exposant RSA n. Classiquement, pour un RSA de taille $n = 1024$ bits, on choisit $l = 20$ ou 32 bits.

Algorithme 7 Algorithme de signature RSA avec exposant masqué

ENTRÉES : $\dot{m}$, N, $\varphi(N)$, d et la taille de l'aléa l
SORTIE : $S = \dot{m}^d \bmod N$
1: {Masquage de l'exposant privé}
2: Tirer un aléa $\lambda \in [\![0; 2^l - 1]\!]$
3: $\bar{d} = d + \lambda\varphi(N)$
4: {Exponentiation modulaire}
5: $S = \texttt{Exp\_Mod}(\dot{m}, \bar{d}, N)$
6: **Retourner** S

5.1.2 Premières attaques physiques

La méthode de masquage d'exposant présentée précédemment est bien connue de l'industrie et déployée sur dans de nombreuses applications. Les principales raisons de son utilisation reposent sur sa facilité à être implantée ainsi que sur son faible coût, pour des tailles raisonnables d'aléa. Cependant sa mise en oeuvre, elle-même, peut représenter une source potentielle d'information.

La première attaque publiée contre cette contre-mesure est l'œuvre de P.-A. Fouque and F. Valette [FV03]. Leur attaque nommée *"Doubling Attack"* permet à un attaquant de retrouver la valeur d'un exposant privé RSA, même masqué, par une analyse élémentaire de la consommation du composant. Cette attaque ne s'applique que dans le cas où l'exponentiation modulaire est implantée dans la version *"Left-To-Right"* (*cf.* Section 1.3.4.1, Algorithme 2). D'autre part, l'attaquant est supposé pouvoir signer plusieurs fois le même message connu et aucune randomisation du message ne doit être effectuée avant le calcul des signatures.

À CHES 2006 [FKJM$^+$06], P.-A. Fouque *et al.* ont présenté une nouvelle attaque sur la contre-mesure de J.-S. Coron, lorsqu'elle est combinée avec une méthode d'exponentiation balayant l'exposant par fenêtres (*cf.* Section 1.3.4.3, Algorithme 5) et une petite clé publique. Dans ces conditions, une analyse élémentaire de la consommation du composant et une étude astucieuse permettent à un attaquant potentiel de retrouver la clé privée d et, par la même, de factoriser le module. Il est intéressant de noter que cette attaque exploitait déjà l'hétérogénéité de la méthode de masquage présentée dans l'algorithme 7.

Plutôt que d'exploiter la fuite d'information engendrée par l'exécution de l'exponentiation modulaire avec un exposant masqué, comme dans les attaques précédentes, P.-A. Fouque *et al.* ont proposé, à CHES 2008, de se focaliser sur l'opération de masquage elle-même. D'après l'algorithme 7, l'opération de masquage repose sur l'addition de la clé privée avec un masque aléatoire. Ces éléments étant gros par rapport à la taille des registres des processeurs, chacun des éléments sera découpé en mots, qui seront additionnées successivement pour donner le résultat escompté. Dans ces conditions, l'espionnage de la valeur de chacune des retenues (résultant d'un dépassement de la somme de deux mots) peut révéler une certaine quantité d'information qui pourra être utilisée pour retrouver les bits de poids fort de chacun des mots constituant la clé secrète. Enfin, dès que le nombre de bits manquants d'exposant privé est suffisamment petit, l'attaquant pourra finir la résolution avec des méthodes mathématiques comme les *pas de bébé/pas de géant*.

La méthode de masquage d'exposant pourrait aussi être utilisée pour protéger les implantations de la signature RSA contre certaines attaques par perturbation. En particulier, cette contre-mesure pourrait être avantageusement utilisée pour enrayer les attaques nécessitant de récolter plusieurs signatures erronées, puisque chacune d'entre elles sera réalisée avec un ex-

posant différent.

Ce chapitre comble ce vide en présentant la première attaque exploitant les perturbations du module RSA pour retrouver un exposant privé, même masqué. Cette attaque s'inspire largement de nos précédents travaux sur les implantations classiques de RSA et repose également sur l'hétérogénéité du masquage des bits d'exposant. Par conséquent, les résultats obtenus par notre étude complètent l'état de l'art des attaques par canaux auxiliaires contre la contre-mesure de masquage d'exposant.

5.2 Perturbation du module public d'un RSA masqué

5.2.1 Analyse binaire d'un exposant masqué

Dans cette section, nous nous attarderons sur la véritable efficacité du masquage d'exposant tel qu'il est réalisé dans l'algorithme 7. Par définition, un exposant privé $\bar{d}$ est masqué par l'addition d'un multiple aléatoire de $\varphi(N)$. Bien que $\varphi(N) = (p-1)(q-1)$ ne soit pas connu par l'attaquant, une certaine quantité d'information peut quand même fuir de son expression. Dans ce cas, on peut encore écrire l'exposant masqué $\bar{d}$ de la manière suivante :

$$\begin{aligned} \bar{d} &= d + \lambda\varphi(N) \\ &= d + \lambda\,(p-1)\,(q-1) \\ &= d + \lambda N - \lambda\,(p+q-1) \end{aligned} \tag{5.1}$$

D'après l'équation précédente, la construction d'un exposant masqué peut se ramener à l'addition de trois termes de tailles différentes. Par conséquent, ces différents termes n'influenceront pas le masquage de l'exposant de la même manière. En d'autres termes les bits d'exposant privé d ne sont pas masqués de manière homogène par cette méthode. La figure 5.2.1 illustre cette affirmation dans le cas d'un RSA de taille n masqué par un aléa λ de taille l. Cette figure montre

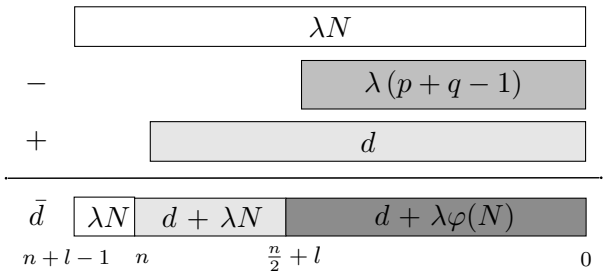


FIGURE 5.1 – Analyse binaire d'un masquage d'exposant

clairement que, contrairement à ce que l'on pourrait penser, l'exposant n'est pas masqué par un multiple aléatoire d'une valeur inconnue de la taille du RSA (*i.e.* $\varphi(N)$). Sur les poids forts, les bits de d sont véritablement masqués par un multiple aléatoire du module public N. Cependant, cette méthode semble être plus efficace pour masquer les poids faibles de d puisque les nombres premiers tenus secrets p et q interviennent également dans le calcul.

Avec l'apparition des premières attaques d'implantations du RSA par perturbation des éléments publics, certains se sont demandé si cette contre-mesure ne pouvait pas aussi être utilisée pour contrer de telles attaques. Également auteurs de la première attaque exploitant les perturbations de $\hat{N}$, E. Brier *et al.* affirment dans [BCMCC06] que cette méthode permet effectivement

de protéger les implantations de RSA contre l'analyse de perturbation du module public. Un peu plus tard, à CT-RSA 2009, nous avons aussi abondé dans ce sens en conjecturant la résistance des implantations de RSA avec masquage d'exposant contre les attaques présentés dans les deux chapitres précédents, ou encore dans [BCG08b, BCDG09]. Cependant, en regardant de plus près la méthode de masquage, nous avons montré que celle-ci est inefficace pour protéger les implantations de RSA contre des perturbations du module public intervenant pendant l'exécution d'une signature. Les sections suivantes détaillent la méthode d'analyse que nous avons imaginée.

5.2.2 Exemple d'exécution perturbée

5.2.2.1 Cadre de l'attaque

Avant de commencer la présentation du principe que nous avons imaginé pour attaquer les implantations de signatures RSA avec masquage d'exposant, définissons précisément le cadre de l'attaque. Le modèle de perturbation que nous avons considéré repose sur la perturbation transitoire d'un octet du module public N pendant l'exponentiation modulaire propre à la signature RSA. Nous avons choisi ce modèle pour sa reproductibilité dans le domaine des cartes à puces [DLV03, BO06] mais aussi parce que nous avons montré qu'il pouvait permettre de retrouver la clé privée pour des implantations classiques de la signature RSA. Concernant justement ces implantations, nous considérerons ici que c'est un méthode *"Right-To-Left"* qui a été choisie pour implanter la signature. Cependant, ce choix est purement arbitraire et l'attaque proposée pourra être généralisée aux implantations duales en utilisant les résultats présentés dans le chapitre précédent ou encore dans [BCDG09]. Enfin, le paramètre t désignera le l'instant où la perturbation est injectée pendant l'exécution de la signature. La section suivante détaille un exemple d'exécution erronée réalisée sous ces hypothèses d'attaque.

5.2.2.2 Résultat d'une exécution erronée

Afin d'illustrer le résultat d'une exécution erronée, dans le cadre d'attaque décrit précédemment, nous noterons $\bar{d} = \sum_{i=0}^{n+l-1} 2^i \cdot \bar{d}_i$ la valeur d'un exposant masqué dans sa représentation binaire. D'autre part, puisque la faute est injectée avant le calcul d'un carré modulaire, à la t-ième itération d'une exponentiation modulaire de type *"Right-To-Left"*, $B_{t-1} \equiv \dot{m}^{2^{t-1}} \bmod N$ sera la valeur du second registre interne avant la perturbation. Ainsi, la perturbation commence par infecter le calcul de $\hat{B}_t$ de la manière suivante :

$$\begin{aligned} \hat{B}_t &\equiv {B_{t-1}}^2 \bmod \hat{N} \\ &\equiv \left(\dot{m}^{2^{t-1}} \bmod N\right)^2 \bmod \hat{N} \end{aligned} \tag{5.2}$$

Comme stipulé dans le modèle, nous considérons une faute transitoire infectant la valeur du module jusque la fin de l'exponentiation. Ainsi, en notant $A_t \equiv \dot{m}^{\sum_{i=0}^{t-1} 2^i \cdot \bar{d}_i}$ la valeur de l'autre registre interne non perturbée, la signature obtenue peut être écrite :

$$\hat{S}_t \equiv A_t \cdot \hat{B}_t^{\bar{d}_t} \cdot \ldots \cdot \hat{B}_t^{2^{(n+l-1)-t} \cdot \bar{d}_{n+l-1}} \bmod \hat{N} \tag{5.3}$$

$$\equiv A_t \cdot \hat{B}_t^{\frac{\bar{d}_{[t]}}{2^t}} \bmod \hat{N} \tag{5.4}$$

où $\bar{d}_{[t]} = \sum_{i=t}^{n+l-1} 2^i \cdot \bar{d}_i$. Comme on pouvait s'y attendre, la perturbation isole une partie des poids forts de l'exposant masqué $\bar{d}$. En d'autres termes, la modification intentionnelle du module public permet à un attaquant de se focaliser sur l'extraction d'une partie de l'exposant utilisé

pour le calcul de la signature. Cependant, dans ce cas précis, c'est une partie d'exposant masqué que l'attaquant est susceptible de retrouver. Or, la méthode de masquage rafraîchit l'exposant utilisé pour chaque calcul de signature. Par conséquent il n'est, *a priori*, pas possible d'utiliser la connaissance d'une partie d'exposant masqué pour retrouver l'exposant privé en cascade, comme cela a été fait précédemment (*cf.* Section 4.2).

Dans les sections suivantes, nous allons montrer comment nous avons profité de l'hétérogénéité de la méthode de masquage pour retrouver un morceau d'exposant privé depuis l'exposant masqué. Nous commencerons par rappeler la méthodologie générale de l'analyse différentielle de perturbations du module public puis, nous détaillerons les méthodes que nous avons imaginées pour extraire des morceaux d'exposant privé à partir de morceaux d'exposant masqué respectivement pour les poids forts (*cf.* Section 5.2.3.2) et les poids faibles (*cf.* Section 5.2.3.3).

5.2.3 Analyse différentielle de la perturbation

5.2.3.1 Principe général

Le principe général de l'attaque consiste à profiter de l'isolation d'une partie de l'exposant par la perturbation du module public. En effet, si cette partie d'exposant isolée est suffisamment petite, il sera alors possible de la déterminer en même temps que le module fauté grâce à la connaissance d'un couple signature perturbée/correcte $(\hat{S}_t, S_t)$. Pour ce faire, l'attaquant utilise le modèle de perturbation pour proposer une valeur candidate pour $\hat{N}$ et cherche simultanément une autre valeur candidate $\bar{d}'_{[t]}$ pour les bits de poids fort de l'exposant masqué. Avec ces valeurs candidates, l'attaquant commence par calculer :

$$A'_t \equiv S_t \cdot \dot{m}^{-\bar{d}'_{[t]}} \bmod N \tag{5.5}$$

Ce premier calcul sert à retourner à l'état du registre interne A_t précédent l'injection de la perturbation. Dans un second temps, l'attaquant utilise ses hypothèses pour simuler une fin d'exponentiation erronée. Ceci peut être réalisé via le calcul suivant :

$$S'_{(\bar{d}'_{[t]},\hat{N}')} \equiv A'_t \cdot \left(\dot{m}^{2^{t-1}} \bmod N\right)^{2\cdot\frac{\bar{d}'_{[t]}}{2^t}} \bmod \hat{N}' \tag{5.6}$$

Enfin, l'attaquant valide ses hypothèses en vérifiant que la signature erronée simulée correspond bien à la signature erronée récupérée en sortie du composant :

$$S'_{(\bar{d}'_{[t]},\hat{N}')} \equiv \hat{S}_t \bmod \hat{N}' \tag{5.7}$$

Lorsque l'équation précédente est satisfaite, l'attaquant peut déduire que le couple de valeurs candidates choisies est le bon avec une forte probabilité (*cf.* Théorème 4.2.1). Dans le cas contraire, l'attaquant doit répéter le calcul précédent pour un autre couple de valeurs candidates. Notons enfin qu'une analyse similaire peut être réalisée lorsque le module est modifiée pour une seule opération, carré ou multiplication modulaire (*cf.* Section 4.2.3.2 pour plus de détails).

Perturbations inexploitables Ce cas correspond à des perturbations du module public intervenant dans les ultimes itérations de l'exponentiation, plus précisément quand $n \leqslant t \leqslant (n + l - 1)$. Dans cette zone de l'exposant masqué, la faible quantité d'information concernant l'exposant privé est due à la propagation de la retenue engendrée par l'addition de d et d'un multiple aléatoire de $\varphi(N)$ (*cf.* Figure 5.2.1). Par conséquent, analyser des signatures perturbées à ces instants de l'exécution n'est pas un choix pertinent. Dans la suite, nous nous focaliserons donc sur l'étude de signatures perturbées plus tôt dans leur exécution respective

5.2.3.2 Extraction des poids forts d'exposant privé

Dans cette section, nous allons développer la méthode que nous avons imaginée pour retrouver les poids forts de la clé privée RSA. Ce cas d'étude correspond à des signatures dont le module a été perturbé pendant que le t-ième bit d'exposant est traité et de telle sorte que $\left(\frac{n}{2}+l\right) \leqslant t < n$. Comme nous l'avons déjà expliqué dans la section précédente, notre but est de déduire les bits d'exposant privé à partir des bits d'exposant masqué extraits par l'analyse différentielle des effets de la perturbation. Cependant, réaliser l'analyse de cette manière semble difficile.

Premièrement $\bar{d}$ dépend de d mais aussi d'un multiple aléatoire de $\varphi(N)$ et, par construction, toutes ces valeurs sont inconnues par l'attaquant. Cette première difficulté peut être surmontée grâce à l'hétérogénéité de la méthode de masquage (*cf.* Figure 5.2.1). En effet, dans l'intervalle que nous considérons ici (*i.e.* $\left(\frac{n}{2}+l\right) \leqslant t < n$) nous pouvons écrire :

$$\bar{d} \approx d + \lambda N \tag{5.8}$$

Par conséquent, on peut affirmer que les poids forts de $\bar{d}$ ne dépendent que des poids forts de la clé privée d et de l'aléa λ.

D'autre part, plutôt que de chercher à extraire certains bits de d depuis les parties de $\bar{d}_{[t]}$ trouvés grâce à l'analyse différentielle, nous avons choisi de prendre le problème dans l'autre sens. Plus précisément, nous allons chercher simultanément des valeurs candidates pour d et λ de façon à construire de *bons* candidats pour $\bar{d}_{[t]}$:

$$\bar{d}_{[t]} = \sum_{i=t}^{n+l-1} 2^i \cdot \bar{d}_i \tag{5.9}$$

$$\approx \sum_{i=t}^{n+l-1} 2^i \cdot (d+\lambda N)_i \tag{5.10}$$

$$\approx \sum_{i=t}^{n-1} 2^i \cdot d_i + \sum_{i=t}^{n+l-1} 2^i \cdot (\lambda N)_i + \mathit{Retenue}_t\,(d, \lambda N) \cdot 2^t \tag{5.11}$$

où la notation $\mathit{Retenue}_t\,(a,b)$ représente le bit de retenue résultant de l'addition bit-à-bit des t premiers bits des opérandes a et b. L'équation (5.11) montre que chercher de bons candidats pour $\bar{d}_{[t]}$ revient à chercher simultanément l'aléa λ et les $w = n - t$ bits de poids fort de l'exposant d. Dans le cas général, cette partie de d est composée d'une partie connue (ou déjà retrouvée) notée d_{MSB} et d'une partie inconnue de w bits notée d_w. Pour alléger les notations, nous prendrons en compte le bit de retenue comme une incertitude sur la parité de d_w. Par conséquent, si un attaquant réussit à construire une valeur candidate pour $\bar{d}_{[t]}$ qui vérifie (5.7), alors il pourra déduire directement la valeur d'aléa utilisée pour le masquage, mais surtout $w-1$ nouveaux bits de d. Alors, la partie connue d_{MSB} grandit de $w-1$ bits, et cette partie pourra être réutilisée pour analyser des signatures perturbées plus tôt dans leur exécutions. Cette méthode présente l'avantage de retrouver en cascade les bits de poids fort tout en garantissant que la partie à déterminer pour chaque analyse d_w soit de taille constante (ou au moins bornée). Finalement, au lieu de chercher à déterminer des couples de valeurs satisfaisant (5.7), comme décrit dans le principe général, il faudra chercher des triplets de valeurs pour $(d_w, \lambda, \hat{N})$ permettant de calculer de bonnes valeurs candidates pour $\bar{d}_{[t]}$.

En étudiant plus méticuleusement cette amélioration, on peut légitimement se demandre si une seule équation est suffisante pour identifier un triplet de valeurs, avec une grande probabilité. Cette remarque est d'autant plus pertinente que l'implantation de notre attaque nous a

montré que plusieurs triplets de valeurs candidates peuvent satisfaire l'équation (5.7). Que ce soit dans [BCG08b, BCDG09] ou dans les deux chapitres précédents, nous avons démontré que l'ordre de grandeur de la probabilité de faux-acceptants est de $\frac{1}{N}$. Cette probabilité est donc négligeable pour des tailles de RSA usuelles. Cependant, nous avons aussi noté dans nos expérimentations que pour tous les faux triplets passant la vérification, les valeurs candidates pour l'aléa retenues sont systématiquement inférieures à la bonne. Par conséquent, nous avons commencé par ne garder que le triplet satisfaisant l'équation (5.7), qui contient la plus grande valeur candidate pour λ. L'utilisation de cette heuristique, nous a permis d'améliorer considérablement les performances de notre algorithme d'attaque. Cette heuristique a ensuite été formalisée dans le théorème 5.2.1, dont la preuve est donnée ensuite.

Théorème 5.2.1. *Soit $\hat{S}_t$ une signature perturbée dont l'exposant est masqué par l'aléa λ, et soit S la signature correcte correspondante. Pour toutes les valeurs candidates $(d'_w, \lambda') \in [\![0; 2^w]\!] \times [\![0; 2^l]\!]$, si $t < n$, la taille du RSA, et (5.7) est satisfaite. alors $\lambda' \leqslant \lambda$.*

Démonstration. La preuve se fait en raisonnant par l'absurde. Soit λ' une valeur candidate pour l'aléa λ telle que $\lambda' > \lambda$. On peut alors écrire $\lambda' \geq \lambda + 1$. Maintenant, nous allons utiliser la relation précédente pour calculer une valeur candidate pour le morceau d'exposant masqué $d_{[t]}$:

$$\begin{aligned} \lambda' N &\geq (\lambda + 1) N && (5.12)\\ \Leftrightarrow \lambda' N &\geq \lambda N + N && (5.13) \end{aligned}$$

Or, puisque l'exposant privé d est calculé comme l'inverse de e modulo $\varphi(N)$, nous savons aussi que :

$$N > \varphi(N) > d \tag{5.14}$$

Par conséquent, on peut déduire de la relation précédente que :

$$\begin{aligned} \lambda' N &> \lambda N + d && (5.15)\\ \Rightarrow \lfloor \frac{\lambda' N}{2^t} \rfloor &\geqslant \lfloor \frac{\lambda N + d}{2^t} \rfloor && (5.16) \end{aligned}$$

Mais, comme on considère que $t < n$, la relation précédente est en fait une inégalité stricte :

$$\lfloor \frac{\lambda' N}{2^t} \rfloor > \lfloor \frac{\lambda N + d}{2^t} \rfloor \tag{5.17}$$

Utilisons à présent la représentation binaire de chacune des opérandes pour modifier l'expression précédente :

$$\begin{aligned} \sum_{i=t}^{n+l-1} 2^{i-t} \cdot \left(\lambda' N\right)_i &> \sum_{i=t}^{n+l-1} 2^{i-t} \cdot (\lambda N + d)_i && (5.18)\\ \Leftrightarrow \sum_{i=t}^{n+l-1} 2^{i} \cdot \left(\lambda' N\right)_i &> \sum_{i=t}^{n+l-1} 2^{i} \cdot (\lambda N + d)_i && (5.19)\\ \Leftrightarrow \sum_{i=t}^{n+l-1} 2^{i} \cdot \left(\lambda' N\right)_i &> \bar{d}_{[t]} && (5.20) \end{aligned}$$

Cette inégalité nous permet de conclure que toutes les valeurs candidates pour le morceau d'exposant masqué calculé avec un tel λ' seront toujours strictement grande que le morceau d'exposant masqué $\bar{d}_{[t]}$ recherché. Cela signifie que pour tout $\lambda' > \lambda$, l'équation (5.7) ne sera jamais satisfaite. □

Pour conclure, en combinant notre approximation d'un exposant masqué (*cf.* Équation (5.11)) avec le théorème précédent, il sera possible de retrouver une partie de l'exposant privé d à partir de l'analyse différentielle d'une seule paire signature correcte/erronée. Comme pour nos précédentes applications aux implantations classiques de la signature RSA, l'avantage de cette attaque réside dans la possibilité de cascader l'extraction des bits de d. en utilisant les parties déjà obtenues par des analyses antérieures De cette manière, il est possible de retrouver presque la moitié des bits de poids fort de d. La section suivante décrit comment extraire le reste des bits de l'exposant privé.

5.2.3.3 Extraction des poids faibles d'exposant privé

Dans cette partie, nous allons nous focaliser sur les perturbations de signatures isolant une partie des poids faibles de l'exposant masqué. D'un point de vue plus formel, cela revient à considérer les signatures dont le module est perturbé à un instant t tel que $0 \leqslant t < \left(\frac{n}{2} + l\right)$.

Contrairement à l'analyse des poids forts, on ne peut pas négliger ici l'influence des nombres premiers p et q dans le masquage de l'exposant. Puisque ces valeurs sont privées, contrairement au module N, l'attaquant ne peut pas utiliser l'analyse précédente pour retrouver les bits de poids faible de l'exposant privé d. Nous allons donc montrer comment nous avons utilisé l'analyse différentielle de plusieurs signatures RSA, perturbées aux mêmes instants de leur exécution, pour contourner cette difficulté.

Comme nous l'avons affirmé précédemment, les bits de poids faible d'un exposant masqué ne peuvent pas s'exprimer comme les bits de poids faible issus de la somme de l'exposant privé d et d'un multiple aléatoire du module public N. Comme l'illustre la figure 5.2.1 $\varphi(N)$ est, cette fois, véritablement impliqué dans le masquage des bits de poids faible. Malgré cela, prenons la peine d'exprimer un morceau d'exposant isolé par la perturbation du module à la t-ième itération d'une exponentiation modulaire *"Right-To-Left"* :

$$\bar{d}_{[t]} = \sum_{i=t}^{n+l-1} 2^i \cdot (d + \lambda\varphi(N))_i \tag{5.21}$$

$$= \sum_{i=t}^{n+l-1} 2^i \cdot (d + \lambda N - \lambda(p+q-1))_i \tag{5.22}$$

$$\approx \sum_{i=t}^{n-1} 2^i \cdot \delta_i + \sum_{i=t}^{n+l-1} 2^i \cdot (\lambda N)_i \tag{5.23}$$

où $\delta_i = (d - \lambda(p+q-1))_i$. En réalisant cette substitution, nous faisons apparaître une expression proche de celle utilisée pour analyser les poids forts. En pratique, l'avantage de cette substitution est qu'elle permet de retrouver efficacement la valeur de l'aléa λ sans connaître $\varphi(N)$.

Pour réaliser l'analyse proprement dite, nous allons supposer que, comme pour l'analyse des poids forts, une partie de d a déjà été déterminée. Cette partie sera notée d_{MSB}. La nouveauté dans l'analyse des poids faibles est que nous allons considérer aussi qu'une partie de la somme des entiers $(p+q-1)$ a pu être déterminée. Cette partie sera notée $(p+q-1)_{MSB}$. Considérons à présent la variable de substitution $\delta = \sum_{i=t}^{n-1} 2^i \cdot \delta_i$. Avec les notations précédentes, nous pouvons exprimer δ comme une partie δ_w, contenant w bits inconnus et une partie δ_{MSB} qui dépend de λ, d_{MSB} et $(p+q-1)_{MSB}$. Cette dernière partie est donc complètement déterminée lorsque la valeur de λ a pu être identifiée. Justement, l'équation (5.23) montre que l'attaquant peut mener la même analyse que pour les poids forts pour retrouver les parties d'exposant masqué isolées

par la perturbation. La seule différence réside dans la substitution de d_w par δ_w. En d'autres termes l'attaquant cherche simultanément les morceaux d'exposant masqué $\bar{d}_{[t]}$ et le module erroné $\hat{N}$ satisfaisant l'équation (5.7) en cherchant le triplet de valeurs candidates $(\delta'_w, \lambda', \hat{N}')$. Les morceaux d'exposant masqué inconnus seront alors construits à partir de δ_w et λ. Le principal avantage de cette analyse préliminaire est qu'elle permet de déterminer précisément l'aléa utilisé pour réaliser la signature erronée.

Par ce biais, l'analyse différentielle d'une paire de signatures RSA correcte/erronée renvoie le bon triplet de valeurs avec une forte probabilité. Cependant, on ne peut pas déduire directement d'information sur l'exposant privé d, comme cela était fait pour l'analyse des poids forts. En effet, l'information extraite concerne la valeur de substitution δ_w. Par conséquent, l'attaquant doit mener une analyse complémentaire sur cette variable δ_w pour extraire de l'information à la fois sur d et $\lambda(p+q-1)$. En se référant à la figure 5.2.1, il est pertinent de noter que δ_w dépend des données suivantes :

- w bits inconnus d'exposant privé d,
- w bits inconnus pour la somme des premiers RSA $(p+q-1)$,
- la valeur aléatoire λ fraîchement déterminée,
- quelques bits de retenue.

Par conséquent, la connaissance δ_w permet d'obtenir une équation à deux inconnues : d_w et $(p+q-1)_w$. Pour pouvoir récupérer simultanément ces w bits d'exposant privé et ces w de la somme des premiers RSA, il est nécessaire d'obtenir des équations supplémentaires (au moins une de plus). Pour ce faire, il suffit d'obtenir une autre signature RSA dont le module a été perturbé à la même itération. En réalisant une nouvelle analyse différentielle avec cette nouvelle signature RSA erronée, l'attaquant obtient une nouvelle valeur pour l'aléa λ ainsi que pour δ_w. On peut alors déduire de cette dernière une équation additionnelle en vue de l'extraction de d_w et $(p+q-1)_w$.

Dans un souci de clarté, nous avons volontairement omis de parler de l'influence des bits de retenue sur la résolution du système d'équations formé. En pratique, comme pour l'analyse des poids forts, ces retenues apportent de l'incertitude sur les valeurs de d_w retrouvées. En effet, plutôt que de retrouver une solution unique pour d_w en pratique, l'algorithme peut se retrouver en face de deux voire trois solutions possibles. Cependant ceci n'est pas gênant puisque le mauvaises valeurs de d_w sont rejetées au moment d'analyser des signatures perturbées plus tôt dans leur exécution. Par conséquent, le nombre de valeurs possibles pour la clé n'augmente pas exponentiellement avec l'analyse en cascade mais reste borné tout au long de l'attaque. Finalement, lorsqu' une grande partie des poids faibles de d et de $\varphi(N)$ a été déterminée l'attaquant pourra avantageusement utiliser les attaques réseaux, type Coppersmith [Cop96, Bau08], pour complètement déterminer la clé privée.

5.2.3.4 Récapitulatif de l'attaque

Ce paragraphe récapitule l'analyse différentelle de perturbation que nous avons imaginée pour retrouver des clés privées RSA malgré l'utilisation d'une méthode de masquage. Ce résumé apporte une vision plus pragmatique de la méthodologie que nous avons mis en œuvre dans notre implantation PC, avec la librairie GMP.

Collecte de plusieurs signatures erronées. L'attaquant commence par définir la granularité de la résolution de la clé privée d en fixant la taille de fenêtre de résolution w. Ensuite, il doit récupérer plusieurs signatures erronées à différents instants de leur exécution respective. En notant t l'instant de l'injection de faute, l'attaquant devra collecter :

- Un couple de signatures correcte/erronée
  pour $\left(\frac{n}{2}+l\right) \leqslant t < n$
- Deux (ou plus) couples signatures correcte/erronées
  pour $0 \leqslant t < \left(\frac{n}{2}+l\right)$

où t est diminué de la valeur w entre chaque perturbation de signatures. Cet ensemble de signatures pourra être trié dans l'ordre décroissant des instants t de l'injection de faute afin de faciliter l'analyse.

Récupération des poids forts de d. Pour chaque paire de signatures correcte/erronée, l'attaquant mène une analyse de type *guess-and-determine* pour retrouver le bon triplet de valeurs $(d_w, \lambda, \hat{N})$ avec une forte probabilité. Cette analyse permet à l'attaquant de retrouver w nouveaux bits de l'exposant privé à partir des données déjà retrouvées. La répétition de cette analyse différentielle sur tous les couples de signatures correcte/erronée $(S_t, \hat{S}_t)$ tels que $\left(\frac{n}{2}+l\right) \leqslant t < n$ permet ainsi de retrouver presque la moitié des bits de d.

Récupération des poids forts de d. Avec le reste des signatures collectées, l'attaquant doit réaliser une analyse scindée en deux phases :

- Premièrement, il essaie de déterminer au moins deux triplets de valeurs $(\delta_w, \lambda, \hat{N})$ en analysant autant de couples perturbées au même instant t de leur exécution respective.
- Ensuite, il utilise les données extraites pour former un système d'équations. La résolution du système permet de retrouver à la fois w bits de d et w bits de la somme des premiers $(p+q-1)$. Le reste des bits de poids faible de d seront déterminés en répétant cette analyse (complétée éventuellement par une analyse mathématique type *Pas de Bébé/Pas de Géant* ou une *attaque réseau*).

5.2.3.5 Performances

Estimation théorique. Puisque le coeur de notre attaque repose sur l'exploitation de perturbations du module public, il nous a semblé pertinent de commencer par estimer le nombre de fautes à réaliser pour retrouver l'exposant. D'après l'analyse que nous avons présentée (*cf.* Section 5.2.3), ce nombre varie en fonction de la partie de d que l'on souhaite retrouver. Dans le cas des poids forts, nous avons affirmé qu'un seul couple de signature correcte/erronée permet de retrouver w bits de l'exposant privé d. En revanche, pour les poids faibles, il faut analyser k signatures RSA (*i.e.* $k \geqslant 2$) dont le module a été modifié au même instant de leur exécution respective et résoudre le système d'équations correspondant pour obtenir w bits de d. Par conséquent, le nombre total de signatures perturbées à extraire $\mathcal{F}$ pour retrouver complètement d est :

$$\mathcal{F} = \mathcal{O}\left(\frac{kn}{w}\right) \tag{5.24}$$

En pratique, pour un RSA de taille 1024 bits et une taille de fenêtre de $w = 2$, notre attaque nécessite de collecter en moyenne 1000 signatures erronées. ce qui est plus qu'attendu. Ce surcoût est expliqué par le nombre de signatures moyen que requiert chaque extraction d'une partie des poids faibles de d (et aussi de $(p+q-1)$). En pratique ce nombre est compris entre 2 et 3. Cependant, ce nombre de perturbations reste raisonnable et apporte, par la même occasion, une nouvelle preuve du caractère réaliste de notre attaque.

L'évaluation de la complexité calculatoire est un autre point classiquement utilisé pour évaluer l'efficacité d'une analyse de perturbations. Dans notre cas, le principe général consiste à utiliser des signatures dont le module a été modifié à différents instants de leur exécution respective pour extraire des morceaux de l'exposant privé d. Comme nous l'avons précédemment montré, la valeur de l'exposant masqué isolé par une perturbation est identifiée en utilisant les

données précédemment retrouvées, mais surtout, en faisant des hypothèses simultanément sur les w bits de d inconnus, la valeur d'aléa sur l bits et sur la valeur du module perturbé $\hat{N}$. En se référant au modèle de perturbation que nous avons considéré et sur notre algorithme d'attaque, la complexité calculatoire $\mathcal{C}$ peut être exprimée ainsi :

$$\mathcal{C} = \mathcal{O}\left(\frac{2^{(l+w)} \cdot n^2}{w}\right) \text{exponentiations} \tag{5.25}$$

Dans cette expression, nous n'avons pas tenu compte de la résolution du système d'équations nécessaires à l'extraction des bits de poids faible. Et ce, tout simplement car le coût de cette résolution est dominé par le coût de la recherche des triplets de valeurs satisfaisant l'équation (5.7). Par ailleurs, il est intéressant de noter que le terme dominant dans l'estimation de la complexité est 2^{l+w}. Cela signifie, d'une part, qu'il faudra plutôt choisir w petit si les capacités de calcul sont limités, mais encore que la complexité augmente exponentiellement avec la taille de l'aléa. Cependant, ce facteur 2^l ne doit pas être effrayant puisque certaines optimisations permettent de relativiser son influence.

Optimisations possibles. Ce paragraphe recense quelques méthodes que nous proposons pour optimiser les performances de notre attaque. Premièrement, plutôt que de calculer les valeurs candidates pour le module erroné "à la volée", l'attaquant pourra utiliser le modèle de perturbation pour construire un dictionnaire contenant toutes ses valeurs possibles. D'autre part, le résultat formalisé dans le théorème 5.2.1 pourra inciter l'attaquant à chercher les valeurs candidates pour l'aléa de manière décroissante puisque c'est la plus grande valeur permettant de satisfaire l'équation (5.7) qui devra être retenue. Cette remarque est d'autant plus intéressante à utiliser lorsque λ est proche de 2^l. Enfin, la recherche des bons triplets de valeurs pourra être avantageusement parallélisée, puisque les calculs ainsi que les tests sont indépendants les uns des autres. Par conséquent, si l'attaquant dispose d'un réseau de plusieurs coeurs, la complexité de la recherche pourra être réduite d'un facteur de l'ordre du nombre de cœurs.

5.3 Conclusion

Dans ce chapitre, nous avons présenté la première attaque par perturbation jamais publiée contre les implantations de la signature RSA utilisant la contre-mesure de masquage d'exposant. Cette attaque est d'autant plus intéressante qu'elle exploite les perturbations du module public N, pendant l'exécution de signatures. Précédemment, P.-A. Fouque et F. Valette avaient alerté la communauté sur les vulnérabilités physiques que pouvait engendrer l'implantation de cette contre-mesure. En effet, leur *"Doubling Attack"* [FV03] a montré que de l'information exploitable pouvait fuir de l'implantation d'une signature RSA *"Left-To-Right"* utilisant le masquage d'exposant.

Contrairement à ce que nous avions conjecturé à CT-RSA 2009, en s'appuyant sur les conclusions de E. Brier *et al.* [BCMCC06], le masquage d'exposant ne peut pas être considéré comme une méthode efficace pour enrailler les attaques par perturbations des données publiques. Nous avons montré, tout au long de ce chapitre, qu'en récoltant un peu plus de signatures et en réalisant une étape de traitement supplémentaire, notre attaque contre les implantations classiques de la signature RSA pouvait s'étendre aux signatures protégées par le masquage d'exposant. Par ailleurs, notre implantation de l'attaque, comme la considération d'un modèle de perturbation réaliste prouvent que la perturbation des éléments publics représente une menace réelle pour les implantations de RSA, même masquées. Par conséquent, nous pensons qu'il pourrait être

intéressant d'étudier la résistance de la méthode de masquage d'exposant face à d'autres types de perturbations. Ces travaux ont été publiés à la conférence CHES 2010.

Chapitre 6

Analyse de la perturbation d'un module DSA

Sommaire

6.1 État de l'art

Comme nous avons pu le voir précédemment, l'exploitation des perturbations d'éléments publics peut mener à des attaques très efficaces. Alors que les premières applications de ces attaques ont vu le jour contre les implantations de crypto-systèmes basés sur les courbes elliptiques [BMM00]. Ce sont ensuite les implantations de l'algorithme RSA qui ont largement été étudiées [Sei05, BCMCC06, BCDG09]. Cela étant, très peu de travaux ont attrait à la sécurité, vis à vis des perturbations, des éléments publics de crypto-systèmes basés sur le problème du logarithme discret. En effet, la seule attaque de ce type que nous avons répertorié est due à une contribution assez récente de C.H. Kim *et al.* [KBPJJ08]. Cette attaque propose une méthode permettant d'exploiter les perturbations des modules publics utilisés dans le crypto-système El Gamal et l'algorithme de signature DSA. Bien que le modèle de perturbation choisi soit reproductible, le nombre de perturbations à réaliser pour réussir l'analyse est plutôt conséquent (*i.e.*

40 millions de signatures erronées pour obtenir une clé secrète DSA de 160 bits). Cet inconvénient a amené les auteurs à conjecturer que les crypto-systèmes basés sur le logarithme discret seraient moins vulnérables que les autres, vis-à-vis des perturbations d'éléments publics.

Cela étant, en approfondissant notre recherche bibliographique, nous avons trouvé une attaque par perturbation contre le DSA présentant une efficacité intéressante. Cette attaque a été proposée par D. Naccache *et al.* [NNTW05] et consiste à utiliser les perturbations pour forcer à zéro des morceaux d'aléas k utilisés pour générer des signatures. Sous ce modèle de faute, il suffit d'analyser 27 signatures DSA erronées pour retrouver une clé secrète. Le principe de l'analyse consiste à réaliser une *attaque réseau* sur les données collectées. Elle s'inspire notamment des travaux réalisés par N.A. Howgrave-Graham and N.P. Smart [HGS01] puis améliorées par P.Q. Nguyen et I. Shparlinski [NS02].

Dans le paragraphe suivant, nous présenterons plus en détail l'attaque de C.H. Kim *et al.* [KBPJJ08], la seule à exploiter les perturbations d'éléments publics de DSA et El Gamal. En remarquant que la praticabilité du modèle de perturbation proposé dans [NNTW05] pouvait être améliorée, nous montrerons ensuite comment nous nous sommes inspirés des attaques réseaux pour proposer la meilleure attaque par perturbation des éléments publics de signatures type El Gamal. Enfin, les résultats obtenus par le biais de notre attaque prouvent que les crypto-systèmes basés sur le logarithme discret sont tout aussi vulnérables face aux perturbations des éléments publics et soulignent l'importance de protéger aussi ces données.

6.1.1 Attaque de C.H. Kim *et al.*

Bien qu'il ait déjà été démontré que la perturbation d'éléments publics peut être une source d'information [BMM00, Sei05], les premières applications contres des crypto-systèmes basés sur le logarithme discret sont très récentes. C'est à EuroPKI 2008 que C.H. Kim *et al.* [KBPJJ08] ont montré, pour la première fois, que le calcul de signatures DSA avec des modules erronés pouvaient être potentiellement exploitables. Afin de rendre l'attaque facilement reproductible, les auteurs ont considéré un modèle basé sur une modification aléatoire des modules p et q. Le principe de l'analyse de telles perturbations est détaillé dans le paragraphe suivant.

Principe général. Considérons à présent un attaquant capable de perturber aléatoirement les modules DSA p et q avant de commencer la signature. Sous ces hypothèses, on peut exprimer les deux parties d'une signature DSA de la façon suivante :

$$\hat{u} \equiv \left(g^k \mod \hat{p}\right) \mod \hat{q} \quad \text{et} \quad \hat{v} \equiv \frac{\dot{m} + x\hat{u}}{k} \mod \hat{p} \tag{6.1}$$

Considérons à présent qu'il existe un entier t tel que $t \mid \hat{p}$ et $t \mid \hat{q}$, mais aussi que l'image de t par l'indicatrice d'Euler $\varphi(t) \mid \hat{q}$. Si le générateur g n'est pas une racine de zéro modulo t alors :

$$\frac{(\hat{u})^{\hat{v}}}{g^{\dot{m}}} \equiv \left(g^{\hat{u}}\right)^x \mod t \tag{6.2}$$

L'équation précédente met en évidence la possibilité de retrouver un résidu modulaire de la clé privée x par résolution d'un petit logarithme discret. En effet, pour tout diviseur r_j de l'ordre multiplicatif de $g^{\hat{u}}$ dans $(\mathbb{Z}/t\mathbb{Z})^*$, $x \mod r_j$ est solution pour le problème du logarithme discret engendré par (6.2). Par conséquent, si l'attaquant est capable de retrouver les valeurs erronées $(\hat{p}, \hat{q})$ résultant de la perturbation et, puisque $\frac{(\hat{u})^{\hat{v}}}{g^{\dot{m}}}$ et $\left(g^{\hat{u}}\right)$ peuvent se calculer à partir de la signatures erronée, alors il peut retrouver un résidu modulaire de la clé privée. Enfin, la clé privée

pourra être entièrement déterminée en répétant l'analyse précédente sur plusieurs signatures erronées suivant le modèle et ce, jusqu'à ce le plus petit multiple commun des r_j soit suffisamment grand (*i.e.* $\prod r_j \geqslant q$). La clé privée pourra alors être entièrement reconstruite grâce au théorème des restes chinois.

Si l'on se réfère au principe général de cette attaque, on peut remarquer que toutes les signatures erronées ne sont pas exploitables. En effet, pour pouvoir retrouver des résidus de clés par résolution de logarithme discret, il faut d'abord que l'attaquant puisse trouver un entier t divisant à la fois $\hat{p}$ et $\hat{q}$, mais encore que $\varphi(t) \mid \hat{q}$. Condition à laquelle il faut ajouter que l'ordre multiplicatif de $g^{\hat{u}}$ dans $(\mathbb{Z}/t\mathbb{Z})^*$ soit divisible par un entier r_j suffisamment petit pour qu'une résolution de logarithme discret soit possible. Toutes ces conditions influencent largement les performances de l'attaque, notamment en termes du nombre de perturbation à réaliser pour obtenir des signatures exploitables. À partir de leur étude statistique, les auteurs ont estimé que la recherche de clé privées DSA de 160 bits nécessite de réaliser plus de 200 millions de perturbations. Cependant ces estimations sont relativement pessimistes en comparaison des résultats obtenus expérimentalement : dans les mêmes conditions d'attaque, 40 millions de perturbations pourraient suffire. Malgré la réelle praticabilité du modèle de perturbation choisi, le nombre de perturbation nécessaire est conséquent ce qui relativise sa faisabilité pratique.

D'un point de vue de l'exploitation, cette attaque semble s'inspirer de celle proposée par Brier *et al.* contre les éléments publics de RSA [BCMCC06]. Néanmoins, le nombre de fautes à réaliser pour attaquer les implantations de signature RSA est bien moins important que celui proposé pour analyser des signatures DSA erronées [26]. Comme conclu par les auteurs de l'attaque, ces résultats laissent croire que la perturbation des éléments publics est moins efficace dans le cas des implantations d'algorithmes de signatures type El Gamal que pour les implantations de signatures RSA. Suite à cette conjecture, nous avons essayer d'attaquer les implantations de DSA sous les mêmes hypothèses de perturbation que celles choisies pour exploiter les perturbations d'éléments publics de RSA. Le résultat de cette étude est détaillé dans les sections suivantes.

6.2 Quelques outils mathématiques

Avant de commencer à présenter l'attaque par perturbation des éléments publics que nous avons étendue aux implantations de DSA, nous allons décrire les deux algorithmes que nous avons utilisés pour retrouver la clé privée DSA. La seconde partie de notre analyse repose sur les attaques réseaux précédemment décrites contre le DSA par N.A. Howgrave-Graham et N.P. Smart [HGS01] et plus récemment par P.Q. Nguyen et I.E. Shparlinski [NS02].

Dans un premier temps, nous présenterons l'*algorithme LLL* [Ajt98] permettant de transformer une base quelconque d'un réseau euclidien en une base réduite, et même *LLL-réduite*. Alors que les premières applications de ce type d'algorithme consistaient en la production d'un algorithme de factorisation des polynômes à coefficients rationnels en produits de polynômes irréductibles, nous utiliserons cet algorithme dans le but d'obtenir des bases adaptées à l'utilisation du second algorithme.

Justement, le second algorithme que nous présenterons est l'algorithme de Babai [Bab86]. Cet algorithme permet de trouver, en temps polynomial, le vecteur d'un réseau le plus proche d'un vecteur n'appartenant pas au réseau, à un facteur d'approximation près. En d'autre termes, cet algorithme permet de résoudre des instances approximatives du problème du vecteur le plus proche, réputé difficile. L'efficacité de cette algorithme dépend de la qualité de la réduction

26. 60000 fautes dans le cas du RSA [BCMCC06] contre près de 40 millions en considérant des tailles de clé similaire et un modèle de faute équivalent

opérée par l'algorithme LLL.

Comme nous le verrons dans la Section 6.3.2, l'utilisation combinée de ces deux algorithmes permettra d'exploiter directement les informations recueillies par le biais des perturbations.

6.2.1 Algorithme de réduction de réseau

Le principe de la réduction de réseau consiste à trouver une base engendrant le réseau de telle sorte que les vecteurs aient des normes assez petites et soient en même temps presque orthogonaux les uns aux autres. L'un des problèmes algorithmiques majeurs liés aux réseaux est la recherche du vecteur le plus court. Bien que ce problème soit prouvé NP-difficile [Ajt98], la notion de réduction introduite par A.K. Lenstra, H.W. Lenstra & L. Lovász [LLL86] apporte, en pratique, une réponse satisfaisante au problème de la recherche de vecteurs courts dans un réseau. Notons $\langle ., . \rangle$, le produit scalaire, $(b_i{}^*)_{1 \leqslant i \leqslant n}$ les orthogonalisés de Gram-Schmidt correspondant à une base $(b_i)_{1 \leqslant i \leqslant n}$ et $\mu_{i,j} = \langle b_i, b_j{}^* \rangle / \| b_j{}^* \|^2$. La notion de *LLL-réduction* est rappelée dans la définition suivante.

Définition 6.2.1.1. *(Base LLL-réduite) Une base $(b_i)_{1 \leqslant i \leqslant n}$ d'un réseau $\mathcal{L}$ est LLL-réduite avec un facteur $\delta \in]1/4, 1]$ si les deux propriétés suivantes sont vérifiées :*

1. $| \mu_{i,j} | \leqslant \frac{1}{2}$ *pour* $1 \leqslant j < i \leqslant n$
2. $\| b_i{}^* + \mu_{i,i-1} b_{i-1}^* \|^2 \geq \delta \| b_{i-1}^* \|^2$ *pour* $1 < i \leq n$

La première condition est appelé *condition de réduction*. Elle permet de s'assurer qu'à chaque étape i, le vecteur b_i est bien réduit par rapport aux orthogonalisés précédents $b_j{}^*$. La deuxième condition est la *condition de Lovász* qui garantit que la taille des vecteurs orthogonalisés $b_i{}^*$ ne croissent pas trop vite. En général, on fixe le paramètre δ égale à $3/4$.

À partir de cette définition de la *LLL-réduction* d'une base d'un réseau, on peut décliner l'algorithme de réduction de base, où *algorithme LLL*, permettant de transformer une base quelconque en une base *LLL-réduite*. Comme décrit dans [Bau08], cet algorithme repose sur l'utilisation des deux fonctions `RED` et `SWAP` suivantes :

1. `RED`(i, j). Cette fonction commence par vérifier la *condition de réduction* : $| \mu_{i,j} | \leqslant \frac{1}{2}$. Ensuite, si le vecteur b_i est déjà réduit par rapport à b_j, cette fonction ne fait rien. Sinon, b_i est remplacé par $b_i \leftarrow b_i - \lfloor \mu_{i,j} \rceil b_j$, où $\lfloor \cdot \rceil$ dénote l'entier le plus proche.
2. `SWAP`(i, δ). Cette fonction, quand à elle vérifie la *condition de Lovász* : $\| b_i{}^* + \mu_{i,i-1} b_{i-1}^* \|^2 \geq \delta \| b_{i-1}^* \|^2$. Lorsque la condition est réalisée, la fonction ne fait rien. Sinon, elle échange les vecteurs b_i et b_{i-1}

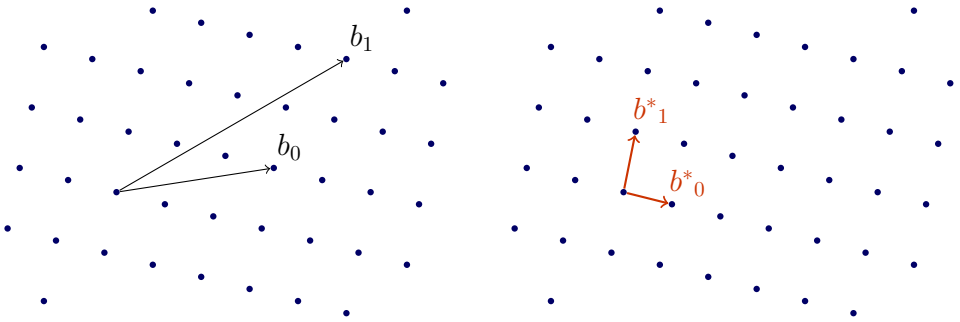
À partir de ces deux fonctions, le principe de la réduction de base LLL peut être décrit en pseudo-langage comme dans l'algorithme 8. La figure 6.1 illustre une réduction de base LLL appliquée à une base d'un réseau à deux dimensions.

L'avantage de cet algorithme est qu'il permet de trouver une base réduite en un temps polynomial. D'autres méthodes de réduction permettent de trouver des bases de meilleure qualité. Par exemple la réduction de Hermite-Korkine-Zolotarev garantit que le premier vecteur de base atteint le premier minimum du réseau. Cependant, les réductions fournissant des bases de grande qualité sont très difficiles à obtenir en pratique puisque les calculer est un problème au minimum NP-difficile pour les réductions randomisées. Une description plus complète de l'algorithme LLL peut être trouvée dans [Coh93] avec de nombreuses variantes. Dans le cadre de l'implantation de notre attaque par perturbation contre les éléments publics de DSA, nous avons choisi d'utiliser la méthode *Block Korkine-Zolotarev* (BKZ) disponible dans la librairie `NTL` de V. Shoup [Sho].

Algorithme 8 Algorithme LLL de réduction de réseau

ENTRÉES : Une base $(b_i)_{1\leqslant i\leqslant n}$ d'un réseau L et $\delta \in (\frac{1}{4}, 1)$
SORTIE : une base LLL-réduite $(b_i)_{1\leqslant i\leqslant n}$

$i \leftarrow 2$
Tant que $i \leqslant n$ **faire**
  RED$(i, i-1)$
  SWAP(i, δ)
  Si Condition de Lovász vérifiée **alors**
    RED$(i, i-2)$, ...,RED$(i, 1)$
    $i \leftarrow i + 1$
  sinon
    $i \leftarrow max(2, i-1)$
  Fin Si
Fin Tant que
Retourner A



(a) Exemple de réseau de 2 dimensions (b) Base réduite retournée par l'algorithme LLL

FIGURE 6.1 – Illustration de l'application de l'algorithme LLL

6.2.2 Problème du vecteur le plus proche (CVP)

L'algorithme de Babai, aussi connu sous le nom de *"Nearest Plane Algorithm"*, est un algorithme développé par L. Babai en 1986 pour résoudre des approximations d'instances du problème du vecteur le plus proche [Bab86]. Plus précisément, le but de l'algorithme est de trouver, étant donné un réseau $\mathcal{L}$ engendré par une base $(b_i)_{1\leqslant i\leqslant n}$ et un vecteur $u \notin \mathcal{L}$, le vecteur $v \in L$ le plus proche possible de u. C'est à dire que :

$$\forall w \in \mathcal{L}, \| v - w \| < \gamma \| u - w \| \tag{6.3}$$

où, $\gamma \geqslant 1$ est appelé le facteur d'approximation. Lorsque $\gamma = 1$ alors on retrouve une solution exacte pour CVP. L'algorithme de Babai permet de résoudre, en un temps polynomial, une approximation d'instance CVP pour un facteur d'approximation $\gamma = 2\left(\frac{2}{\sqrt{3}}\right)^n$ où n dénote la dimension du réseau. Dans la suite, nous présenterons la version de l'algorithme permettant de résoudre les approximations d'instances de CVP pour $\gamma = 2^{\frac{n}{2}}$. L'autre facteur d'approximation peut être atteint en réalisant quelques modifications.

On suppose que la base en entrée de l'algorithme est une base réduite (*e.g.* par l'algorithme LLL avec un facteur $\delta = 3/4$). Ensuite, le principe de l'algorithme de Babai peut se décomposer

en deux grandes étapes. La première consiste à orthogonaliser la base réduite par la méthode d'orthogonalisation de Gramm-Schmidt. Très brièvement, la seconde phase consiste à chercher itérativement les différents hyperplans les plus proches du vecteur u, projeté dans la base réduite. Ce principe est décrit dans l'algorithme 9 et le résultat de la recherche du vecteur le plus proche est illustré par la figure 6.2. Nous laisserons le lecteur intéressé se référer à [Bab86] pour plus de détails sur l'algorithme.

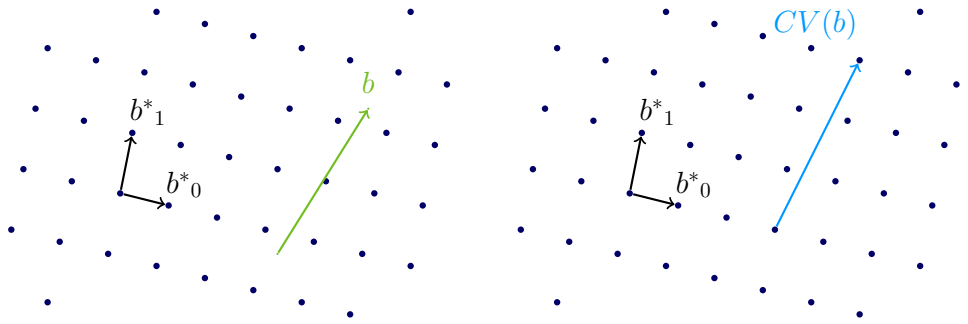
Pour l'implantation de notre attaque, nous avons de choisi d'utiliser la routine `NearVector`

Algorithme 9 Algorithme de Babai

ENTRÉES : Une base $(b_i)_{1 \leqslant i \leqslant n}$ LLL-réduite d'un réseau $\mathcal{L}$ et un vecteur $u \notin \mathcal{L}$.
SORTIE : $v \in \mathcal{L}$ tel que $\forall w \in \mathcal{L} \parallel v - w \parallel < 2^{\frac{n}{2}} \parallel u - w \parallel$

1: {Orthogonalisation de Gramm-Schmidt de la base}
2: $({b_1}^*, \ldots, {b_n}^*) \leftarrow GS\_Orthogonalisation(b_1, \ldots b_n, 3/4)$
3: $b \leftarrow u$
4: **Pour** $i = n$ **à** 1 **faire**
5: $\quad c_i \leftarrow \langle b, {b_i}^* \rangle / \parallel {b_i}^* \parallel^2$
6: $\quad b \leftarrow b - \lfloor c_i \rceil b_i$
7: **Fin Pour**
8: **Retourner** $v = u - b$

fournie par la librairie `NTL` de V. Shoup [Sho]. Par ce biais, nous avons été en mesure de retrouver, dans la majorité des cas, les clés privées DSA utilisées pour réaliser les signatures perturbées.



(a) Exemple de vecteur n'appartenant pas à un réseau à deux dimensions

(b) Vecteur le plus proche retourné par l'algorithme de Babai

FIGURE 6.2 – Illustration de l'application de l'algorithme de Babai

6.3 Attaque DSA

6.3.1 Perturbation d'un module public de DSA

6.3.1.1 Modèle de faute

Le modèle de perturbation que nous avons choisi pour réaliser notre attaque s'inspire directement de celui que nous avons utilisé pour attaquer les implantations standard de RSA (*cf.* Sections 4.2 et 4.3). En effet, nous allons considérer un attaquant capable d'injecter une faute transitoire sur le paramètre public p, à la t-ième itération du calcul de la première partie de la

signature DSA : u (*cf.* Section 1.3.3). Dans un souci de clarté, nous supposerons que l'exponentiation modulaire utilisée pour ce calcul est implantée avec un algorithme de type *"Right-To-Left"* (*cf.* Algorithme 3). Nous détaillerons dans la section 6.3.3, le cas où une implantation duale est attaquée. Comme pour les attaques présentées précédemment, nous considérons qu'un octet de p peut être modifié aléatoirement et que la valeur du module perturbé $\hat{p}$ n'est pas connue de l'attaquant. Pour rappel, la valeur du module ainsi perturbé peut être encore exprimée :

$$\hat{p} = p \oplus \varepsilon \tag{6.4}$$

où $\varepsilon = R_8 \cdot 2^{8i}, i \in [\![0; \frac{log_2(p)}{8} - 1]\!]$ et R_8 est une valeur aléatoire, non nulle, sur un octet. D'après les résultats déjà montrés dans le cadre de notre attaque par perturbation des éléments publics de RSA, on peut noter que ce modèle de perturbation peut être étendu à la perturbation d'une seule opération pendant l'exécution (carré ou multiplication modulaire). Par ailleurs, il est aussi possible d'analyser ces perturbations pour des algorithmes déclinés de la version *"Right-To-Left"* comme les algorithmes Fixed/Sliding windows ou la contre-mesure SPA *"Square & Multiply Always"* (*cf.* Section 1.3.4.3).

Discussion. En comparaison avec les précédentes attaques par perturbation contre le DSA, notre modèle de perturbation semble beaucoup moins restrictif que celui adopté dans l'attaque de D. Naccache *et al.* [NNTW05]. En effet, ceux-ci ont considéré un attaquant capable de forcer la valeur de certains octets de l'aléa k à 0. Bien que la reproductibilité de leur modèle ait été étudiée, de telles fautes semblent quand même plus difficiles à obtenir. En revanche, notre modèle de perturbation paraît moins général que celui adopté par C.H. Kim *et al.* dans le sens où ilé repose sur une modification aléatoire de modules. Mais, bien que leur modèle semble plus facile à reproduire, le nombre de perturbations que requiert l'attaque représente un sérieux inconvénient. Par conséquent, en comparaison avec les autres attaques par perturbation contre le DSA, notre attaque semble offrir le meilleur compromis entre praticité du modèle et nombre de signatures à perturber.

6.3.1.2 Exemple d'exécution perturbée

Dans cette section, nous allons détailler comment une faute, commise suivant notre modèle, se propage jusque dans la signature DSA renvoyée. Soit $k = \sum_{i=0}^{\kappa-1} 2^i \cdot k_i$ l'aléa tiré pour la signature DSA avec $\kappa = log_2(q)$ et $k < q$. Nous rappelons que le module public p est perturbé à la t-ième itération pendant le calcul de u par une exponentiation modulaire de type *"Right-To-Left"*. Par conséquent, si la première opération perturbée est un carré modulaire, le registre interne B prendra comme valeur :

$$\hat{B} \equiv \left(g^{2^{t-1}} \mod p\right)^2 \mod \hat{p} \tag{6.5}$$

Puis, à la première exécution d'un multiplication modulaire, cette erreur se propage dans l'autre registre A. À la fin d'une exponentiation perturbée, la première partie de la signature DSA $\hat{u}$ pourra être exprimée ainsi :

$$\hat{u} \equiv \left(((A \cdot \hat{B})...)\hat{B}^{2^{\kappa-t-1}} \mod \hat{p}\right) \mod q \tag{6.6}$$

$$\equiv \left(A \cdot \hat{B}^{\frac{k_{[t]}}{2^t}} \mod \hat{p}\right) \mod q \tag{6.7}$$

où $k_{[t]} = \sum_{i=t}^{\kappa-1} 2^i \cdot k_i$. Évidemment, la seconde partie de la signature v est aussi infectée par la perturbation :

$$\hat{v} \equiv \frac{\dot{m} + x\hat{u}}{k} \mod q \tag{6.8}$$

Comme pour notre première attaque sur le RSA, nous remarquons que l'injection d'une faute pendant une exponentiation isole la partie d'exposant $k_{[t]}$ balayée dans la partie infectée du calcul. Ainsi, en adaptant la méthode déjà proposée dans la partie 4.2, il sera possible de retrouver efficacement ce morceau d'exposant. Cependant, contrairement à l'algorithme de signature RSA, ce n'est pas la clé privée qui est utilisée dans l'exponentiation modulaire. Par conséquent, retrouver $k_{[t]}$ qui est un morceau d'aléa n'apporte, a priori, aucune information sur la clé privée DSA : x. Mais puisque l'aléa est aussi utilisé dans le calcul de la seconde partie de la signature, nous verrons par la suite comment utiliser la connaissance de ces morceaux d'aléas pour retrouver la clé secrète x.

6.3.2 Analyse des perturbations

L'analyse différentielle de perturbations que nous avons proposée précédemment contre les implantations *"Right-To-Left"* de l'algorithme de signature RSA utilise la différence entre une signature correcte et erronée. Or, dans le cas de signature DSA, il n'est pas possible de d'obtenir un tel couple. En effet, les signatures DSA sont réalisées avec un aléa k rendant chaque signature différente (*cf.* Section 1.3.3). Dans la partie suivante, nous allons montrer qu'en pratique, la seule connaissance d'une signature DSA erronée $(\hat{u}, \hat{v})$ est suffisante pour retrouver les bits de poids fort d'aléa. Enfin, nous nous inspirons des travaux sur les attaques réseau afin d'extraire la clé privée à partir de plusieurs morceaux d'aléa [HGS01, NS02].

6.3.2.1 Vers l'extraction d'un morceau d'aléa

La première partie de l'analyse de signatures DSA perturbées consiste à essayer de retrouver les morceaux d'aléas $k_{[t]}$ isolés par l'injection de faute. Or, comme nous l'avons souligné, cette analyse nécessite de connaître une signature erronée mais aussi la signature correcte correspondante ce qui n'est pas possible dans le cas du DSA. Mais, en utilisant les propriétés de vérification de signature il est quand même possible de retrouver un u correct correspondant à la signature erronée obtenue $(\hat{u}, \hat{v})$. La méthode est décrite dans le paragraphe suivant.

Obtention d'un u correct. En effet, à partir de l'astuce utilisée pour vérifier des signatures DSA, il est possible d'exraire un u correct depuis une signature erronée $(\hat{u}, \hat{v})$. Si $\hat{v}$ est inversible dans $\mathbb{Z}_q^*$, on pose $\hat{w} \equiv \hat{v}^{-1} \mod q$ puis :

$$\begin{aligned} \left(g^{\hat{w}\dot{m}} \cdot y^{\hat{w}\hat{u}}\right) \mod p &\equiv g^{\hat{w}(\dot{m}+x\hat{u})} \mod p \\ &\equiv g^k \mod p \\ &\equiv u_p \end{aligned} \tag{6.9}$$

À partir de u_p, on peut calculer simplement la partie de signature correcte u par la réduction modulaire $u \equiv u_p \mod q$. Ainsi, la seule connaissance du message d'entrée et de la signature DSA erronée associée $(\hat{u}, \hat{v})$ permet de trouver rapidement le résultat correct d'une exponentiation modulaire à la puissance k. La seule contrainte à satisfaire est que $gcd(\hat{v}, q) = 1$ ce qui est toujours le cas puisque q est un nombre premier et $\hat{v} < q$. L'attaquant peut donc alors utiliser la différence existant entre $\hat{u}$ et u pour retrouver un morceau d'aléa. La méthode que

nous avons utilisée s'inspire largement de notre analyse réalisée sur la perturbation du module public d'implantations *"Right-To-Left"* de la signature RSA et est rappelée ci-dessous dans le cadre de DSA.

Extraction d'un morceau de k. Ici, le but de l'attaquant est de retrouver une partie des bits de poids fort de l'aléa k qui a été isolé par la perturbation du module p. Or, l'attaquant sait que la perturbation intervient à la t-ième itération de l'exponentiation et que cette faute correspond à une modification aléatoire d'un octet de p. Par conséquent, il essaie d'utiliser ces informations pour retrouver simultanément les valeurs de $k_{[t]}$ et $\hat{p}$. Plus précisément, l'attaquant choisit une paire de valeurs candidates $(k'_{[t]}, \hat{p}')$ pour calculer d'abord :

$$A_{k'_{[t]}} \equiv u_p \cdot g^{-k'_{[t]}} \mod p \tag{6.10}$$

On peut remarquer que l'on utilise pas exactement u dans (6.10). En effet, puisqu'il est possible de retrouver $u_p \equiv g^k \mod p$ grâce l'astuce précédente, on préfère utiliser cette valeur non réduite pour *"remonter"* l'exécution. Ensuite, l'attaquant utilise le couple de valeurs candidates $(k'_{[t]}, \hat{p}')$ pour simuler un calcul de signature erronée, en accord avec le modèle :

$$\hat{u}_{(k'_{[t]},\hat{p}')} \equiv \left(A_{k'_{[t]}} \cdot \left(g^{2^{t-1}} \mod p \right)^{2 \cdot \frac{k'_{[t]}}{2^t}} \mod \hat{p}' \right) \mod q \tag{6.11}$$

Enfin, pour valider la pertinence des valeurs candidates qu'il a choisi pour $(k_{[t]}, \hat{p})$, l'attaquant compare cette signature erronée probable $\hat{u}_{(k'_{[t]},\hat{p}')}$ avec celle retournée par le composant perturbé $\hat{u}$:

$$u'_{(k'_{[t]},\hat{p}')} \equiv \hat{u} \bmod q \tag{6.12}$$

Lorsque (6.12) est satisfaite, cela signifie que le couple de valeurs candidates $(k'_{[t]}, \hat{p}')$ est très probablement le bon (*cf.* Section 4.2.2, Théorème 4.2.2). Ainsi, l'attaquant identifie les $\kappa - t$ bits de poids fort de l'aléa k utilisés pour signer. Dans la suite nous utiliserons la taille de fenêtre w égale à $\kappa - t$ pour désigner le nombre de bits d'aléas retrouvés par ce biais.

Intrinsèquement, retrouver ces morceaux d'aléas n'a pas beaucoup d'intérêt puisqu'a priori, leur connaissance n'est pas censée apporter d'informations sur la clé privée DSA. Cela étant, ces aléas ne sont pas seulement utilisés pour générer la première partie de la signature u, mais ils sont aussi utilisés dans le calcul de v. Les *attaques réseaux* proposent justement d'exploiter cette partie linéaire du DSA pour retrouver la clé secrète DSA.

6.3.2.2 Extraction de la clé privée par réduction de réseau

L'objectif principal de cette partie est d'utiliser les parties d'aléa k, retrouvées comme précédemment, afin de donner une approximation de la clé privée DSA. Or, dans le cadre des algorithmes de signature type El Gamal, notre étude bibliographique a mis en évidence des méthodes existantes permettant d'extraire des clés privées à partir de morceaux d'aléas. Ces méthodes reposent sur le principe des attaques réseaux [HGS01, NS02], déjà utilisées pour exploiter des perturbations contrôlées d'aléas [NNTW05]. Par conséquent, nous décrirons brièvement, dans cette partie, comment nous avons utilisé le principe des attaques réseaux pour exploiter les perturbations d'éléments publics du DSA. Nous laisserons au lecteur intéressé le plaisir de parcourir les articles [HGS01, NS02] traitant plus en détail des attaques réseaux.

Le principe général des attaques réseaux est d'utiliser la linéarité induite par le calcul de la deuxième partie de la signature, à savoir pour une signature erronée :

$$\hat{v} \equiv \frac{\dot{m} + x\hat{u}}{k} \bmod q \tag{6.13}$$

Dans ce cas, la connaissance partielle de l'aléa k peut entraîner une fuite d'information sur la valeur de la clé privée x. Par conséquent, en analysant le résultat de plusieurs signatures perturbées suivant notre modèle, il pourrait être possible de déterminer complètement x.

Bits de poids fort "modulaires". Dans la section précédente, nous avons montré que, par une approche type *"guess-and-determine"*, il est possible de retrouver les bits de poids fort de k isolés par la perturbation du module public p pendant le calcul de u. Pour rappel, cette partie est notée $k_{[t]} = \sum_{i=t}^{\kappa-1} 2^i k_i$ et représente les w bits de poids fort de l'aléa k. Or, on peut noter que, si q est proche de $2^{\kappa-1}$, et comme $k \in {\mathbb{Z}_q}^*$, alors le bit de poids fort des aléas tirés sera, la plupart du temps, égal à 0. Dans ce cas, la quantité d'information effective apportée par la récupération de $k_{[t]}$ sera seulement de $w-1$ bits. Afin d'enrailler ce phénomène, il a été proposé dans [NS02] d'utiliser une caractérisation différente pour représenter les bits de poids fort d'un nombre. Cette caractérisation est basée sur la définition déjà proposée par D. Boneh et R. Venkatesan dans [BV96], puis reprise dans [NS02] sous le nom de *bits de poids fort modulaires*[27]. Suivant la définition proposée, les w bits de poids fort modulaires d'un élément $k \in \mathbb{Z}_q^*$ sont définis comme l'unique entier, noté $\mathrm{MSMB}_{w,q}(k)$, tel que :

$$0 \leq k - \mathrm{MSMB}_{w,q}(k)q/2^w < q/2^w \tag{6.14}$$

L'avantage de cette représentation est qu'elle permet d'exprimer un morceaux de k en fonction de q. Utilisons maintenant cette définition pour exprimer différemment la deuxième partie de l'une signature DSA erronée.

$$x\hat{u} \equiv k\hat{v} - \dot{m} \mod q \tag{6.15}$$

Si $\hat{v} \neq 0$, on peut alors écrire :

$$\begin{aligned} x\hat{u}\hat{v}^{-1} &\equiv k - \hat{v}^{-1}\dot{m} \mod q & (6.16)\\ \Leftrightarrow x\hat{u}\hat{v}^{-1} + \hat{v}^{-1}\dot{m} &\equiv k \mod q & (6.17) \end{aligned}$$

Afin d'utiliser l'inégalité encadrant k, on peut transformer l'équation précédente en égalité. En effet, $\exists \lambda \in \mathbb{Z}$ tel que :

$$x\hat{u}\hat{v}^{-1} + \hat{v}^{-1}\dot{m} = k + \lambda q \tag{6.18}$$

Ainsi, en utilisant (6.14), on obtient finalement :

$$|x\hat{u}\hat{v}^{-1} + \dot{m}\hat{v}^{-1} - \mathrm{MSMB}_{w,q}(k)q/2^w - q/2^{w+1} - \lambda q| \leq q/2^{w+1} \tag{6.19}$$

Pour clarifier les choses, posons deux variables définies uniquement par des éléments connus (ou retrouvés) par l'attaquant :

$$\begin{aligned} a &= \hat{u}\hat{v}^{-1} \\ b &= \mathrm{MSMB}_{w,q}(k)q/2^w - \dot{m}\hat{v}^{-1} + q/2^{w+1} \end{aligned}$$

27. Traduction littérale de *Most Significant Modular Bits*

Cela permet de faire ressortir l'approximation :

$$|xa - b - \lambda q| \leq q/2^{w+1} \tag{6.20}$$

À partir de (6.20), l'attaquant déduit une approximation de xa modulo q par la valeur b. On pourra d'ailleurs remarquer que plus la différence w est grande et plus l'approximation sera précise. Ce résultat n'est pas surprenant car cette différence correspond à la quantité d'information retrouvée sur la valeur de l'aléa. Cela étant, une seule approximation ne peut suffire pour déterminer précisément x. Aussi, pour réaliser une attaque réseau, il faudra que l'attaquant collecte suffisamment de signature erronées pour obtenir autant d'approximations de x que nécessaire.

Supposons à présent que l'attaquant ait réussi à collecter η signatures erronées $(\hat{u_i}, \hat{v_i})_{1 \leq i \leq \eta}$. Pour chacune d'entre elles, nous supposons que l'attaquant a réussi à extraire les w bits de poids fort des aléas k_i correspondants, en appliquant la méthode décrite précédemment. À partir de ces morceaux d'aléas, des valeurs de signatures erronées et des données publiques de DSA, l'attaquant peut alors calculer les a_i et b_i telles que, $\exists \lambda \in \mathbb{Z}$:

$$|xa_i - b_i - \lambda_i q| \leq q/2^{w+1} \tag{6.21}$$

Dans la partie suivante, nous allons voir comment utiliser toutes les approximations ainsi obtenues pour retrouver la clé privée x.

Construction et réduction de réseaux. Le problème consistant à retrouver la clé secrète x par les approximations précédentes a été formalisé par D. Boneh et R. Venkatesan sous le nom de *problème du nombre caché*[28] [BV96](HNP). Pour résoudre un tel problème, l'astuce consiste à le transformer en un problème de *"vecteur le plus proche"*[29] (CVP) dans un réseau [BV96, NS02]. En effet, une approximation de ce problème peut être résolu en temps polynomial par l'algorithme de Babai [Bab86].

Commençons par considérer un réseau $\mathcal{L}$ engendré par les vecteurs colonnes de la matrice suivante :

$$\begin{pmatrix} q & 0 & \cdots & 0 & a_1 \\ 0 & q & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & q & a_\eta \\ 0 & \cdots & \cdots & 0 & 1/2^{w+1} \end{pmatrix} \in M_{(\eta+1),(\eta+1)}(\mathbb{Q}) \tag{6.22}$$

Ainsi, le vecteur $\alpha = (xa_1 + \lambda_1 q, \ldots, xa_\eta + \lambda_\eta q, x/2^{w+1})$ appartient au réseau car il peut s'exprimer comme une combinaison linéaire des vecteurs de base :

$$\alpha = \sum_{i=1}^{\eta} \lambda_i L_i + x L_{\eta+1} \tag{6.23}$$

où L_i correspond au i-ème vecteur de la matrice. Ce vecteur α est aussi appelé *vecteur caché* car sa dernière coordonnée (*i.e.* $x/2^{w+1}$) dépend directement du *nombre caché* x.

En pratique, on ne sait pas construire un tel vecteur car ses coordonnées dépendent de valeurs inconnues par l'attaquant (*i.e.* que ce soit les λ_i et surtout x). Aussi, à partir des différentes relations d'approximation tirées des signatures DSA erronées (*cf.* Équation 6.21), ce vecteur

28. Traduction littérale de *Hidden Number Problem*
29. Traduction littérale de *Closest Vector Problem*

caché α est approché par un vecteur $\beta = (b_1, \ldots, b_\eta, 0)$. La principale particularité de β est que ce vecteur n'appartient pas au réseau. Cependant ses coordonnées peuvent être déterminées par un attaquant puisqu'elles ne dépendent que des éléments publics du système où des parties d'aléas $(k_{i[t]})_{1 \leq i \leq \eta}$ retrouvées par la première analyse. Sachant que β représente une approximation d'α, qui lui appartient à $\mathcal{L}$, le principe de l'attaque réseau consiste à chercher le vecteur du réseau $\mathcal{L}$ le plus proche de β. Et quand la base engendrant le réseau est une base LLL-réduite, il est possible de trouver un tel vecteur en temps polynomial, grâce à l'algorithme de Babai [Bab86, Sho]. Une fois le vecteur de $\mathcal{L}$ le plus proche de β déterminé, l'attaquant n'a plus qu'à dériver une valeur potentielle pour x depuis la dernière coordonnée du vecteur trouvé. Enfin, pour vérifier que la valeur potentielle d'exposant obtenue x' est la bonne, il vérifie si l'équation suivante est satisfaite :

$$g^{x'} \equiv y \bmod p \tag{6.24}$$

Le taux de réussite de l'attaque présenté dépend fortement du nombre de signature erronées collectées η ainsi que du nombre de bits d'aléa w qu'il est possible d'extraire de chacune de ces signatures. Selon [NS02, NNTW05], $\eta \approx \kappa/w$ signatures erronées peuvent suffire pour retrouver x avec une bonne probabilité (*i.e.* rappelons que κ correspond au $log_2(q)$). Nos résultats expérimentaux ont confirmé cette évaluation théorique. Pour finir, on peut noter qu'il est possible d'exploiter des signatures perturbées à différentes instants t_i de leur exécution respective. Dans ce cas, le nombre de signatures erronées à obtenir est de l'ordre de $\eta \approx l/\min_i(w_i)$ pour espérer réussir l'attaque réseau.

6.3.2.3 Performances

Afin de comparer les performances de notre attaque par perturbation des éléments publics de DSA avec celles de l'état de l'art, nous avons évalué sa complexité de manière théorique. Les résultats obtenus par cette étude ont ensuite été vérifiés par notre propre implantation de l'attaque.

Estimations théoriques. Le principal inconvénient de la précédent attaque par perturbation des éléments publics de DSA est certainement le nombre de perturbations qu'elle requiert [KBPJJ08]. Afin de souligner l'importance de l'amélioration que nous proposons, commençons par estimer le nombre de perturbations $\mathcal{F}$ à répéter pour espérer réussir notre attaque. Selon le modèle de perturbation que nous avons considéré (*cf.* Section 6.3.1), chaque signature erronée est potentiellement exploitable. En d'autre termes, chaque signature erronée va permettre de retrouver quelques bits d'aléas. Par conséquent, $\mathcal{F}$ peut être estimé par :

$$\mathcal{F} = \mathcal{O}\left(\frac{\kappa}{w}\right) \tag{6.25}$$

où w correspond au nombre minimal de bits d'aléa retrouvés à partir d'une signature erronée. Pour un DSA de 160 bits et en fixant $w = 10$ (ce qui est un choix raisonnable si l'on se réfère aux résultats de la table 6.1), 16 signatures DSA erronées peuvent suffire pour retrouver rapidement la clé secrète. Cependant, ce nombre de perturbations peut être réduit théoriquement à $\log(\kappa)$ d'après [NS02]. En guise de comparaison, la meilleure attaque par perturbation contre les éléments publics de DSA requiert en moyenne $4 \cdot 10^7$ signatures erronées en pratique ($2 \cdot 10^8$ selon l'analyse théorique) [KBPJJ08]. Cette différence significative s'explique d'une part, par la différence de méthode employée et, d'autre part, car notre analyse permet d'exploiter toutes les signatures perturbées suivant le modèle et apportent donc une certaine quantité d'information

sur la clé privée.

En ce qui concerne la complexité calculatoire, celle-ci dépend à la fois de l'extraction de morceaux d'aléas depuis le signatures erronées ainsi que de la recherche d'une solution approché de CVP. Or, la complexité de la réduction LLL dépend fortement du nombre de vecteurs engendrant le réseau. Ce nombre est lui même, étroitement lié avec la taille des morceaux d'aléas w que l'on peut extraire des signatures erronées. Si l'on considère que $w \approx 10$ ce qui est avantageux en terme de nombre de signatures à perturber, alors c'est la complexité de l'extraction de morceaux d'aléa qui dominera la complexité totale de l'attaque. Dans ce cas, celle-ci peut être exprimée par :

$$\mathcal{C} = \mathcal{O}(\frac{2^w \kappa \log_2(p)}{w})\text{exponentiations} \tag{6.26}$$

En revanche, choisir des petites tailles pour w diminue la quantité d'information exploitable sur x via la relation (6.20) et, par la même, augmente considérablement le nombre de relations d'approximation. Dans ce cas, ce sera la réduction LLL qui sera l'opération critique de l'attaque.

Résultats expérimentaux. Afin de valider notre estimation théorique et véritablement trancher sur la praticité réelle de notre attaque, nous avons implanté toute la méthode décrite dans la Section 6.3.2. Nous avons alors généré différentes signatures DSA erronées, dont les perturbations ont été simulées suivant notre modèle (*cf.* Section 6.3.1). Ensuite, pour mettre en place l'attaque réseau, nous avons choisi d'utiliser l'algorithme de réduction de base *Block Korkin-Zolotarev* (BKZ) [SE94] et l'algorithme de Babai [Bab86], tous deux fournis par la librairie C++ `NTL` [Sho]. Les résultats expérimentaux détaillés plus bas ont été obtenus sur un PC Linux (Red Hat Scientifique et `gcc 4.1.2`) embarquant un processeur Intel Core 2 Duo cadencé à 2.66 GHz.

Dans un premier temps, nous avons mesuré le temps nécessaire à l'extraction de morceaux d'aléas à partir des signatures erronées, et ce, pour différentes tailles w. Le temps moyen d'extraction de morceaux d'aléas a été calculé en mesurant les résultats de 100 exécutions. Ces mesures de temps sont présentées dans la table 6.1, pour différentes valeurs de w.

TABLE 6.1 – Temps moyen d'extraction de w bits d'aléa pour un DSA de 160 bits

Taille des morceaux d'aléas w	4 bits	5 bits	6 bits	8 bits	10 bits
Temps moyen	16 s	33 s	1 min	4 min 30 s	17 min

Les résultats ainsi obtenus mettent en évidence la dépendance exponentielle existant entre le temps d'exécution et le nombre de bits à extraire. Par conséquent, fixer le nombre de bits que l'attaquant peut espérer retrouver, à partir de chaque signature erronée, est un compromis entre temps d'exécution et nombre de perturbations à répéter. Ce paramètre devra donc être minutieusement choisi en fonction des capacités de l'attaquant. Pour finir, on pourra noter que, puisque les différentes extractions de morceaux d'aléas sont indépendantes, elle pouront être parfaitement paralléliser ce qui permet d'améliorer encore les performances d'attaque en termes de temps d'exécution.

Pour compléter l'analyse de performances de notre attaque par perturbation, nous avons aussi mesuré l'efficacité de l'attaque réseau. Dans les conditions de tests que nous avons décrites en amont, quelques secondes suffisent pour retrouver une clé DSA de 160 bits, en connaissant des morceaux d'aléas de taille 10 bits extraits de 16 signatures erronées. De telles performances

soulignent le caractère pratique de notre analyse, même sur un PC classique.

En ce qui concerne le taux de réussite de l'attaque, celui-ci dépend à la fois du nombre

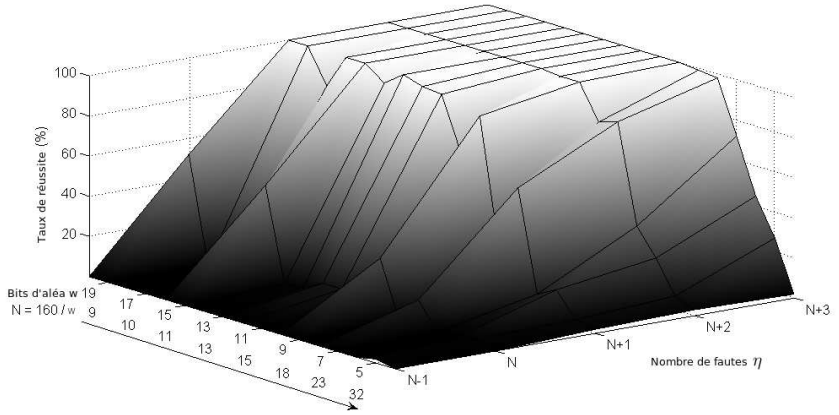


FIGURE 6.3 – Taux de réussite de l'attaque réseau en fonction de la taille de fenêtre w et du nombre de fautes η

de signatures erronées dont on dispose, mais aussi du nombre de bits w qu'il est possible d'en extraire. Ce dernier paramètre peut être aussi interprété comme l'indice de précision des approximation du vecteur caché α. La figure 6.3 présente le taux moyen de réussite de l'attaque en fonction des deux paramètres précédents. Celle-ci met en évidence que si η est égale à $\mathcal{F} = \frac{160}{w}$ et si le nombre de bits d'aléa retrouvés est petit, alors le nombre d'approximations déduites ne sera pas suffisant pour converger avec certitude vers la clé privée DSA. Par exemple, si l'attaquant choisi de retrouver 6 bits d'aléas, car cela ne prend qu'une minute en pratique (*cf.* Table 6.1), alors $160/6 = 26$ signatures erronées ne permettent de retrouver la clé que dans 1% des cas. En revanche, avec 29 signatures erronées, soit seulement 3 de plus, le taux de réussite passe à 45%. Enfin d'un point de vue du taux de réussite, plus le nombre de fautes et le nombre de bits retrouvés sont grands et mieux l'attaque marche. Par contre, elle mettra plus de temps à s'exécuter. Par conséquent, le choix de ces deux paramètres devra être fait de concert et en sachant que cela représente un compromis entre le temps d'exécution, les ressources matérielles et les capacités de l'attaquant.

6.3.3 Extensions de l'attaque

Cas des exponentiations modulaires *"Left-To-Right"*. Dans les sections précédentes, nous avons montré comment il était possible d'extraire des clés DSA à partir de signatures erronées, pour les implantations *"Right-To-Left"* de l'exponentiation. Nous décrirons ici la méthode à utiliser dans le cas des implantations *"Left-To-Right"* et, nous soulignerons les modifications mineures à effectuer pour l'analyse.

Commençons par montrer les effets d'une perturbation du module public à t itérations de la fin de l'exponentiation. En suivant l'étude déjà réalisée dans le cadre de RSA, une signature DSA erronée suivant le modèle peut être exprimée comme suit :

$$\begin{cases} \hat{u} \equiv \left(A^{2^t} \cdot g_{[t]}^k \mod \hat{p} \right) \mod q \\ \hat{v} \equiv \dfrac{\dot{m} + x\hat{u}}{k} \mod q \end{cases} \tag{6.27}$$

où $k_{[t]} = \sum_{i=0}^{t-1} 2^i \cdot k_i$ sont les bits de l'aléa k isolés par la perturbation et $A \equiv g^{\sum_{i=t}^{\kappa-1} 2^{i-t} \cdot k_i} \mod p$. D'après l'équation précédente, on peut remarquer que, cette fois ci, ce sont les bits de poids faible de l'aléa qui sont isolés. Aussi, ce seront ces bits qui devront être cherchés. Pour ce faire, l'attaquant pourra utiliser la méthode déjà proposée pour attaquer le RSA (*cf.* Section 4.3.3.1).

Remarque 5. *On pourra d'ailleurs noter que, puisque p est un module premier, on pourra essayer de "recréer" une signature erronée à partir de la signature correcte plutôt que de faire l'inverse, comme pour l'analyse présentée dans le cas du RSA. Ainsi, chaque signature erronée pourra être exploitée (possibilité de calculer des racines carrées modulaires car p est premier) ce qui permet de garder les même performances en termes de nombre de perturbations à réaliser. En revanche, la nouvelle complexité calculatoire de notre attaque devra tenir compte du calcul de racines carrées modulaires induit par l'utilisation d'une exponentiation modulaire de type "Left-To-Right".*

Une fois ces morceaux d'aléas extraits de plusieurs signatures DSA perturbées, ceux-ci seront utilisés pour approcher la valeur de la clé secrète x et construire le vecteur d'approximation. En effet, la connaissance de w bits de poids faible de l'aléa k, permet d'utiliser la linéarité du calcul de $\hat{v}$:

$$x\hat{u} \equiv k\hat{v} - \dot{m} \mod q \tag{6.28}$$

En décomposant k suivant la partie que l'attaquant vient de retrouver $k_{[w]}$ et une partie inconnue $r \geq 0$, $k = r2^w + k_{[w]}$, on peut encore écrire :

$$x\hat{u}\hat{v}^{-1}2^{-w} \equiv \left(k_{[w]} - \hat{v}^{-1}\dot{m}\right)2^{-w} + r \mod q \tag{6.29}$$

Comme pour le cas des implantations *"Right-To-Left"* de la signature DSA, on peut définir deux variables :

$$\begin{aligned} a &= \hat{u}\hat{v}^{-1}2^{-W} \mod q \\ b &= \left(k_{[w]} - \hat{v}^{-1}\dot{m}\right)2^{-w} \mod q \end{aligned}$$

Puis, en substituant dans (6.29) :

$$xa - b \equiv r \mod q \tag{6.30}$$

Or, puisque $0 \leq r \leq q/2^w$, $\exists \lambda \in \mathbb{Z}$ tel que :

$$0 \leq xa - b - \lambda q \leq q/2^w \tag{6.31}$$

On obtient finalement :

$$|xa - b - q/2^{w+1} - \lambda q| \leq q/2^{w+1} \tag{6.32}$$

L'équation précédente nous donne donc une approximation sur w bits de la clé DSA. Comme pour l'analyse précédente, on utilisera le coefficient a, calculé pour chacune des signatures erronées obtenues, pour calculer le réseau et le coefficient $b' = b + q/2^{w+1}$ appartiendra au vecteur d'approximation. La clé secrète sera alors retrouvée en combinant l'utilisation de l'algorithme LLL [LLL86] et de l'algorithme de Babai [Bab86].

Application aux signatures El Gamal. Le principe de l'attaque par perturbation du module public de DSA peut tout aussi s'appliquer à l'algorithme de signature El Gamal. En effet, sous les même hypothèses de perturbation (*cf.* Section 6.3.1) et en considérant une implantation de type *"Right-To-Left"* de l'exponentiation modulaire, on obtient :

$$\hat{u} \equiv A \cdot \hat{B}^{\frac{k_{[w]}}{2^w}} \mod \hat{p} \tag{6.33}$$

$$\hat{v} \equiv \frac{\dot{m} + x\hat{u}}{k} \mod p-1 \tag{6.34}$$

où $k_{[w]} = \sum_{i=w}^{\kappa-1} 2^i \cdot k_i$. Contrairement au DSA, aucune réduction modulaire supplémentaire (*i.e.* modulo q) n'est réalisée avec El Gamal. D'autre part, La deuxième partie de la signature est calculée modulo $p-1$, aussi, l'approximation de la clé secrète x sera réalisée avec ce facteur :

$$|xa - b - (p-1)/2^{w+1} - \lambda(p-1)| \leq (p-1)/2^{w+1} \tag{6.35}$$

$$\begin{aligned} a &= \hat{u}\hat{v}^{-1} \\ b &= \mathrm{MSMB}_{w,p-1}(k)(p-1)/2^w - \dot{m}\hat{v}^{-1} \end{aligned}$$

De manière analogue, on peut construire la matrice du réseau ainsi que le vecteur d'approximation à partir de plusieurs d'approximations de la clé. La clé est enfin retrouvée en cherchant le vecteur appartenant au réseau le plus proche du vecteur d'approximation. Notons enfin que, comme les approximations sont réalisées avec $p-1$ (*i.e.* au lieu de q), les performances d'attaques proposées dans le cadre du DSA devront être adaptées en conséquence.

6.4 Conclusion

Dans le chapitre précédent, nous avons proposé d'étudier la vulnérabilité des éléments publics de l'algorithme de signature DSA vis-à-vis des perturbations. Bien que la perturbation des éléments publics soit une thématique déjà étudiée pour les implantations de crypto-systèmes basées sur les courbes elliptiques ou pour le RSA, les premières applications contre les algorithmes de signatures de type El Gamal sont plutôt récentes.

En s'inspirant de l'attaque de Brier *et al.* contre les implantations de signatures RSA, C.H. Kim *et al.* ont proposé une application contre les implantations du DSA en perturbant aléatoirement les modules publics p et q. Bien que ce modèle de perturbation ait été choisi pour la facilité relative avec laquelle il peut être reproduit, le nombre de perturbations à réaliser apparaît comme un sérieux inconvénient. Pour un modèle de perturbation équivalent, il faut près de 40 millions de signatures erronées pour retrouver une clé privée DSA là où il fallait seulement 60000 signatures RSA pour l'attaque de Brier *et al.*. Ces résultats ont amené les auteurs à supposer que les implantations de signatures de type El Gamal pourraient être plus résistantes face à la perturbation de leurs éléments publics.

En remarquant que cette première application exploitait des perturbations infectant entièrement la signature DSA, nous avons tenté de transposer notre attaque contre les implantations de RSA. Cette tentative a aussi été motivée par la possibilité d'élargir l'étude des implantations du DSA aux perturbations des éléments publics induites pendant l'exécution d'une signature. En combinant l'attaque que nous avons proposé dans le cas du RSA avec les résultats obtenus par P. Q. Nguyen & I. E. Shparlinski pour analyser des morceaux d'aléas, nous avons proposé une nouvelle méthode permettant de retrouver une clé privée DSA à partir de signatures erronées.

Par exemple, lorsque une implantation *"Right-To-Left"* de l'exponentiation modulaire est utilisée, 16 perturbations du module public p pour autant d'exécutions de signatures DSA peuvent suffire pour retrouver la clé privée quelques minutes sur un PC. Ces résultats améliorent considérablement l'état de l'art et suggèrent que les perturbations d'éléments publics représentent une réelle menace pour les implantations de DSA. C'est pourquoi, nous conseillons aux développeurs de futurs solutions sécurisées de protéger la manipulation des données publiques pendant les opérations mêlant des données secrètes.

Chapitre 7

Conclusion

Bien que l'on trouve des prémices du principe dans certaines attaques par perturbation contre des implantations de courbes elliptiques [BMM00, CJ05], l'étude des perturbations d'éléments publics a réellement été introduite par J.-P. Seifert avec une attaque contre un mécanisme de vérification de signatures RSA [Sei05]. Cependant, ces perturbations sont considérées comme une véritable menace depuis qu'E. Brier *et al.* ont montré que leur exploitation peut mener jusque l'extraction complète de l'exposant privé d [BCMCC06]. En répertoriant ces travaux, nous avons remarqué que ceux-ci exploitaient exclusivement des perturbations du module N intervenant avant le calcul de signature. Ainsi, nous nous sommes demandés si des perturbations du module RSA intervenant pendant le calcul de signature pouvaient aussi être exploitées.

Pour commencer, nous avons étudié les effets de ces perturbations lorsque des implantations classiques de la signature RSA. Dans le chapitre 4.2, nous avons montré que, lorsque l'exponentiation modulaire *"Right-To-Left"* est utilisée, les perturbations du module intervenant pendant l'exécution peut engendrer une fuite d'information critique. Ainsi, pour un RSA de taille 1024 bits, la clé privée peut être déterminée graduellement à partir de 250 signatures erronées et au prix de quelques minutes d'analyse sur un PC standard.

Nous avons ensuite essayé d'étendre l'attaque précédente aux implantations *"Left-To-Right"* de la signature RSA. Bien que les deux implantations duales présentent de fortes similitudes, l'analyse de perturbations a dû être adaptée pour que les signatures erronées puissent être exploitées. En effet, seules les signatures dont le module erroné est un nombre premier laissent fuir une quantité d'information exploitable sur l'exposant privé. Aussi, il a fallu adapter le modèle de perturbation en conséquence. Dans le chapitre 4.3, nous nous sommes appuyés sur la théorie des nombres pour montrer que de telles signatures peuvent être obtenus dans un cas sur 305, ce qui est plus fréquent qu'on ne pourrait le penser. En combinant ce résultat avec l'utilisation de la méthode de Tonelli & Shanks [BM06] pour le calcul de racines carrées, nous avons montrés que l'exposant privé pouvait encore être retrouvé avec un surcoût raisonnable.

Pour contrer ces attaques, nous avons conjecturé que l'utilisation de la méthode de masquage de l'exposant proposée par P. Kocher [Koc96] serait suffisante. Cependant, l'étude approfondie du masque lui-même nous a permis de proposer la première attaque par perturbation des données publiques visant à extraire un exposant privé RSA masqué (*cf.* Chapitre 5). Bien que les poids faibles de l'exposant paraissent mieux masqués que les bits de poids fort, nous avons contourné cette difficulté en utilisant plus de signatures erronées. Par rapport à l'attaque d'implantations classiques, le nombre additionnel de signatures à perturber reste raisonnable et les performances de l'analyse pourront être grandement améliorés en parallélisant l'analyse et en utilisant les attaques de Coppersmith [Cop97, May03, Bau08].

Pour clore l'étude des perturbations d'éléments publics, nous avons regardé du côté des algorithmes de signatures de type El Gamal. De manière surprenante, cette problématique n'a été adressée que très récemment par C.H. Kim *et al.* [KBPJJ08]. Cependant l'extraction de la clé privée requiert un nombre conséquent de signatures erronées ce qui a amené les auteurs à conjecturer que les signatures type El Gamal seraient plus résistantes face à ce type de perturbation. En s'inspirant de l'attaque réseau proposée plus tôt part P.Q. Nguyen et I.E. Shparlinski [NS02], nous avons montré que les perturbations du module p intervenant pendant le calcul de signatures DSA pouvaient être efficacement exploitées. Selon les résultats présentés dans le chapitre 6, l'analyse de 16 signatures erronées ne prend que quelques minutes sur un PC et permet d'obtenir la clé privée DSA de 160 bits avec une forte probabilité. Ces résultats montrent donc que les implantations d'algorithmes de signatures de type El Gamal ne sont pas plus résistantes face à la perturbation des éléments publics.

Les différentes attaques que nous avons imaginées complètent ainsi l'état de l'art des attaques par perturbation des éléments publics. Nous avons montré que de telles perturbations sont source de fuite d'information si elles interviennent avant et pendant l'exécution de signatures. Ainsi, nous recommandons d'être particulièrement vigilant à l'intégrité des données publiques lorsqu'elles sont manipulées dans les routines utilisant la clé privée. Cette recommandation s'applique pour toutes les implantations de crypto-systèmes asymétriques.

Troisième partie

Perturbations d'algorithmes de chiffrement à flot

Chapitre 8

Étude des algorithmes de chiffrement à flot

8.1 État de l'art

8.1.1 Algorithmes à structure interne linéaire

Alors que les premières exploitations des perturbations de composants cryptographiques sont apparues à la fin des années 90 [BDL97], il a fallu attendre 2004 pour voir les premières applications contre les implantations d'algorithmes de chiffrement à flot [HS04]. Dans cet article, J. Hoch et A. Shamir ont détaillé une méthode permettant d'exploiter les perturbations induites dans un registre à décalage avec rétroaction linéaire (LFSR). Cette méthode est assez générique puisqu'à l'époque, la plupart des algorithmes de chiffrement à flot reposaient sur ce type de structure. Le papier présente les applications de cette méthode générique pour exploiter les perturbations d'implantations de LILI-128, SOBER-t32 et RC4.

La sécurité des implantations de RC4 vis-à-vis des perturbations a encore été adressée, un peu plus tard, par E. Biham *et al.* [BGN05]. Dans ce papier, ils montrent notamment que les perturbations du registre d'état interne peuvent être utilisées pour mettre RC4 dans un état *"impossible"*. L'exploitation de ces états impossibles conduit à une attaque plus efficace que les précédentes, tant sur le plan du nombre de perturbations à reproduire que sur le coût de l'analyse. Ces deux attaques de RC4 sont largement détaillées dans la section 2.4.1.2.

Les registres à décalage avec rétroaction linéaire ne sont pas seulement utilisés dans les algorithmes de chiffrement à flot mais aussi dans les générateurs pseudo-aléatoires. C'est très certainement qui a amené M. Gomulkiewicz *et al.* à étudier le comportement des implantations de l'algorithme A5/1 face aux fautes [GKW05]. En effet, ce générateur pseudo-aléatoire, connu pour être utilisé dans les réseaux GSM, est composé de trois LFSR ce qui lui confère une implantation matérielle efficace et peu coûteuse. Cependant, cette structure simpliste s'avère être aussi vulnérable aux perturbations. Pour montrer cette vulnérabilité, les auteurs ont considéré un attaquant capable d'empêcher l'un des LFSR de se mettre à jour à un instant donné de l'exécution. En étudiant la suite chiffrante dérivant d'une exécution perturbée suivant le modèle, il est possible d'améliorer d'un facteur 100 les précédentes attaques mathématiques connues contre A5/1.

8.1.2 Nouvelles tendances, nouvelles attaques

Alors que les algorithmes de chiffrement à flot étaient promis à un avenir flou, le projet eSTREAM a été monté pour sélectionner les algorithmes les plus prometteurs et répondre ainsi à la demande de certains marchés de niche (*e.g.* étiquettes RFID). Ces dernières années ont donc vu l'avènement de nouveaux algorithmes de chiffrement à flot. Or, ces algorithmes sont amenés à être implantés sur des composants avec des niveaux de sécurité physiques hétérogènes. C'est pourquoi la communauté s'est intéressée de nouveau à la résistance de ces algorithmes, du point de vue des perturbations.

La première attaque par perturbation contre un finaliste du projet eSTREAM, Trivium [CP05], a été présenté à FSE 2008 par M. Hojsik & B. Rudolf [HR08]. Cet algorithme de chiffrement à flot est basé sur un état interne de 288 bits répartis dans trois registres à décalage avec rétroaction non-linéaire. Le principe de l'attaque proposée consiste à basculer aléatoirement la valeur d'un bit de l'état interne. En analysant les suites chiffrantes dérivant de plusieurs perturbations, il est possible de construire un système d'équations en les bits d'état interne. Ce système sera simplifié, puis résolu en utilisant la simplicité de la fonction de rétroaction non-linéaire. Selon [HR08], 43 perturbations (12 dans la version optimisée) peuvent suffire pour retrouver l'état interne, et même remonter jusque l'extraction de la clé secrète. On peut noter que cette attaque par perturbation est aussi la première contre un registre de chiffrement à flot basé sur des registres à décalage avec rétroaction non-linéaire.

Les implantations d'un autre finaliste de eSTREAM ont aussi été étudiés vis-a-vis des perturbations, il s'agit de Rabbit [KY09]. Cet article montre comment exploiter les basculements unitaires des bits de l'état interne. Sous ce modèle classiquement considéré, A. Kirkanski & A. M. Youssef ont montré qu'il est possible de retrouver l'intégralité de l'état interne de Rabbit à partir de 128 – 256 différentes perturbations et au prix de $\mathcal{O}\left(2^{38}\right)$ calculs et d'une table précalculé de $\mathcal{O}\left(2^{41.6}\right)$ octets. Néanmoins, l'analyse échoue si, à un instant donné, la perturbation infecte plus d'un bit d'état interne. D'autre part, l'analyse proposée se limite à l'extraction de l'état interne seulement.

Dans le cadre de cette thèse, nous nous sommes particulièrement intéressés à scruter le comportement de ces nouveaux algorithmes de chiffrement à flot vis-a-vis perturbations malicieuses. Cette étude a abouti à la publication de deux nouvelles attaques qui seront présentées dans les chapitres suivants.

8.2 Nouveaux enjeux

Les transmissions cryptées haut débit, utilisées principalement aujourd'hui dans les réseaux informatiques et la télévision numérique, sont amenées à se généraliser autour des objets portables à grande capacité de stockage (clés USB, lecteurs MP3, cartes à puce, ...). Afin de renforcer la sécurité de ces liens, pour des besoins privés ou commerciaux, il est nécessaire de disposer de modules cryptographiques haut débit, basse consommation et résistant aux attaques.

Les membres du projet ANR Odyssée [30] ont choisi de s'intéresser aux algorithmes de chiffrement à flot, dont les caractéristiques semblent répondre au besoin identifié. Mais, contrairement aux algorithmes de chiffrement par blocs, comme le DES ou l'AES, les attaques physiques sur des implantations de chiffrement à flot ont été peu étudiées et les contre-mesures logicielles ou matérielles quasi inexistantes.

Le but du projet vise à étudier les algorithmes déjà proposés, particulièrement ceux émergeant

30. ODysSÉe : HAUT Débit SÉcurisé

du projet eSTREAM, et sélectionner ceux ayant des propriétés particulièrement favorables. Un des objectifs du projet était de définir précisément ces propriétés de manière à identifier les implantations d'algorithmes de chiffrement à flot offrant le meilleur compromis entre les performances et les possibilités de protection.

Parmi les critères de choix, la résistance aux attaques physiques des implantations d'algorithmes a particulièrement été étudiée. L'objectif de cette étude est double. Elle vise d'abord à adapter des attaques connues ou développer de nouvelles attaques basées sur des hypothèses de perturbations classiques ou innovantes. L'autre aspect est d'utiliser la connaissance de ces attaques pour identifier les sources de vulnérabilité afin de proposer des contre-mesures efficaces.

Dans ce cadre, nous avons proposé deux attaques différentes contre deux des finalistes du projet eSTREAM. La première consiste à utiliser un modèle de perturbation très peu étudié, mais pouvant être réalisé, pour améliorer les performances des attaques connues contre les implantations de Rabbit (*cf.* Section 9). La seconde, réalisée en collaboration avec l'ensemble des partenaires du projet, consiste à utiliser un modèle de perturbation classique pour attaquer les implantations de Grain-128 (*cf.* Section 10). Chacune de ces deux contributions a abouti sur la proposition de contre-mesures spécifiques visant à améliorer la sécurité physique sans amputer les performances.

Chapitre 9

Application à l'algorithme Rabbit

Sommaire

9.1 Préliminaires

9.1.1 Motivations

Notre idée d'attaque par perturbation sur les implantations de Rabbit s'inspire directement d'une des conclusions tirées par les concepteurs de l'algorithme au moment de sa description [BVCZ05]. En effet, ils proposent, dans cet article, d'analyser une version affaiblie de l'algorithme Rabbit, où toutes les additions de 32 bits sont remplacées par de simples *"ou exclusif"* (ou `XOR`) bit-à-bit dans la fonction de mise à jour de l'état interne (*cf.* Section 1.2.2.2). Les auteurs montrent alors dans [BVCZ05] que malgré cette substitution, le niveau de résistance de Rabbit face aux attaques mathématiques classiques n'est pas diminué. A partir de cette remarque nous nous sommes demandés si la substitution temporaire d'une seule addition par un `XOR` pouvait être exploitée. Ainsi, l'objectif de notre attaque est double. La première est de modérer les conclusions tirées par les concepteurs de Rabbit concernant la substitution des additions de la fonction de mise à jour. Le second objectif est d'étudier la sécurité de l'implantation d'un algorithme de chiffrement à flot sous une hypothèse de perturbation assez original. En effet, les modèles de perturbations classiquement utilisés reposent sur le basculement d'un ou plusieurs octets de l'état interne. Par conséquent, nous espérons que notre étude pourra motiver l'imagination de nouvelles attaques contre les implantations d'algorithmes de chiffrement par flot sous

des modèles de perturbations alternatifs.

Dans les sections suivantes, nous montrerons comment nous avons utilisé les perturbations pour réaliser de telles substitutions. Puis, nous détaillerons la méthode que nous avons mis en œuvre pour exploiter ces perturbations et retrouver l'intégralité de l'état interne de RABBIT à un instant donné.

9.1.2 Quelques propriétés des retenues

Avant de commencer la description de l'attaque proprement dite, nous allons définir quelques notations et rappeler certaines propriétés des retenues. En effet, en substituant malicieusement les additions par des `XOR`s, l'analyse de retenues sera au cœur de notre attaque. Commençons par donner une définition de la retenue.

Définition 9.1.2.1. *Soit (x, y) une paire d'entiers de taille n bits, le vecteur de retenu de n bits résultant de l'addition $x + y \mod 2^n$, aussi noté Retenue (x, y), peut être exprimé :*

$$\textit{Retenue } (x, y) = (x + y) \oplus (x \oplus y) \tag{9.1}$$

où $\oplus$ représente le "ou exclusif" bit-à-bit communément appelé `XOR`.

Complétons cette définition de la retenue de l'addition de deux entiers par une définition récursive de l'expression de chacun des bits de retenue.

Définition 9.1.2.2. *Soit (x, y) une paire d'entiers de taille n bits, le i-ème bit de retenue résultant de l'addition $x + y \mod 2^n$, noté $\textit{Retenue}_{(i)}(x, y)$, peut être exprimé récursivement ainsi :*

- *Si $i = 0$, $\textit{Retenue}_{(0)}(x, y) = 0$*
- *Si $i = 1$, $\textit{Retenue}_{(1)}(x, y) = x_0 \wedge y_0$*
- *Si $1 < i \leq n$, $\textit{Retenue}_{(i)}(x, y) = x_{i-1} \wedge y_{i-1} \vee \big(\textit{Retenue}_{(i-1)}(x, y) \wedge (x_{i-1} \vee y_{i-1})\big)$*

où x_i, y_i représentent respectivement les i-ème bits de x et y.

Regardons à présent un cas nettement plus intéressant pour analyser les perturbations de la mise à jour de RABBIT. Ce cas concerne l'étude de la somme de 3 entiers.

Proposition 9.1.1. *Soit (x, y, z) un triplet d'entiers de n bits, alors :*

$$(x + y + z) \oplus ((x + y) \oplus z) = \textit{Retenue } (x + y, z) \tag{9.2}$$

Démonstration. Afin de démontrer cette proposition, utilisons la définition 9.1.2.1 en utilisant le changement de variable $X = (x + y) \mod 2^n$. Alors,

$$\begin{aligned} & (x + y + z) \oplus ((x + y) \oplus z) \\ = \; & (X + z) \oplus (X \oplus z) \\ = \; & \text{Retenue } (X, z) \end{aligned}$$

ou encore, en substituant X par sa véritable valeur, on obtient Retenue $(x + y, y)$ □

Proposition 9.1.2. *Soit (x, y, z) un triplet d'entiers de n bits, alors :*

$$\begin{aligned} & (x + y + z) \oplus ((x \oplus y) + z) \\ = \; & \textit{Retenue } (x + y, z) \oplus \textit{Retenue } (x \oplus y, z) \oplus \textit{Retenue } (x, y) \end{aligned} \tag{9.3}$$

Démonstration. Ce résultat découle aussi de la définition 9.1.2.1. Pour un triplet de valeurs quelconques $(x, y, z) \in (\mathbb{Z}/n\mathbb{Z})^3$, $x + y + z$ peut aussi s'écrire :

$$\begin{aligned} x + y + z &= (x + y) \oplus z \oplus \text{Retenue } (x + y, z) \\ &= x \oplus y \oplus \text{Retenue } (x, y) \oplus \\ &\quad z \oplus \text{Retenue } (x + y, z) \end{aligned}$$

De plus, $((x \oplus y) + z) = x \oplus y \oplus z \oplus \text{Retenue } (x \oplus y, z)$. Ainsi, on obtient finalement :

$$\begin{aligned} & (x + y + z) \oplus ((x \oplus y) + z) \\ = \; & x \oplus y \oplus z \oplus \text{Retenue } (x, y) \oplus \text{Retenue } (x + y, z) \oplus \\ & x \oplus y \oplus z \oplus \text{Retenue } (x \oplus y, z) \\ = \; & \text{Retenue } (x + y, z) \oplus \text{Retenue } (x \oplus y, z) \oplus \text{Retenue } (x, y) \end{aligned}$$

□

9.2 Description de l'attaque

9.2.1 Perturbation transitoire d'une opération

9.2.1.1 Modèle de perturbation

Dans ce chapitre, nous allons proposer une méthode permettant d'exploiter le remplacement temporaire d'une addition avec retenue par une addition sans retenue (ou `XOR`) dans la fonction de mise à jour de l'état interne. Afin de nous placer dans ce cas d'étude, nous avons choisi de perturber le composant embarquant une implantation logicielle de Rabbit. Une telle perturbation pourra s'apparenter à une modification partielle du flot d'exécution de l'algorithme (*cf.* Section 2.3).

Description du modèle. Dans un premier temps, décrivons le but principal de notre attaque. Celui-ci sera d'utiliser les perturbations pour retrouver l'ensemble des valeurs retournées par la fonction principale de mise à jour *"g"*, que nous noterons $(g_{j,i})_{0 \leqslant j \leqslant 7}$, et ce, pour deux itérations consécutives. De cette manière, il sera possible d'utiliser le théorème 9.2.1, décrit plus loin, pour retrouver l'état interne et prédire la suite chiffrante. Comme décrit dans le schéma de mise à jour (*cf.* Section 1.2.2.2), le rafraîchissement de chacune des variables d'état interne $x_{j,i+1}$ est le résultat l'addition de trois variables : $g_{j,i}$, $g_{(j+7)_{[8]},i}$ et $g_{(j+6)_{[8]},i}$. Ce calcul est réalisé en exécutant deux additions consécutives sur 32 bits prenant en entrée 2 opérandes. Dans le modèle de faute que nous proposons, nous considérons un attaquant capable de perturber l'une de ces 2 additions de façon à ce qu'elle soit remplacée, juste pour une seule exécution, par un `XOR` de 32 bits. Ainsi, toutes les additions suivantes devront s'exécuter normalement. Par ailleurs, l'attaquant sera capable de choisir l'itération i ainsi que l'indice j de l'octet d'état interne $x_{j,i+1}$ dont il souhaite perturber la mise à jour. Enfin, comme notre analyse requiert de collecter plusieurs suites chiffrantes perturbées de différentes manières, l'attaquant devra être encore capable d'exécuter Rabbit avec le même vecteur d'initialisation. Comme le choix de ce vecteur n'influence pas notre analyse, la seule connaissance de sa valeur est suffisante.

Discussion. Dans le paragraphe suivant, nous avons un modèle de perturbation assez original pour étudier la sécurité des implantations de Rabbit. Nous allons donc présenter différentes

techniques, plus ou moins difficiles à réaliser, pour obtenir de tels effets. En effet, l'obtention de telles erreurs semble nécessiter une étude préliminaire du composant attaqué. Premièrement, comme nous étudions une implantation logicielle de l'algorithme, nous pouvons considérer que les opérations sont exécutées de manière séquentielle. Dans ce cas, il est possible de repérer assez finement l'instant où la valeur d'état interne $x_{j,i+1}$ est calculée. En ce qui concerne la substitution transitoire d'une addition, les perturbations générées par des défaillances électroniques ou des stimulations lumineuses peuvent produire de multiples effets [BECN+04, Gir04, SA02, Sko06]. Dans notre cas, la transformation temporaire d'une seule addition par un `XOR` peut être obtenue par les deux manières suivantes :

- Forcer la valeur contenant le bit de retenue : si cette valeur est forcée à 0 pendant la durée d'une addition, alors l'opération réalisée est une addition binaire sans retenue, soit un `XOR` bit-à-bit. Dans le cas où la valeur contenue dans le registre de retenue est forcée à 1, l'addition est replacée par un `XOR` bit-à-bit suivi d'une complémentation à 2.
- Corruption du code de l'instruction : La mémoire non-volatile, où le code l'instruction est normalement stocké, peut être modifiée au moment de la lecture en mémoire. Par exemple, pour le jeu d'instructions du microcontrôleur Intel 8051, le code d'une addition `ADD` commence par `0x20` alors que c'est `0x60` pour l'instruction de *"ou exclusif"* `XRL`. Dans ce cas, il suffit seulement de basculer la valeur d'un bit pour changer l'addition en `XOR`. On peut noter par ailleurs que le code de l'opération peut aussi être modifié dans la mémoire cache d'un microprocesseur.

Comme nous allons le montrer dans le paragraphe suivant, il est possible, en analysant la suite chiffrante erronée, d'identifier l'erreur introduite. Cette méthode permettra donc de choisir, parmi les suites chiffrantes collectées, celles qui sont potentiellement exploitables. Bien que de telles perturbations soient relativement difficiles à réaliser en pratique, nous avons quand même trouvé différentes façons d'obtenir les effets escomptés. Aussi, puisqu'il ne faut pas sous-estimer les moyens que peut mettre en œuvre un attaquant pour arriver à ses fins, nous allons proposer une méthode permettant d'exploiter de telles perturbations.

9.2.1.2 Exemple d'exécution erronée

Cette section détaille un exemple d'exécution erronée obtenue en perturbant la mise à jour de la variable de l'état interne x_1 à l'instant $i+1$. Plus précisément, nous allons détailler le cas où la seconde addition de 32 bits, exécutée pour mettre à jour x_1, est substituée par un `XOR` bit-à-bit :

$$\begin{aligned} & x_{1,i+1} = (g_{1,i} + (g_{0,i} \lll 8)) + g_{7,i} \\ \Rightarrow \quad & \hat{x}_{1,i+1} = (g_{1,i} + (g_{0,i} \lll 8)) \oplus g_{7,i} \end{aligned} \tag{9.4}$$

La figure 9.1 illustre encore cette perturbation. Ainsi, pour chaque exécution erronée, nous ne perturberons seulement que le calcul d'un x_j à la fois. On peut alors généraliser cette faute à chacune des additions utilisées pour mettre à jour les variables de l'état interne $(x_{j,i+1})_{0 \leqslant j \leqslant 7}$. On peut alors dissocier deux cas :

- Si j est pair,

$$\hat{x}_{j,i+1} \quad = \quad \begin{cases} \Big(g_{j,i} + \Big(g_{(j+7)_{[8]},i} \lll 16\Big)\Big) \oplus \Big(g_{(j+6)_{[8]},i} \lll 16\Big) \\ \text{ou } \Big(g_{j,i} \oplus \Big(g_{(j+7)_{[8]},i} \lll 16\Big)\Big) + \Big(g_{(j+6)_{[8]},i} \lll 16\Big) \end{cases}$$

– Sinon, quand j est impair,

$$\hat{x}_{j,i+1} = \begin{cases} \left(g_{j,i} + \left(g_{(j+7)_{[8]},i} \lll 8\right)\right) \oplus g_{(j+6)_{[8]},i} \\ \text{ou } \left(g_{j,i} \oplus \left(g_{(j+7)_{[8]},i} \lll 8\right)\right) + g_{(j+6)_{[8]},i} \end{cases}$$

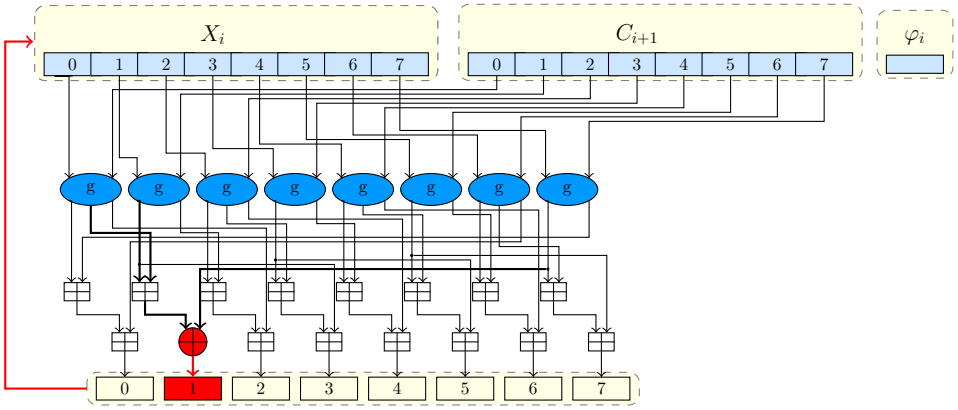


FIGURE 9.1 – Perturbation du calcul de $x_{1,i+1}$ pendant la mise à jour de RABBIT

Comme nous l'avons montré précédemment, la suite chiffrante de RABBIT est directement générée à partir des variables d'état interne $(x_{j,i+1})_{0 \leqslant j \leqslant 7}$. Aussi la perturbation commence par infecter la suite chiffrante, comme illustré par la figure 9.2. Suivant l'indice j de la variable d'état interne dont on a perturbé la mise à jour, il faut encore dissocier deux cas :

– Si j est pair, alors 32 bits consécutifs de suite chiffrante sont infectés. Cette partie pourra être repérée suivant j par $\hat{s}_i^{[(16\cdot(j+2)-1)..(16\cdot j)]}$,
– Sinon, quand j est impair, alors l'erreur infecte deux parties de 16 bits localisées respectivement par $\hat{s}_i^{[16\cdot((j-2)_{[8]}+1)-1..16\cdot(j-2)_{[8]}]}$ et $\hat{s}_i^{[16\cdot((j+3)_{[8]}+1)-1..16\cdot(j+3)_{[8]}]}$ (*cf.* Figure 9.2 pour $j = 1$).

Enfin, la perturbation se propage en infectant plusieurs variables d'état interne lors des itérations suivantes. Mais, le résultat de cette propagation aux itérations suivantes ne sera pas analysé ici. À présent, voyons comment nous avons analysé les perturbations détaillées précédemment pour retrouver l'intégralité de l'état interne de RABBIT et, dans certains cas, la clé secrète.

Remarque 6. *Il est intéressant de noter que, suivant l'indice j de la variable d'état interne dont on a perturbé la mise à jour, la faute se propagera de manière différente dans la suite chiffrante générée. Par conséquent, en identifiant les morceaux de suite chiffrante infectés, il est possible de déduire la variable d'état interne dont on a réussi à perturber la mise à jour. Cette identification pourra être simplement réalisée en comparant la suite chiffrante erronée avec la suite chiffrante correcte et en repérant la position des valeurs non nulles. Cette remarque sera notamment utilisée au moment de collecter des perturbations infectant le calcul de différentes variables d'état internes.*

9.2.2 Analyse différentielle des perturbations

Comme nous le décrivions au moment de présenter le modèle de perturbation que nous avons choisi, le but de notre analyse consiste à retrouver l'ensemble des valeurs retournées par la

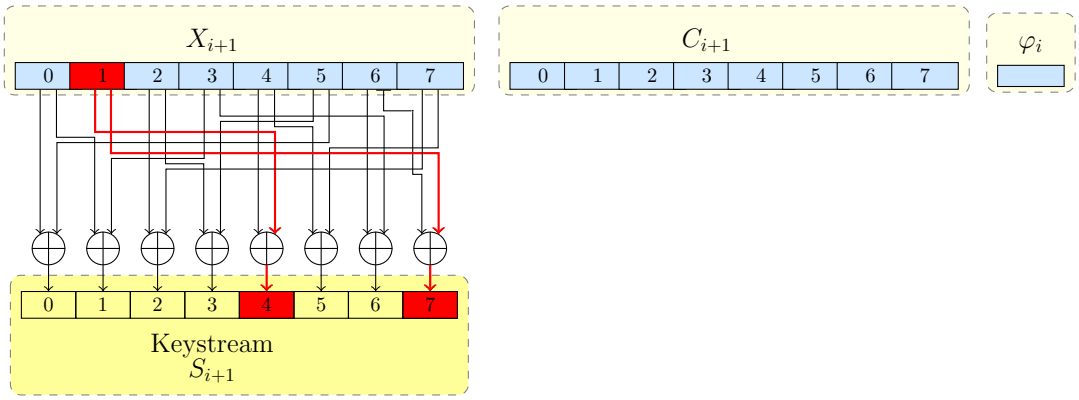


FIGURE 9.2 – Propagation de la perturbation dans la suite chiffrante de RABBIT

fonction principale de mise à jour *"g"*: $(g_{j,i})_{0\leqslant j\leqslant 7}$. En effet, si un attaquant est capable d'obtenir ces valeurs pour deux itérations consécutives, $i-1$ et i, il pourra utiliser le théorème suivant pour retrouver l'intégralité de l'état interne de RABBIT à l'instant $i+1$.

Théorème 9.2.1. *Si un attaquant connaît les valeurs résultant de la fonction "g", $(g_{j,i})_{0\leqslant j\leqslant 7}$, pour deux itérations consécutives i et $i+1$, alors il peut déterminer toutes les variables d'état interne à $(x_{j,i+1})_{0\leqslant j\leqslant 7}$ et réduire le nombre de valeurs possibles pour le reste de l'état interne à l'instant $i+1$ de 2^{256+1} à 82, en moyenne.*

Démonstration. Supposons que l'attaquant réussisse à obtenir toutes les valeurs de $(g_{j,i-1})_{0\leqslant j\leqslant 7}$ et $(g_{j,i})_{0\leqslant j\leqslant 7}$. À partir de ces valeurs, il peut calculer les variables d'état interne $(x_{j,i})_{0\leqslant j\leqslant 7}$ et $(x_{j,i+1})_{0\leqslant j\leqslant 7}$ en utilisant les opérations réalisées pour mettre à jour l'état interne (*cf.* Section 1.2.2.2). L'attaquant a alors retrouvé la moitié de l'état interne de RABBIT pour deux itérations consécutives.
Afin de déterminer complètement l'état interne à l'itération $i+1$, l'attaquant doit encore retrouver la valeur des huit compteurs $(c_{j,i+1})_{0\leqslant j\leqslant 7}$ et du bit de retenue $\phi_{7,i+1}$. Or, d'après le schéma de mise à jour de RABBIT, ces compteurs sont utilisés en entrée de la fonction *"g"* pour mettre à jour les variables d'état interne (*cf.* Section 1.2.2.2 Équation 1.4). Bien que cette fonction *"g"* ne soit pas bijective [Aum07], des résultats précédents [YHL08] ont montré qu'en moyenne 1.59 entrées correspondent à une même sortie. (*i.e.* nombre de couples $(x_{j,i}, c_{j,i+1})$ ayant la même image par la fonction *"g"*). Ce résultat a été confirmé par nos expérimentations. Par conséquent, si l'attaquant connaît à la fois les résultats de cette fonction $(g_{j,i})_{0\leqslant j\leqslant 7}$ et une partie de l'entrée, $(x_{j,i})_{0\leqslant j\leqslant 7}$, il peut alors déterminer en moyenne 1.59 valeurs possibles pour chacun des $c_{j,i+1}$. De cette manière, le nombre de valeurs candidates pour tous les compteurs $(c_{j,i+1})_{0\leqslant j\leqslant 7}$ est de $1.59^8 \approx 41$ au lieu de $2^{8\cdot 32} = 2^{256}$. Le dernier bit de retenu $\phi_{7,i+1}$ peut, quand à lui être retrouvé par une recherche exhaustive ou, de manière plus astucieuse, en comparant $c_{7,i+1}$ à a_7 [31]. Finalement, le nombre de valeurs candidates pour retrouver le reste de l'état interne de RABBIT à l'instant $t+1$ est de $2 \times 41 = 82$ ce qui confirme le résultat du théorème. □

Afin de retrouver ces $(g_{j,i})_{0\leqslant j\leqslant 7}$, nous allons mener une analyse différentielle des perturbations. C'est à dire que nous allons comparer (grâce à l'opérateur `XOR`) la suite chiffrante erronée

31. En effet, si $c_{7,i+1} < a_7$, cela signifie qu'une réduction modulaire à eu lieu au moment de l'addition, et alors $\phi_{7,i+1} = 1$

avec une suite chiffrante correcte correspondante pour mettre en évidence les effets de nos perturbations. Dans le paragraphe précédent, nous avons montré qu'une perturbation réalisée suivant notre modèle n'infecte que 32 bits de la suite chiffrante et que leur répartition nous donne des informations précises sur les opérations perturbées. Par conséquent, le `XOR` entre la suite chiffrante correcte et celle erronée sera nul excepté pour les 32 bits infectés :

– Si j est pair, alors

$$\begin{aligned} & s_i^{[(16\cdot(j+2)-1)..(16\cdot j)]} \oplus \hat{s_i}^{[(16\cdot(j+2)-1)..(16\cdot j)]} \\ = \quad & x_{j,i}^{[31..0]} \oplus \hat{x}_{j,i}^{[31..0]} \end{aligned} \tag{9.5}$$

et 0 partout ailleurs,

– Sinon, quand j est impair, alors

$$\begin{aligned} & s_i^{[16\cdot((j-2)_{[8]}+1)-1..16\cdot(j-2)_{[8]}]} \oplus \hat{s_i}^{[16\cdot((j-2)_{[8]}+1)-1..16\cdot(j-2)_{[8]}]} \\ = \quad & x_{j,i}^{[15..0]} \oplus \hat{x}_{j,i}^{[15..0]}, \end{aligned} \tag{9.6}$$

$$\begin{aligned} & s_i^{[16\cdot((j+3)_{[8]}+1)-1..16\cdot(j+3)_{[8]}]} \oplus \hat{s_i}^{[16\cdot((j+3)_{[8]}+1)-1..16\cdot(j+3)_{[8]}]} \\ = \quad & x_{j,i}^{[31..16]} \oplus \hat{x}_{j,i}^{[31..16]} \end{aligned} \tag{9.7}$$

et 0 partout ailleurs.

Par ailleurs, ces expressions peuvent encore être détaillées suivant l'addition de 32 bits qui a été substituée. Ainsi, l'expression de $x_{j,i}^{[31..0]} \oplus \hat{x}_{j,i}^{[31..0]}$ peut être détaillée en utilisant les propositions 9.1.1 et 9.1.2. Par exemple, si l'on considère seulement les perturbations de la deuxième addition, on obtiendrait :

– Si j est pair,

$$\begin{aligned} & x_{j,i}^{[31..0]} \oplus \hat{x}_{j,i}^{[31..0]} \\ = \quad & \Big(g_{j,i-1} + (g_{(j+7)_{[8]},i-1} \lll 8) + g_{(j+6)_{[8]},i-1}\Big) \oplus \\ & \Big(g_{j,i-1} + (g_{(j+7)_{[8]},i-1} \lll 8) \oplus g_{(j+6)_{[8]},i-1}\Big) \\ = \quad & \text{Retenue } \Big((g_{j,i-1} + g_{(j+7)_{[8]},i-1} \lll 8), g_{(j+6)_{[8]},i-1}\Big) \end{aligned} \tag{9.8}$$

– ou bien, si j est impair,

$$\begin{aligned} & x_{j,i}^{[31..0]} \oplus \hat{x}_{j,i}^{[31..0]} \\ = \quad & \Big(g_{j,i-1} + (g_{(j+7)_{[8]},i-1} \lll 16) + (g_{(j+6)_{[8]},i-1} \lll 16)\Big) \oplus \\ & \Big(g_{j,i-1} + (g_{(j+7)_{[8]},i-1} \lll 16) \oplus (g_{(j+6)_{[8]},i-1} \lll 16)\Big) \\ = \quad & \text{Retenue } \Big((g_{j,i-1} + g_{(j+7)_{[8]},i-1} \lll 16), g_{(j+6)_{[8]},i-1} \lll 16\Big). \end{aligned} \tag{9.9}$$

Des expressions similaires peuvent être obtenues si l'on considère, cette fois, que c'est la première des deux additions qui est perturbée lors de la mise à jour d'une variable d'état interne. L'attaquant doit alors répéter les perturbations afin de construire un système d'équations à partir des différentes expressions obtenues. Le système d'équations ainsi obtenu est détaillé ci-dessous.

Premier jeu d'équations. Le premier jeu d'équations est obtenu en perturbant les secondes additions exécutées pour mettre à jour chacune des variables d'état interne, à la même itération $i+1$.

$$\begin{aligned}
x_{0,i} \oplus \hat{x}_{0,i} &= \text{Retenue}\ ((g_{0,i-1} + (g_{7,i-1} \lll 16)), (g_{6,i-1} \lll 16)) \\
x_{1,i} \oplus \hat{x}_{1,i} &= \text{Retenue}\ ((g_{1,i-1} + (g_{0,i-1} \lll 8)), g_{7,i-1}) \\
x_{2,i} \oplus \hat{x}_{2,i} &= \text{Retenue}\ ((g_{2,i-1} + (g_{1,i-1} \lll 16)), (g_{0,i-1} \lll 16)) \\
x_{3,i} \oplus \hat{x}_{3,i} &= \text{Retenue}\ ((g_{3,i-1} + (g_{2,i-1} \lll 8)), g_{1,i-1}) \\
x_{4,i} \oplus \hat{x}_{4,i} &= \text{Retenue}\ ((g_{4,i-1} + (g_{3,i-1} \lll 16)), (g_{2,i-1} \lll 16)) \\
x_{5,i} \oplus \hat{x}_{5,i} &= \text{Retenue}\ ((g_{5,i-1} + (g_{4,i-1} \lll 8)), g_{3,i-1}) \\
x_{6,i} \oplus \hat{x}_{6,i} &= \text{Retenue}\ ((g_{6,i-1} + (g_{5,i-1} \lll 16)), (g_{4,i-1} \lll 16)) \\
x_{7,i} \oplus \hat{x}_{7,i} &= \text{Retenue}\ ((g_{7,i-1} + (g_{6,i-1} \lll 8)), g_{5,i-1})
\end{aligned}$$

Second jeu d'équations. Le second jeu d'équations, quand à lui, est obtenu de la même manière, mais en modifiant les premières additions. Ces deux jeux d'équations sont obtenues à partir de 16 suites chiffrantes perturbées à la même itération $i+1$, mais de façon différentes. Par ailleurs, ces deux jeux d'équations forment le système qui sera résulu pour obtenir tous les $(g_{j,i})_{0 \leqslant j \leqslant 7}$. Ainsi, pour toutes les valeurs de j telles que $0 \leqslant j \leqslant 7$, nous avons :

– Si j est pair, $x_{j,i} \oplus \hat{x}_{j,i}$ est égal à :

$$\begin{aligned}
&\text{Retenue}\ \Big((g_{j,i-1} + (g_{(j+7)_{[8]},i-1} \lll 16)), (g_{(j+6)_{[8]},i-1} \lll 16)\Big) \\
\oplus\ &\text{Retenue}\ \Big((g_{j,i-1} \oplus (g_{(j+7)_{[8]},i-1} \lll 16)), (g_{(j+6)_{[8]},i-1} \lll 16)\Big) \\
\oplus\ &\text{Retenue}\ \Big(g_{j,i-1}, (g_{(j+7)_{[8]},i-1} \lll 16)\Big)
\end{aligned}$$

– Sinon, quand j est impair, $x_{j,i} \oplus \hat{x}_{j,i}$ est égal à :

$$\begin{aligned}
&\text{Retenue}\ \Big(\Big(g_{j,i-1} + \Big(g_{(j+7)_{[8]},i-1} \lll 8\Big)\Big), g_{(j+6)_{[8]}}\Big) \\
\oplus\ &\text{Retenue}\ \Big(\Big(g_{j,i-1} \oplus \Big(g_{(j+7)_{[8]}} \lll 8\Big)\Big), g_{(j+6)_{[8]}}\Big) \\
\oplus\ &\text{Retenue}\ \Big(g_{j,i-1}, \Big(g_{(j+7)_{[8]}} \lll 8\Big)\Big)
\end{aligned}$$

Ainsi, l'analyse différentielle permet de construire un système d'équations mettant en évidence les vecteurs de retenue issus des additions effectuées pour mettre à jour l'état interne. Dans les paragraphes suivants, nous allons détailler la méthode que nous avons développée pour résoudre le système d'équations.

9.2.2.1 Analyse préliminaire des vecteurs de retenue

Dans un premier temps, nous allons nous intéresser au premier jeu d'équations, à savoir celui obtenu en perturbant la seconde des deux additions. En effet, on remarque que, dans ce cas, le résultat de l'analyse différentielle est égale à la retenue issue de la somme d'un $g_{j,i-1}$ et de deux autres sorties de la fonction *"g"*. Or, d'après la définition 9.1.2.2, les bits de retenue tels que $1 < i \leqslant n$ peuvent être exprimés récursivement :

$$\text{Retenue}_{(i)}(x, y) = x_{i-1} \wedge y_{i-1} \vee \big(\text{Retenue}_{(i-1)}(x, y) \wedge (x_{i-1} \vee y_{i-1})\big) \tag{9.10}$$

Alors, une simple analyse du vecteur de retenue peut permettre d'obtenir directement de l'information sur chacune des opérandes. Pour être plus précis, si l'on connaît le vecteur de retenue Retenue (x, y) résultant de l'addition des opérandes x et y, on peut alors déduire :

- Si $\text{Retenue}_{(i-1)}(x, y) = 0$: Dans ce cas, $\text{Retenue}_{(i)}(x, y) = x_{i-1} \wedge y_{i-1}$. Par conséquent, si le i-ème bit de retenue vaut 1, on peut déduire que $x_{i-1} = y_{i-1} = 1$. Sinon, on sait que $x_{i-1} = \bar{y}_{i-1}$ mais nous n'exploiterons pas cette information pour notre analyse.
- Ou bien, si $\text{Retenue}_{(i-1)}(x, y) = 1$: Ici $\text{Retenue}_{(i)}(x, y) = x_{i-1} \wedge y_{i-1} \vee x_{i-1} \vee y_{i-1} = x_{i-1} \vee y_{i-1}$. Ici, quand $\text{Retenue}_{(i)}(x, y) = 0$ on peut déduire que $x_{i-1} = y_{i-1} = 0$. En revanche, nous n'exploiterons pas le cas contraire même si, dans ce cas, $x_{i-1} = \bar{y}_{i-1}$.

Sachant que chacun des $g_{j,i-1}$ apparaît exactement une fois comme seule opérande dans un des 8 vecteurs de retenue issus de l'analyse différentielle, l'application de l'observation précédente peut permettre de retrouver en moyenne 25% des bits des $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$. Nous avons implanté cette analyse rapide en pratique et nous avons mesuré qu'en moyenne environ 19% des bits de $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$ pouvaient être retrouvés de cette manière. Ce résultat est donc légèrement inférieur à ce que l'on pouvait espérer obtenir car nous avons supposé que le vecteur de retenue était parfaitement aléatoire ce qui est incorrect. Cependant ces performances pourraient être améliorées en utilisant les informations que nous avons volontairement laissées de côté.

9.2.2.2 Extraction de l'état interne complet de Rabbit

Grâce à l'analyse préliminaire d'une partie des équations issues de différentes exécutions erronées, nous avons réussi à déterminer près de 20% des bits de $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$. Les bits restants seront déterminés grâce à l'analyse du système d'équations complet (*cf.* Section 9.2.2). Afin de construire ce système d'équations en $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$, l'attaquant doit préalablement perturber, une par une, chacune des additions exécutées lors de la mise à jour des variables d'état interne. En d'autre termes, l'attaquant doit perturber, à la même itération i, $8 + 8 = 16$ exécutions de Rabbit et collecter les suites chiffrantes erronées correspondantes. En comparant ces suites chiffrantes erronées à la suite chiffrante correcte, il peut alors extraire les équations en $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$ qui formeront le système à résoudre (*cf.* Section 9.2.2).

Considérons à présent les inconnues $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$, qui sont des valeurs de 32 bits, comme une famille de $8 \times 32 = 512$ valeurs binaires. Le système d'équations obtenu grâce aux perturbations peut alors être considéré comme un système formé de $16 \times 31 = 496$ équations pour 512 inconnues [32]. L'avantage de cette représentation est qu'elle permet de prendre en compte facilement les bits déjà obtenus grâce à l'analyse préliminaire. En revanche, comme ces équations expriment les retenues d'addition, le degré des polynômes multivariés mis en jeu augmente avec la profondeur des bits de retenue que l'on souhaite analyser (*i.e.* le polynôme multivarié représentant $\text{Retenue}_{(i-1)}(x, y)$ est de degré i d'après la définition 9.1.2.2). Par conséquent, l'utilisation de méthodes de re-linéarisation usuelles, type algorithme XL [CKPS00], n'est pas pertinent. En fait, le moyen le plus simple que nous avons trouvé pour résoudre ce système est de réaliser une recherche exhaustive sur des morceaux de 8-bits de chacune des équations du système. Cela revient à considérer chaque équation de 32 bits, comme $x_{j,i} \oplus \hat{x}_{j,i}$ un jeu de 4 sous-équations de

32. Le bit de poids faible d'un vecteur de retenu est toujours nul, ainsi seuls les 31 bits restants sont exploitables

8 bits :

$$x_{j,i} \oplus \hat{x}_{j,i} \quad \Rightarrow \begin{cases} x_{j,i} \oplus \hat{x}_{j,i}^{[7..0]} \\ x_{j,i} \oplus \hat{x}_{j,i}^{[15..8]} \\ x_{j,i} \oplus \hat{x}_{j,i}^{[23..16]} \\ x_{j,i} \oplus \hat{x}_{j,i}^{[31..24]} \end{cases} \tag{9.11}$$

Suivant l'addition qui a été perturbée, chaque sous-équation peut prendre deux formes différentes faisant intervenir de 8 bits de $g_{j,i-1}$, $g_{(j+7)_{[8]},i-1}$ et $g_{(j+6)_{[8]},i-1}$. Par exemple, pour $x_{0,i} \oplus \hat{x}_{0,i}^{[7..0]}$, l'attaquant cherchera simultanément les valeurs de $g_{0,i-1}^{[7..0]}$, $g_{7,i-1}^{[23..16]}$ et $g_{6,i-1}^{[23..16]}$ qui devront satisfaire :

$$\begin{cases} \Delta_1 = \text{Retenue } \left(\left(g_{0,i-1}^{[7..0]} + g_{7,i-1}^{[23..16]} \right), g_{6,i-1}^{[23..16]} \right) \\ \Delta_2 = \quad \text{Retenue } \left(\left(g_{0,i-1}^{[7..0]} + g_{7,i-1}^{[23..16]} \right), g_{6,i-1}^{[23..16]} \right) \\ \qquad \oplus \text{Retenue } \left(\left(g_{0,i-1}^{[7..0]} \oplus g_{7,i-1}^{[23..16]} \right), g_{6,i}^{[23..16]} \right) \\ \qquad \oplus \text{Retenue } \left(g_{0,i-1}^{[7..0]}, g_{7,i-1}^{[23..16]} \right) \end{cases}$$

où Δ_1 et Δ_2 représentent $x_{0,i-1} \oplus \hat{x}_{0,i-1}^{[7..0]}$ respectivement quand la seconde et la première des deux additions de la fonction de mise à jour est perturbée. De cette manière, il faut donc résoudre 4 paires d'équations pour chacun des $0 \leqslant j \leqslant 7$. Par conséquent, le système d'équations que nous avons résolu peut être considéré comme un système de $2 \times 4 \times 8 = 64$ équations pour $8 \times 4 = 32$ inconnues de 8 bits, puisque l'on cherche chacun des $g_{j,i}$ par tranches de 8 bits. Résoudre ainsi chacune des sous-équations de 8 bits requiert de chercher simultanément au plus 8 bits de 3 différents $g_{j,i-1}$ (sachant que l'analyse préliminaire permet déjà de déterminer près de 20% des bits des $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$). De plus, pour pouvoir résoudre les 3 sous-équations allant des bits 8 à 31, l'attaquant devra faire des hypothèses sur les bits de retenue :

- $\text{Retenue}_{(7)}({g_{j,i-1}}^{[7..0]}, {g_{(j+7)_{[8]},i-1}}^{[23..16]})$,
- $\text{Retenue}_{(15)}({g_{j,i-1}}^{[7..0]}, {g_{(j+7)_{[8]},i-1}}^{[23..16]})$,
- $\text{Retenue}_{(23)}({g_{j,i-1}}^{[7..0]}, {g_{(j+7)_{[8]},i-1}}^{[23..16]})$.

Ces valeurs étant, a priori, inconnues au début de la résolution. D'un point de vue de la complexité calculatoire, pour chaque $0 \leqslant j \leqslant 7$, celle-ci est au pire de $\mathcal{O}\left(4 \times 2^{3\times 8+3}\right) = \mathcal{O}\left(2^{29}\right)$ possibilités à tester. En pratique, puisque l'analyse préliminaire permet de retrouver près de 20% des bits des $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$ ce qui ramène le nombre moyen de bits à déterminer pour chacun des $g_{j,i-1}$ à $25,6$ bits ou encore $6,4$ bits par équations où ils apparaissent comme seul opérande. Le nombre de possibilités à tester peut alors être réduit à $\mathcal{O}\left(4 \times 2^{3\times 6,4+3}\right) \approx \mathcal{O}\left(2^{24}\right)$.

Maintenant que nous avons évalué le nombre de possibilités à tester en entrée du système d'équations, nous avons voulu évaluer le nombre de solutions en sortie. Pour ce faire, nous avons généré 20000 suites chiffrantes erronées possibles et compté le nombre de $(g_{j,i-1})_{0 \leqslant j \leqslant 7}$ solutions possibles des systèmes obtenus. Les résultats expérimentaux ont montré qu'en moyenne, le système d'équations construit à partir des suites chiffrantes erronées retourne $2^{13.72}$ solutions possibles pour les $(g_{j,i})_{0 \leqslant j \leqslant 7}$. Or, pour pouvoir appliquer le théorème 9.2.1 et retrouver l'état interne de RABBIT à un instant donné, il faut pouvoir retrouver les $(g_{j,i})_{0 \leqslant j \leqslant 7}$ pour deux itérations consécutives. Cela implique que le nombre de solutions retournées par la résolution de deux systèmes est, en moyenne de $2^{2\times 13.72} = 2^{27.44}$. En combinant ce résultat avec le résultat

du théorème 9.2.1, le nombre total de combinaisons à tester pour retrouver l'intégralité de l'état interne de RABBIT à l'instant $i+1$ est de $2^{27.44} \times 80 \approx 2^{34}$.
Pour finir, l'état interne de RABBIT sera déterminé en construisant, pour chacune des 2^{34} valeurs candidates, les variables d'état interne $(x'_{j,i+1})_{0\leqslant j\leqslant 7}$, les compteurs d'état interne $(c'_{j,i+1})_{0\leqslant j\leqslant 7}$ et le bit de retenue $\phi'_{7,i+1}$. Puis l'attaquant générera la suite chiffrante chiffrante associée s' qu'il comparera à la suite chiffrante correcte sur les itérations $i+1$ et $i+2$. Si les deux suites chiffrantes correspondent, alors les valeurs candidates choisies pour l'état interne sont les bonnes. L'attaquant sera donc alors capable de prédire les suites chiffrantes suivantes à partir de l'état interne qu'il a retrouvé.

Vers une obtention de la clé secrète. Dans l'analyse que nous avons proposé tout au long de ce chapitre, nous avons montré qu'il était possible d'exploiter des perturbations pour retrouver complètement l'état interne de RABBIT à un instant i. Comme nous l'avons souligné, cela permet à l'attaquant de prédire la valeur des bits de suite chiffrante mais, si i est suffisamment petit, alors l'attaquant peut même aller jusqu'à l'extraction de la clé secrète. En effet, selon [YHL08], si $i = 2$, alors l'attaquant peut obtenir la clé secrète à partir de l'état interne et ce, au prix de $\mathcal{O}\left(2^{32}\right)$ possibilités à tester. En quelques mots, pour extraire la clé secrète, l'attaquant commence par deviner les valeurs des $(\phi_{j,i})_{0\leqslant j\leqslant 7}$ pour *"inverser"* la fonction de mise à jour de l'état interne de RABBIT (cf. Section 1.2.2.2), puis l'initialisation par la clé secrète (cf. Section 1.2.2.1).
Par conséquent, lorsque $i = 2$ la complexité de cette étape est dominée par la complexité de l'analyse de perturbations. Donc le coût total de l'attaque menant à l'extraction de la clé secrète reste dominé par $\mathcal{O}\left(2^{34}\right)$ possibilités à tester. Nous laisserons le soin au lecteur de se référer à [YHL08] pour plus de détails concernant cette analyse complémentaire.

9.2.3 Algorithme d'attaque

Algorithme. Notre attaque par perturbation contre les implantations de RABBIT peut être divisée en 5 étapes distinctes, allant de l'injection de fautes pendant différentes exécutions de RABBIT jusqu'à l'extraction de la clé secrète. Ces différentes étapes sont résumées et listées dans ce paragraphe.

Étape 1 : Perturber, une par une, chacune des seize additions de la fonction de mise à jour de RABBIT pour les deux itérations consécutives i et $i+1$. Il faut donc collecter les suites chiffrantes erronées correspondant à $2 \times 16 = 32$ exécutions perturbées de RABBIT (avec le même vecteur d'initialisation),

Étape 2 : Comparer les morceaux de suites chiffrantes erronées $\hat{s}_i^{[127..0]}$ et $\hat{s}_{i+1}^{[127..0]}$ avec les morceaux de suite chiffrante correcte correspondants ${s_i}^{[127..0]}$ et ${s_{i+1}}^{[127..0]}$. Ensuite, vérifier que les fautes ont été correctement injectées (*cf.* Section 9.2.2),

Étape 3 : À partir des comparaisons entre les différentes sorties correcte/erronées réalisées dans l'étape précédente, construire deux systèmes de 16 équations chacun (*cf.* Section 9.2.2) pour les perturbations réalisées aux itérations i et $i+1$ respectivement. Puis, en menant une analyse préliminaire sur les vecteurs de retenue suivie de la résolution de sous-systèmes d'inconnues de 8 bits, retrouver des valeurs probables pour les $(g_{j,i-1})_{0\leqslant j\leqslant 7}$ et $(g_{j,i})_{0\leqslant j\leqslant 7}$. Enfin, on finit par calculer les valeurs probables pour les variables d'état interne $(x_{j,i})_{0\leqslant j\leqslant 7}$ et $(x_{j,i+1})_{0\leqslant j\leqslant 7}$,

Étape 4 : Calculer les valeurs probables des compteurs d'état interne $(c_{j,i+1})_{0\leqslant j\leqslant 7}$ et de la retenue $\phi_{7,i+1}$ correspondants aux valeurs possibles des variables d'état interne retrouvées à l'étape précédente (*cf.* Théorème 9.2.1),

Étape 5 : Enfin, pour chacune des valeurs possibles de l'état interne complet à l'instant $i+1$, comparer les bits de suite chiffrante correspondants à ceux de la suite chiffrante correcte (pour 2 itérations). En cas d'égalité, la valeur possible d'état interne est alors identifiée comme la bonne et il est alors possible de prédire les futurs bits de suite chiffrante. Si $i = 2$, l'attaquant peut encore choisir de réaliser une analyse complémentaire pour extraire la clé de chiffrement.

Complexité. L'efficacité d'une attaque par perturbation ne se mesure pas seulement en fonction du modèle de perturbation mais aussi par le nombre de perturbations à provoquer pour pouvoir réussir à extraire les informations secrètes. En théorie, pour obtenir un nombre d'équations suffisant pour réussir notre analyse l'attaquant devra perturber 32 exécutions différentes de RABBIT. Ces 32 perturbations correspondent à la modification de chacune des 16 additions de la fonction de mise à jour de l'état interne sur deux itérations consécutives. En pratique, ce nombre de perturbations est plus important et, dépend fortement des capacités de l'attaquant à reproduire finement les effets escomptés sur le composant ciblé (*cf.* Section 9.2.1.1).

En termes de temps de calcul, la complexité globale de notre attaque est dominée par la complexité de l'étape 5, à savoir tester toutes les solutions probables pour l'état interne (*i.e.* obtenus en résolvant les systèmes d'équations formés par les morceaux de suites chiffrantes erronées) jusqu'à identifier celle générant la suite chiffrante correcte. La complexité de notre attaque est donc dominée par la comparaison de $\mathcal{O}\left(2^{34}\right)$ possibilités pour l'état interne de RABBIT. Par conséquent, les performances de notre attaque par perturbation sont meilleures que pour celle proposée par A. Kirkanski et A. M. Youssef [KY09], que ce soit en termes de nombre de perturbations ou encore de complexité. En effet, l'attaque précédente nécessitait d'analyser le résultat de 128 à 256 perturbations de l'état interne au prix de 2^{38} comparaisons. Néanmoins, le modèle de perturbation adopté dans [KY09] pourra être considéré comme plus facile à reproduire que celui que nous avons choisi.

9.3 Conclusion

Depuis sa présentation à FDTC 2003 [BVP+03], l'algorithme de chiffrement à flot RABBIT est considéré par la communauté comme un algorithme plutôt sûr d'un point de vue mathématique. Mais curieusement, la question de la sécurité des implantations de RABBIT n'a été adressée que très récemment [KY09].

Dans ce chapitre, nous avons présenté une nouvelle attaque par perturbation visant les implantations logicielles de RABBIT. Notre analyse théorique combinée aux résultats obtenus avec notre implantation de l'attaque permet d'améliorer l'état de l'art des attaques par perturbation sur RABBIT, que ce soit d'un point de vue du nombre de fautes à réaliser de la comlexité calculatoire. Bien que le modèle que nous ayons considéré soit assez original et relativement peu étudié dans la littérature, nous avons montré que la substitution d'une addition par un "ou exclusif" pouvait se ramener à forcer le basculement d'un bit du registre contenant les codes d'opérations. Sous cette hypothèse, nous avons montré qu'il est possible de retrouver complètement l'état interne de RABBIT, à un instant donné, à partir de seulement 32 perturbations et au prix de 2^{34} comparaisons.

D'autre part, les résultats obtenus par le biais de notre attaque nous ont permis de nuancer l'analyse initiée par les concepteurs de RABBIT. En effet, ceux-ci avaient conclu que le remplacement de toutes les additions par des "ou exclusifs", dans la fonction de mise à jour, ne baisse pas le niveau de sécurité de RABBIT. En se référant aux performances de notre attaque, nous

pouvons affirmer que les mêmes conclusions ne s'appliquent pas si ces additions sont temporairement remplacées, une par une, par des "ou exclusifs".

Afin de colmater cette vulnérabilité potentielle, nous proposons d'ajouter de la redondance, voire de doubler, le calcul des additions de façon à pouvoir comparer les résultats et détecter les exécutions perturbées. Comme ces additions sont beaucoup moins coûteuses que le calcul d'images par la fonction *"g"* (*i.e.* une mise au carré d'une entrée de 32 bits), une telle contre-mesure ne devrait pas augmenter outre mesure la complexité globale de Rabbit.

Pour conclure, nous pouvons affirmer que les attaques par perturbations représentent une menace réelle pour les implantations de Rabbit. Ainsi, par précaution, nous recommandons aux futurs développeurs de porter une attention particulière à l'implantation de la fonction de mise à jour de Rabbit.

Chapitre 10

Application à l'algorithme Grain-128

Sommaire

10.1 Description de l'attaque

Dans le cadre du projet ANR Odyssée, nous nous sommes intéressés à la sécurité des implantations de Grain-128 vis-à-vis des perturbations. Alors que la résistance de ses implantations face aux attaques par perturbations a déjà été étudiée [AMMA08], nous n'avons pas trouvé de précédents résultats dans le domaine des attaques par faute. Cependant, certaines méthodes génériques ont été développées afin d'analyser les perturbations d'algorithmes de chiffrement à flot basés sur des registres d'état interne à rétroaction linéaire [HS04, AM06]. Malheureusement, la présence de registres à décalage avec rétroaction non-linéaire dans la structure de Grain-128 ne permet pas d'utiliser directement ces méthodes.

Contrairement à notre première attaque contre les implantations de Rabbit (*cf.* Chapitre 9), nous avons étudié la résistance de Grain-128 vis-à-vis des perturbations sous un modèle de faute classique. Dans les sections suivantes, nous décrirons plus précisément ce modèle et nous détaillerons la méthode d'analyse que nous avons imaginée pour retrouver la clé secrète de Grain-128 à partir de quelques suites chiffrantes erronées. Ce travail est le fruit d'une collaboration avec les partenaires du projet Odyssée.

10.1.1 Basculement d'un bit de l'état interne

Le modèle de perturbation que nous avons choisi est inspiré de celui adopté par J. Hoch et A. Shamir [HS04], à savoir le basculement d'un ou de plusieurs bits de la RAM ou des registres internes d'un composant cryptographique. Ces fautes transitoires sont injectées de telle

sorte que l'attaquant ait un contrôle partiel sur le nombre et la position des bits basculés ainsi que sur la position temporelle de la faute dans l'exécution de l'algorithme. Nous supposons, par ailleurs, que l'attaquant est en possession du composant cryptographique embarquant une implantation matérielle de GRAIN-128 et qu'il peut connaître le vecteur d'initialisation ainsi que la suite chiffrante générée par l'algorithme. Dans le cadre de notre application des attaques par perturbation aux implantations de GRAIN-128 nous avons considéré deux variantes de ce modèle, décrites ci-après.

Première variante. Dans la première variante, l'attaquant est supposé pouvoir basculer la valeur d'un seul bit du LFSR (le contrôle de la position n'est pas nécessaire). De plus, il est supposé pouvoir contrôler précisément l'instant auquel la faute est injectée. De tels effets peuvent être obtenus en effectuant des tirs lasers sur le composant cryptographique visé [SA02, Sko06]. Compte tenu de la nature éphémère de la perturbation induite et de la connaissance de l'IV, on suppose que l'attaquant peut réinitialiser le composant dans son état original et reproduire les mêmes fautes pour différentes exécutions de l'appareil.

Comme on étudie une implantation matérielle, le choix de la position temporelle de la faute peut être fait à l'aide du signal d'entrée sortie I/O. Les registres à décalages sont mis à jour régulièrement, de telle sorte qu'un seul bit soit calculé par cycle d'horloge. De cette manière, l'attaquant peut identifier chaque itération dans l'exécution de GRAIN-128 et on peut alors supposer qu'il peut complètement contrôler l'instant auquel il souhaite injecter une faute.

Seconde variante. Un autre modèle, plus restrictif peut être décliné de celui décrit précédemment. Dans cette variante, on considère que l'attaquant peut perturber aléatoirement la fonction linéaire de mise à jour de l'état interne f. Ce modèle est plus restrictif car, dans un cas sur deux, il revient à considérer que le 128-ème bit du LFSR est perturbé. Une telle perturbation pourra être avantageusement utilisée pour déterminer l'état du LFSR de GRAIN-128 (*cf.* Section 10.1.2.3).

10.1.2 Analyse différentielle des perturbations

10.1.2.1 Principe général

La présence de non-linéarité dans la mise à jour des bits d'état interne de GRAIN-128 ne permet pas d'appliquer directement la méthode proposée par J. Hoch et A. Shamir [HS04]. Cependant, nous nous sommes inspirés de cette attaque pour étudier la sécurité des implantations de GRAIN-128 vis-à-vis des perturbations. Le principe général de notre attaque peut se résumer en cinq phases distinctes décrites brièvement ci-dessous :

1. *Caractérisation.* Pendant cette phase, l'attaquant utilise le composant de test qu'il a en sa possession. Le but ici est de trouver l'endroit où le LFSR est stocké sur le composant. Dans la Section 10.1.2.2, nous développerons la méthode que nous avons utilisée pour déterminer quelles cellules du LFSR sont perturbées en étudiant la différence entre les sorties correctes et erronées.
2. *Vérification des hypothèses.* À présent, l'attaquant utilise véritablement le composant contenant la clé secrète cherchée. En se basant sur les résultats obtenus pendant la phase de caractérisation, l'attaquant essaie de reproduire les même fautes. La réussite de cette étape peut nécessiter de réaliser certains ajustements. À la fin de ces deux premières étapes, l'attaquant est supposé capable de perturber le composant comme il le souhaite, sur les bits voulus du LFSR et aux instants souhaités.

3. *Détermination du LFSR.* Pour déterminer le LFSR, nous nous sommes directement inspirés de la méthode décrite dans [HS04]. En quelques mots, le principe consiste à construire et résoudre un système d'équations linéaires en les bits d'état du LFSR. Cette étape sera plus largement décrite dans la Section 10.1.2.3.

4. *Détermination du NFSR.* De part la nature non-linéaire de la fonction de mise à jour du registre que l'on souhaite déterminer, le principe décrit dans cette étape n'est couvert par aucune précédente attaque par perturbation contre un algorithme de chiffrement à flot. Nous montrerons dans la section 10.1.2.4 que des perturbations du LFSR peuvent laisser fuir de l'information exploitable sur le NFSR par la résolution d'un système d'équations.

5. *Extraction de la clé.* Une fois que tout de l'état interne de Grain-128 a pu être déterminé grâce aux deux étapes précédentes, le but de cette étape est de retrouver la clé secrète connaissant le vecteur d'initialisation. Notre méthode d'extraction de clé depuis la connaissance de l'état interne de Grain-128 sera détaillée dans la Section 10.1.2.5

10.1.2.2 Caractérisation des perturbations

Dans un premier temps, nous avons cherché à identifier les effets d'une faute provoquée dans le LFSR sur la suite chiffrante générée par Grain-128 en sortie. Pour ce faire, nous avons répertorié, pour chaque position p de la perturbation dans le LFSR, les motifs obtenus en sortie. Ainsi, à partir de la comparaison entre la suite chiffrante correcte et une suite chiffrante erronée, il sera possible d'identifier quels bits de LFSR ont été infectés. Par exemple, lorsque le bit 127 du LFSR est basculé à un instant t, on peut s'attendre à ce que la faute infecte la suite chiffrante aux instants $t+35$ (via s_{93}), $t+39$ (via b_{89}) et ainsi de suite. Nous noterons ce motif $\{35, 39, 55, 64\}$. Nous avons choisi d'étudier plus particulièrement l'infection due aux bits s_{93}, b_{64}, b_{73} et b_{89} car leur influence est linéaire sur le calcul de la suite chiffrante en sortie.

Afin d'identifier tous les motifs possibles engendrés par les perturbations, nous avons généré les tables de correspondances décrites dans la figure 10.1.2.2. À partir de ces tables, nous avons pu établir un algorithme de correspondance permettant de dire si la position de la faute peut induire un motif donné en pratique. Ainsi, en étudiant la différence entre une suite chiffrante erronée et la suite chiffrante correcte correspondante, il nous est possible de savoir dans quel intervalle du LFSR la faute a été induite. Dans cet algorithme, nous avons utilisé une version simplifiée de l'algorithme Grain-128 nommée ΔGrain. ΔGrain utilise la même fonction de rétroaction, pour le LFSR, que celle de Grain-128 . Cependant, la fonction g de rétroaction du NFSR est remplacée par un simple "ou" entre les bits d'entrée de g. Par ailleurs, la fonction de filtrage h est remplacée par une addition dans les entiers (*cf.* [BCC+09] pour plus de détails).

La présence dans la suite chiffrante de l'un des motifs donnés dans la Table I est une condition nécessaire pour savoir si la faute a été injectée à une position i telle que $0 \leq i \leq 41$ ou $96 \leq i \leq 127$ du LFSR. Lorsque l'attaquant n'arrive pas à identifier l'un des motifs de la Table I, il peut alors déduire que la faute a pu être injectée entre les positions 42 et 95. Par conséquent, s'il trouve une correspondance avec un des motifs de la Table II, il aura une condition suffisante pour identifier la position d'une faute dans le LFSR.

Maintenant que nous avons montré comment identifier la position d'une perturbation en analysant les effets sur la sortie, nous allons montrer comment remonter jusqu'à l'extraction complète des bits contenus dans le LFSR à un instant donné.

Position de la faute	Motifs
$0 \leq i \leq 31$	$\{35+i, 39+i, 55+i, 64+i\}$
$32 \leq i \leq 37$	$\{35+(i-32), 42+(i-32), 46+(i-32), 62+(i-32), 71+(i-32)\}$
$38 \leq i \leq 41$	$\{35+(i-38), 66+(i-38), 73+(i-38), 77+(i-38)\}$
$96 \leq i \leq 127$	$\{3+(i-96), 35+(i-96), 50+(i-96), 61+(i-96)\}$

TABLE I

Position de la faute	Motifs
$42 \leq i \leq 67$	$\{35+(i-42), 66+(i-42), 73+(i-42), 77+(i-42)\}$
$81 \leq i \leq 95$	$\{35+(i-81), 46+(i-81), 67+(i-81), 82+(i-81)\}$

TABLE II

Position de la faute	Motifs
$70 \leq i \leq 80$	$\{35+(i-70), 82+(i-42), 93+(i-42), 98+(i-42)\}$

TABLE III

10.1.2.3 Détermination complète du LFSR

Dans la section précédente, nous avons montré que, quel que soit le bit du LFSR dont la valeur a été basculée, il est toujours possible d'analyser la différence entre les suites chiffrantes correctes et erronées pour retrouver la position de la faute. À présent, le but est d'analyser cette différence pour extraire des équations linéaires en les bits du LFSR.

Dans un premier temps, rappelons comment sont calculés chacun des bits de suite chiffrante z_i. Ceux-ci sont calculés via un *"ou exclusif"* entre certains bits provenant du LFSR/NFSR et la sortie de la fonction de filtrage h :

$$\begin{aligned} z_i \quad = \quad & b_{i+2} \oplus b_{i+15} \oplus b_{i+36} \oplus b_{i+45} \oplus b_{i+64} \oplus b_{i+73} \oplus b_{i+89} \\ & \oplus s_{i+93} \oplus b_{i+12}.s_{i+8} \oplus s_{i+13}.s_{i+20} \\ & \oplus b_{i+95}.s_{i+42} \oplus s_{i+60}.s_{i+79} \oplus b_{i+12}.b_{i+95}.s_{i+95} \end{aligned}$$

D'après cette expression, on peut observer qu'à l'itération i, si un bit parmi s_{13}, s_{20}, s_{60} ou s_{79} est perturbé, la différence entre une suite chiffrante correcte et erronée correspond à la valeur d'un bit du LFSR. Par exemple, si $s_{13} \oplus \hat{s}_{13} = \sigma_{13} = 1$ alors la différence entre les deux suites chiffrantes renvoie directement la valeur s_{20}. Par conséquent, le nombre de bits du LFSR pouvant être retrouvés par une analyse différentielle des perturbations dépend de la position de la faute ainsi que du nombre d'itérations réalisées après l'injection. L'algorithme 10 décrit la méthode que nous avons utilisée pour évaluer le nombre de bits pouvant être récupérés ainsi. Dans un souci de clarté, nous avons ajouté dans cet algorithme la notation σ_i correspondant à la différence entre $s_i \oplus \hat{s}_i$.

Brièvement, le premier bloc conditionnel compte le nombre de fois où il est possible de retrouver la valeur d'un bit du LFSR. Le second bloc conditionnel compte, quant à lui, le nombre d'équations linéaires en les bits du LFSR. À partir de cet algorithme, nous avons dérivé une méthode "`CalculRang`" permettant de calculer le rang du système d'équations extrait en analysant les effets d'une perturbation.À chaque fois qu'une équation est trouvée par l'algorithme précédent, celle-ci est ajoutée comme vecteur colonne d'une matrice $\mathcal{M}$ de 128 lignes

Algorithme 10 Exponentiation binaire *"Left-To-Right"*

ENTRÉES : Position de la faute p, nombre d'itérations depuis la faute NC
SORTIE : Nombre de bits de LFSR retrouvés KB

1: `Init`(ΔGrain, 0)
2: **Pour** $i = 0$ **à** NC **faire**
3: **Si** $\big[\big((\sigma_{13} = 1)$ ou $(\sigma_{20} = 1)$ ou $(\sigma_{60} = 1)$ ou $S(\sigma_{79} = 1)\big)$ ET $(\Delta\text{Grain output} = 1)\big]$ **alors**
4: $KB \leftarrow KB + 1$
5: **Fin Si**
6: **Si** $\big[(\sigma_{13} \oplus \sigma_{20} = 1)$ ET $(\sigma_{60} \oplus \sigma_{79} = 1)$ ET $(\Delta\text{Grain output} = 2)\big]$ **alors**
7: $KB \leftarrow KB + 1$
8: **Fin Si**
9: `Exécute`(ΔGrain)
10: **Fin Pour**
11: **Retourner** KB

(correspondant à chacun des bits du LFSR).À la fin de l'exécution de cette méthode, le nombre de colonnes de $\mathcal{M}$ correspond au nombre d'équations trouvées. De fait, le rang de la matrice nous donne le nombre d'équations indépendantes contenues dans le système linéaire ainsi formé.

La méthode `CalculRang` peut être étendue en concaténant dans $\mathcal{M}$ toutes les équations extraites à partir de plusieurs perturbations. Ainsi, nous avons pu déterminer le nombre de perturbations nécessaires pour extraire tous les bits du LFSR, pour un nombre d'itérations donné. Les résultats obtenus par cette méthode sont résumés dans le tableau 10.1.

En étudiant la table, on remarque qu'en moyenne 24 fautes injectées consécutivement permettent de déterminer entièrement l'état du LFSR. D'autre part, plus les fautes sont espacées temporellement et plus elles apportent d'informations. Ce phénomène n'est pas surprenant puisque, plus les fautes sont espacées et moins les équations ont des chances d'être dépendantes.

Position de la faute	Nombre de fautes consécutives	Nombre de fautes toutes les 4 itérations	Nombre de fautes toutes les 30 itérations
0 – 6	90	62	44
7 – 12	81	32	19
13 – 19	41	23	16
20 – 37	34	19	15
38 – 59	20	17	11
60 – 69	16	17	11
70 – 78	14	15	10
79 – 80	11	14	9
81 – 127	6	8	6
Moyenne	23,8	17,3	12,1

TABLE 10.1 – Fautes nécessaires pour retrouver l'état du LFSR de GRAIN-128

Par cette étude, nous avons évalué le nombre de fautes qu'il faut provoquer sur un composant cryptographique pour espérer extraire tous les bits du LFSR. Dans un souci d'efficacité, nous nous sommes volontairement limités à l'étude des équations linéaires. Malgré tout cette étude a

mis en évidence qu'en moyenne 24 fautes consécutives suffisent à déterminer le LFSR. Cependant, l'état interne de GRAIN-128 est aussi composé d'un registre à rétroaction non-linéaire. Bien qu'aucune méthode pour exploiter des fautes pour déterminer des NFSR n'ait été publiée à notre connaissance, nous avons contourné cette difficulté en continuant d'exploiter des fautes provoquées dans le LFSR. La méthode que nous avons mis au point pour déterminer le reste de l'état interne de GRAIN-128 est décrite dans la section suivante.

10.1.2.4 Détermination complète du NFSR

Une fois tous les bits du LFSR obtenus, pour un instant donné, il reste encore à déterminer les valeurs contenues dans le NFSR à ce même instant. Notons que lorsque le LFSR a été défini à l'instant t, la linéarité de la fonction de rétroaction permet de déterminer l'état du LFSR à n'importe quel instant. Dans cette section, nous allons décrire la méthode que nous avons mis au point pour retrouver tous les bits du NFSR avec notre modèle de perturbation.

Supposons que tous les bits nécessaires ont été déterminés grâce à l'analyse précédente. Alors, il est assez aisé de déduire des équations linéaires en les bits du NFSR à partir d'une suite chiffrante correcte. En effet, le seul monôme dans l'expression de z_i est $b_{i+12}b_{i+95}s_{i+95}$. Aussi, quand $s_{i+95} = 0$, alors z_i peut être exprimé linéairement en fonction des bits du NFSR. Par ailleurs, les différences entre les suites chiffrantes correctes et erronées, déjà utilisées pour déterminer l'état du LFSR, peuvent être utilisées à nouveau pour obtenir des équations linéaires supplémentaires.

Supposons qu'une faute ait été injectée à une position p du LFSR, $0 \leqslant p \leqslant 127$, de telle sorte que la valeur du bit s_{i+p} soit basculée. À l'instant $t = i + p + 32$, cette faute se sera propagée jusqu'au 96-ème bit du NFSR sans avoir infecté la fonction de rétroaction g (car il n'y a pas de branches de rétroaction avant). Par conséquent, en analysant les différences entre les suites chiffrantes correctes et erronées à l'instant $t = i + p + 32$, on obtient des équations en les valeurs d'état du NFSR à l'instant t. Certaines équations linéaires apparaissent lorsque la perturbation se propage jusque dans les bits 12 et 95 du NFSR, grâce à la simplification du monôme $b_{i+12}b_{i+95}s_{i+95}$. D'autres équations peuvent être obtenues quand la faute infecte des bits du NFSR ayant une contribution "quadratique" dans la rétroaction. Par exemple, lorsque la faute infecte le 84-ème bit du NFSR à un instant t', la différence $b_{t'+84}$ résultera du calcul du monôme $b_{t'+68}b_{t'+84}$ pour $b_{t'+128}$. Finalement, nous pourrons extraire la valeur de ce bit depuis la différence entre les suites chiffrantes puisque, lorsque la faute arrive en position 89 dans le NFSR, elle contribuera linéairement à la génération de la suite chiffrante.

Après avoir décrit la méthodologie générale pour obtenir des équations en les bits du NFSR grâce à des perturbations induites dans le LFSR, voyons quelle quantité d'information il est possible d'extraire de chacune des perturbations. Cette étude est résumée dans les paragraphes suivants.

Nombre d'équations obtenues depuis une seule perturbation. Nous avons estimé empiriquement le nombre d'équations linéaires en les bits du NFSR obtenues suivant les valeurs contenues dans le LFSR (normalement déjà retrouvées par l'attaquant à cette étape de l'attaque) ainsi que les positions des fautes injectés dans ce même LFSR. Pour ce faire, nous avons mis au point et simulé l'algorithme de comptage suivant. Cet algorithme utilise deux instances de GRAIN-128 , l'une correcte et l'autre perturbée :

1. Initialisation du LFSR avec des valeurs aléatoires
2. Basculement volontaire d'un bit de LFSR à une position p tel que $0 \leqslant p \leqslant 127$

3. Exécution de $32 + p$ itérations du LFSR correct et de celui perturbé
4. Initialisation du NFSR correct avec les valeurs formelles $b_0, b_1, \ldots, b_{96}, \ldots, b_{127}$ et du NFSR perturbé correspondant à la faute injectée dans le LFSR : $b_0, b_1, \ldots, b_{96} \oplus 1, \ldots, b_{127}$
5. Exécution de 117 itérations des deux instances formelles de Grain-128 constituées par les 2 NFSR et comptage des équations linéaires en les variables du NFSR extraites de la différence entre les suites chiffrantes correctes et erronées.

Nous avons choisi de nous limiter à 117 itérations seulement car les expressions linéaires obtenues pour des itérations ultérieures se sont révélées plutôt lourdes et peu linéaires. D'autre part, le fait de calculer $32+p$ itérations du LFSR avant de réaliser l'analyse formelle permet de s'assurer que la propagation de la faute va permettre de produire rapidement des équations exploitables en sortie. Le tableau 10.2 décrit le nombre moyen d'équations linéaires obtenues suivant la position p de la faute dans le LFSR.

En observant les résultats obtenus, on peut remarquer que le nombre d'équations linéaires

Position de la faute dans le LFSR	Nombre d'équations linéaires en les variables du NFSR
0 à 6	10,97
7 à 37	10,09
38 à 69	11,64
70 à 80	12,43
81 à 95	16,32
96 à 127	21,51

Table 10.2 – Nombre d'équations linéaires en les bits du NFSR suivant le position de la faute dans le LFSR

obtenues dépend véritablement de la position du bit de LFSR basculé. Globalement, ce nombre d'équations augmente avec l'indice de la position de la faute. Ce résultat n'est pas surprenant puisque, si les bits de LFSR sont déjà déterminés de nombreuses équations linéaires peuvent être obtenues grâce à la simplification de la fonction de filtrage de la sortie. Pour résumer grossièrement, chaque perturbation du LFSR permet de retrouver une dizaine de bits de NFSR. Étudions à présent le nombre de fautes nécessaires à l'extraction complète du NFSR.

Estimation du nombre de fautes pour retrouver le NFSR. À présent, nous souhaitons évaluer le nombre de perturbations qu'il faut réaliser dans le LFSR pour retrouver l'intégralité du NFSR. Ce résultat ne peut pas se déduire directement de l'étude précédente car les ensembles de valeurs du NFSR, tirés de différentes erreurs réalisées sur le LFSR, ne sont pas indépendants les uns des autres. Afin de réaliser cette estimation, nous avons mis au point un algorithme récursif permettant de retrouver les bits du NFSR depuis les équations extraites à la fois de la suite chiffrante correcte mais aussi de la différence entre les suites chiffrantes correcte et erronée. Le principe de cet algorithme est le suivant :

1. Pour chaque suite chiffrante erronée
   (a) Calculer la différence avec la suite chiffrante correcte
   (b) Utiliser l'algorithme de comptage pour collecter les équations linéaires et quadratiques simples en les variables de NFSR obtenues pour une perturbation à l'instant t. Ces équations forment un système (S).

(c) Ajouter à (S) les équations linéaires directement obtenues depuis la suite chiffrante correcte.

(d) Calculer la base de Groebner de (S) et résoudre les équations en une variable

(e) Utiliser la détermination de certains bits de NFSR pour extraire de nouvelles équations linéaires ou quadratiques simples depuis la suite chiffrante et la différence entre celle correcte et celle erronée. Retourner à l'étape précédente si de nouvelles équations apparaissent.

2. Retourner (S)

Dans nos simulations, nous avons retenu, en guise d'équations quadratiques simples, les équations ayant au plus deux monômes quadratiques. Cela permet au calcul des bases de Groebner d'être relativement efficace. Cet algorithme nous a permis d'évaluer le nombre de perturbations à réaliser pour retrouver complètement le NFSR. Pour des fautes consécutives dans le LFSR, les résultats obtenus sont résumés par le tableau 10.3. Le tableau 10.4, quand à lui, résume les résultats obtenus pour des fautes non consécutives dans le LFSR.

Position de la faute dans le LFSR	Nombre de fautes pour retrouver le NFSR
0 à 6	8.29
7 à 37	7.48
38 à 69	6.75
70 à 80	6.09
81 à 95	4.53
96 à 127	4.40

Table 10.3 – Fautes consécutives pour retrouver le NFSR de Grain-128

Espacement des fautes	Nombre de fautes min/max	Nombre correspondant d'équations linéaires indépendantes
2	4/6	108/99
5	3/9	103/102
10	3/12	106/79
20	5/8	104/59
30	4/5	45/79

Table 10.4 – Nombres d'équations pour des fautes non consécutives sur Grain-128

Par conséquent, lorsque le LFSR est déterminé, il est assez facile de retrouver complètement le NFSR à partir de quelques perturbations induites dans le LFSR. Ainsi, localiser la zone mémoire du composant à évaluer contenant le LFSR de Grain-128 suffit pour réaliser les perturbations adéquates menant à l'extraction complète de l'état interne de cet algorithme de chiffrement à flot. On pourra remarquer que, contrairement au cas du LFSR, il est quand même plus intéressant de réaliser des fautes proches (d'un point de vue temporel) que des fautes éloignées puisque les premières apportent plus d'information sur le NFSR. Notons enfin, pour appuyer encore la faisabilité de notre attaque, que le nombre de perturbations à réaliser est relativement faible (de l'ordre de la dizaine) et que l'analyse ne prend que quelques minutes sur un PC.

10.1.2.5 Extraction de la clé secrète

Supposons à présent que l'intégralité de l'état interne de Grain-128 a pu être déterminé grâce à l'analyse que nous avons proposée dans les sections précédentes. Notons $S_t = \{s_t, \ldots, s_{t+127}, b_t, \ldots, b_{t+127}\}$ les bits d'état interne de Grain-128 à la t-ème itération de l'algorithme. À partir de cet état, il est possible de remonter dans l'exécution de Grain-128 en calculer les états antérieurs. En effet, durant la phase d'initialisation, la sortie de la fonction h renvoyée est xorée avec les entrées du LFSR et du NFSR. Pour retrouver l'état à l'instant $t-1$, nous commençons par calculer z_{t-1} :

$$\begin{aligned} z_{t-1} &= h(b_{t+11}, s_{t+7}, s_{t+12}, s_{t+19}, b_{t+94}, s_{t+41}, s_{t+59}, s_{t+78}, s_{t+94}) \oplus s_{t+92} \\ &\oplus\ b_{t+1} \oplus b_{t+14} \oplus b_{t+35} \oplus b_{t+44} \oplus b_{t+63} \oplus b_{t+72} \oplus b_{t+88} \end{aligned} \tag{10.1}$$

Ainsi, on a encore :

$$s_{t-1} = z_{t-1} \oplus s_{t+127} \oplus s_{t+6} \oplus s_{t+37} \oplus s_{t+69} \oplus s_{t+80} \oplus s_{t+95} \text{ et} \tag{10.2}$$

$$\begin{aligned} b_{t-1} &= z_{t-1} \oplus s_{t-1} \oplus b_{t+127} \oplus b_{t+25} \oplus b_{t+55} \oplus b_{t+90} \oplus b_{t+95} \\ &\oplus\ b_{t+2}b_{t+66} \oplus b_{t+10}b_{t+12} \oplus b_{t+16}b_{t+17} \oplus b_{t+26}b_{t+58} \\ &\oplus\ b_{t+39}b_{t+47} \oplus b_{t+60}b_{t+64} \oplus b_{t+67}b_{t+83} \end{aligned} \tag{10.3}$$

Il ne restera alors plus qu'à décaler les bits dans chacun des deux registres à décalage pour remonter un cran dans l'exécution. Maintenant que nous avons montré comment calculer l'état interne S_{t-1} à partir de S_t, il suffit de répéter ce calcul pour retrouver tous les états internes antérieurs S_{t-i} pour $0 \leqslant i \leqslant t$. Or, à l'initialisation, les bits de clé sont copiés dans le NFSR. Par conséquent, en remontant jusque l'état interne initial, il suffit de lire la valeur des bits du NFSR pour extraire la clé secrète.

10.2 Conclusion

Dans ce chapitre, nous avons montré que les implantations de Grain-128 sont vulnérables aux perturbations. Sous un modèle de perturbation classiquement utilisé pour l'étude des implantations d'algorithmes de chiffrement à flot, nous avons prouvé qu'en moyenne, 28 perturbations consécutives du LFSR permettent de retrouver la clé secrète en quelques minutes sur un PC. L'utilisation de perturbations est d'autant plus intéressante, que la meilleure attaque connue contre Grain-128 jusqu'alors était la recherche exhaustive. D'autre part, et contrairement à ce que l'on aurait pu penser, l'utilisation de fonctions non-linéaires pour la mise à jour d'un registre d'état ainsi que pour le filtrage de la sortie n'est pas suffisante pour empêcher les attaques par fautes.

Cependant, nous pouvons remarquer que notre attaque n'exploite que des fautes commises dans le LFSR. Et, puisque nous ne connaissons pas d'attaques par perturbation visant des registres à rétroaction non-linéaire, nous proposons, en guise de contremesure, de protéger la partie linéaire de Grain-128 seulement. De manière générale, protéger un algorithme de chiffrement à flot contre les perturbations nécessite de protéger entièrement les registres d'état interne. Par conséquent, cette simple observation permet de protéger efficacement les implantations matérielles de Grain-128 vis-à-vis des perturbations que d'autres algorithmes de chiffrement à flot dans le sens où seul le LFSR devra être protégé.

Une contremesure très simple consisterait à dupliquer le LFSR dans un autre registre "miroir" qui sera lui aussi mis à jour à chaque coup d'horloge. Un comparateur pourra ainsi vérifier que

le contenu des deux registres est identique pendant toute la durée de l'exécution. Lorsqu'une différence est détectée, le circuit lève une interruption visant à stopper l'exécution de l'algorithme de chiffrement à flot. De cette manière, aucun bit de suite chiffrante ne sera renvoyé après la détection de la faute ce qui rend toute exploitation impossible.

Une contremesure plus complexe consisterait à ajouter de la redondance au LFSR de telle sorte que l'opération de rétroaction du LFSR soit remplacée par une transformation linéaire incluant une méthode de détection d'erreurs adaptée. De cette manière, un certain nombre de perturbations induites dans le LFSR seront détectées (voire corrigées) suivant les caractéristiques du code correcteur choisi.

Quatrième partie

Analyse de contre-mesures DFA

Chapitre 11

Cas du RSA-CRT

Sommaire

11.1 Présentation de RSA-CRT

11.1.1 Description générale

L'implantation CRT de RSA [QC82] utilise le théorème des restes chinois pour augmenter la vitesse d'un déchiffrement ou d'une signature mais aussi réduire la taille des données stockées en mémoire. En termes d'opérations binaires, cette implantation est théoriquement quatre fois plus rapide que la version standard. C'est pourquoi cette implantation de RSA est, en pratique, très largement déployée sur les systèmes embarqués. Dans ce paragraphe, nous rappellerons les détails de la signature en mode CRT.

11.1.2 Signature en mode CRT

La signature en mode CRT nécessite de connaître les facteurs secrets p et q. On commence par exprimer la valeur de l'exposant d modulo $\varphi(p)$ et $\varphi(q)$:

$$\begin{cases} d_p \equiv d \bmod (p-1) \\ d_q \equiv d \bmod (q-1) \end{cases} \tag{11.1}$$

L'étape de recombinaison par le Théorème des Restes Chinois, ou *recombinaison CRT*, peut être réalisée en utilisant la formule de Garner [MOVR97] :

$$S = CRT(S_p, S_q) = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p) \text{ avec } \begin{cases} i_q \equiv q^{-1} \bmod p \\ S_p \equiv (\dot{m})^{d_p} \bmod p \\ S_q \equiv (\dot{m})^{d_q} \bmod q \end{cases} \tag{11.2}$$

11.2 Contre-mesures DFA : cas du RSA-CRT

Le but de cette partie est de mettre en évidence toute la difficulté d'élaborer des contre-mesures pour protéger les implantations d'algorithmes cryptographiques contre les perturbations. Dans ce but, nous avons choisi de développer le cas de l'algorithme de signature standard RSA, dans son mode CRT. Dans un premier temps, nous décrirons les premières attaques par perturbation sur les implantations de cet algorithme de signature, puis nous verrons comment l'évolution des attaques a favorisé l'élaboration de concepts innovants pour protéger ces implantations vis-à-vis des fautes.

11.2.1 Attaque de Bellcore et amélioration

C'est en attaquant la variante CRT du RSA que les trois chercheurs de Bellcore ont réellement mis en évidence le potentiel des attaques par perturbation. En effet, ils ont montré dans [BDL97, BDL01] que si une seule erreur se produit au moment de calculer s_p ou bien s_q, alors, connaissant la signature correcte et la signature perturbée, il est possible de factoriser très facilement N. Cette attaque est d'autant plus intéressante qu'elle ne nécessite pas de modéliser précisément la faute injectée. Par ailleurs, les hypothèses d'attaques ont été réduites à la seule connaissance d'un texte clair et de sa signature perturbée par A. Lenstra. Cette amélioration a été publiée dans [Len96, JLQ99].

Principe de l'attaque. Pour illustrer cette attaque, supposons qu'une faute se soit produite pendant le calcul de S_p. Notons le résultat correspondant $\hat{S}_p$. Ce résultat erroné est ensuite utilisé lors de la recombinaison CRT : $\hat{S} = CRT\ (\hat{S}_p, S_q)$. Or, comme $S_p \equiv S \bmod p$ et $S_q \equiv S \bmod q$, on a alors $\hat{S} \equiv S \bmod q$, en revanche, $\hat{S} \not\equiv S \bmod p$. On en déduit alors que $p \nmid \hat{S} - S$ alors que $q \mid \hat{S} - S$. Ce déséquilibre induit par la perturbation peut ensuite être exploité pour retrouver le facteur secret q grâce à un simple calcul de *pgcd* :

$$q = pgcd\ \left(\left(\hat{S} - S\right) \bmod N, N\right) \tag{11.3}$$

Par ailleurs, l'autre facteur premier du modulo RSA p peut aussi être calculé : $p = N/q$.

Amélioration de Lenstra. Comme pour l'attaque précédente, considérons qu'une faute s'est produite pendant le calcul de S_p et notons le résultat erroné $\hat{S}_p$. Cette erreur se propage alors dans la recombinaison CRT : $\hat{S} = CRT\ (\hat{S}_p, S_q)$. Le principe de l'amélioration proposée par A. Lenstra consiste à considérer le résultat de $\hat{S}^e$ modulo p et q. On a montré précédemment que la conséquence de la perturbation est que $\hat{S} \equiv S \bmod q$ alors que $\hat{S} \not\equiv S \bmod p$. En ne connaissant qu'une seule signature erronée et le texte clair associé m , on peut obtenir le paramètre secret q :

$$q = pgcd\ \left(\left(\hat{S}^e - \dot{m}\right)\ \bmod N, N\right) \tag{11.4}$$

Comme précédemment, l'attaquant retrouve aussi l'autre facteur secret p en calculant $p = N/q$.

Discussion de contre-mesures naïves. Afin de parer l'attaque précédente, une contre-mesure évidente serait d'utiliser la clé publique (e, N) pour vérifier que le mécanisme de signature n'a pas été perturbé (*i.e.* vérifier que $S^e \bmod N \equiv m$). Malheureusement, cette contre-mesure présente deux défauts majeurs qui la rend inutilisable en pratique. Premièrement, elle nécessite de pouvoir exécuter une seconde exponentiation modulaire. Or, cette opération peut s'avérer très coûteuse si l'exposant public e est grand. Par ailleurs, l'exposant public e n'est pas toujours stocké sur les périphériques cryptographiques [33](*c.f.* Signature RSA dans l'API JavaCard version 2.2.2). Une autre contre-mesure tout aussi naturelle serait de dupliquer l'exécution de la signature afin de détecter si l'une des deux a été perturbée. Bien évidemment, cette contre-mesure n'est pas acceptable en pratique car elle double la complexité calculatoire du schéma de signature. De plus, le résultat de la comparaison des deux exécutions pourrait lui aussi être corrompu, de façon à ce que la présence d'une faute dans une des signatures ne soit pas détectée. Par exemple, un attaquant pourrait forcer à 1 la valeur d'une variable booléenne utilisée pour comparer les deux signatures dans un branchement conditionnel. Ainsi, le résultat du branchement sera toujours le même, quelle que soit la valeur des signatures. Par conséquent, il a fallu imaginer des concepts astucieux pour protéger les implantations CRT de RSA contre les perturbations.

Dans la partie suivante, nous décrirons de manière chronologique comment l'évolution des attaques a favorisé l'émergence de nouvelles méthodes de protection et inversement.

11.2.2 Contre-mesures par extension aléatoire du module

Dans [Sha97], A. Shamir propose de lutter contre l'attaque de Bellcore en ajoutant de l'aléa dans le calcul de S_p et S_q. Cet aléa permet de vérifier, avant la recombinaison CRT, qu'aucune erreur ne s'est glissée pendant l'exécution des exponentiations. L'algorithme de signature modifié par Shamir est décrit ci-dessous.

$$\text{Soit } t \text{ un petit nombre entier, on calcule : } \begin{cases} S_{pt} \equiv m^d \bmod (p \cdot t) \\ S_{qt} \equiv m^d \bmod (q \cdot t) \end{cases} \tag{11.5}$$

Ensuite, on vérifie l'absence d'erreur avant la recombinaison CRT :

1. Si $S_{pt} \equiv S_{qt} \bmod t$, renvoyer $S = CRT\ (S_{pt}, S_{qt})$,
2. Sinon, renvoyer `Erreur : Faute détectée`.

33. Cet inconvénient pourrait être évité si la méthode de M. Joye, publiée récemment, peut être utilisée pour construire des modules RSA embarquant l'exposant public [Joy09]. De cette manière, il n'est plus nécessaire de stocker l'exposant e car celui-ci peut être directement retrouvé à partir du module N.

L'inconvénient majeur de cette contre-mesure est qu'elle nécessite de connaître d. Or, en pratique, sur les composants cryptographiques qui intègrent le RSA-CRT, seuls $d_p \equiv d \bmod (p-1)$ et $d_q \equiv d \bmod (q-1)$ sont disponibles (ce qui signifie que d doit être recalculé). C'est ce qu'ont noté M. Joye, P. Paillier et S.-M. Yen dans [JPY01]. Dans ce même article, les auteurs proposent de vérifier indépendemment chacune des deux exponentiations. Enfin, à *CHES 2008*, D. Vigilant proposa une méthode basée sur l'extension du module par le carré d'un aléa [Vig08]. À ce jour, aucune vulnérabilité face aux perturbations n'a été publiée contre les implantations de cet algorithme.

11.2.3 Perturbation sur la recombinaison CRT

Bien que les contre-mesures précédentes protègent le RSA-CRT contre la perturbation des deux exponentiations modulaires, elles sont vulnérables face à la perturbation de l'opération de recombinaison CRT. Cette vulnérabilité a été exploitée par C. Aumüller *et al.* [ABF+02]. En effet, si une faute transitoire se produit à l'étape de recombinaison sur S_{pt}, S_{qt} ou encore i_{qt}, la méthode de Shamir ne détecte pas la perturbation. Dans ce cas, l'attaquant peut facilement factoriser le modulo RSA N à partir de la signature erronée $\hat{S}$ en calculant $pgcd\left(\left(\hat{S}^e - m\right) \bmod N, N\right)$. Toujours dans [ABF+02], les auteurs proposent une amélioration de l'algorithme de Shamir qui permet de résister à cette dernière attaque. Malgré tout, S.-M. Yen, S. Moon et J. Ha ont montré dans [YMH02] que cette dernière amélioration reste vulnérable face à l'injection de fautes permanentes au moment de la recombinaison CRT.

11.2.3.1 Calcul Pathogène

Le principe du *calcul pathogène*[34] a été introduit en 2001 par S.-M. Yen *et al.* [YKLM01]. Cette contre-mesure propose de modifier l'algorithme RSA-CRT classique de telle sorte que si une faute modifie une des composantes CRT (S_p ou bien S_q), la faute contamine aussi l'autre composante. Ainsi, si $\hat{S} \not\equiv S \bmod q$ alors $\hat{S} \not\equiv S \bmod p$, ce qui rend l'exploitation par calcul du *pgcd* inefficace. Afin d'illustrer ce principe, nous allons décrire le premier protocole proposé dans [YKLM01]. Avant de commencer toute signature, les clés RSA sont légèrement modifiées. La clé privée d est remplacée par une clé privée $d_r = d - r$ à laquelle on retranche un petit entier r de telle sorte que $pgcd(r, \varphi(N)) = 1$ et que $e_r = d_r^{-1} \bmod \varphi(N)$ soit aussi un petit entier. Notons que e ne doit pas être forcément petit pour que la dernière condition soit satisfaite. Les détails du reste du calcul de la signature sont décrits ci-dessous.

1. L'exécution commence par le calcul de :

$$k_p = \lfloor \frac{m}{p} \rfloor \tag{11.6}$$

$$k_q = \lfloor \frac{m}{q} \rfloor \tag{11.7}$$

2. Elle se poursuit par les exponentiations à la puissance d_r suivantes :

$$\begin{aligned} S_p &\equiv \dot{m}^{d_r} \bmod p \\ S_q &\equiv \hat{m}^{d_r} \bmod q \end{aligned} \tag{11.8}$$

où

$$\hat{m} = ((S_p{}^{e_r} \bmod p) + k_p \cdot (p - q)) \bmod q \tag{11.9}$$

34. Traduction de *Infective Computation* dans la littérature

3. Enfin le calcul de la signature se termine par une recombinaison CRT du résultat des demi-exponentiations :
$$S = CRT(S_p, S_q) \cdot \tilde{m}^r \bmod N \tag{11.10}$$
où
$$\tilde{m} = ({S_q}^{e_r} \bmod q) + k_q \cdot q \tag{11.11}$$

Un autre protocole utilisant le calcul pathogène a été proposé dans [YKLM01]. Malheureusement, ces deux protocoles furent cassés quelques années plus tard par S.-M. Yen et D. Kim dans [YK04, YKM06]. Malgré cela, le principe du calcul pathogène reste considéré comme une manière élégante et prometteuse pour protéger efficacement les implantations CRT de RSA.

11.2.3.2 Algorithme *BOS*

Dans [BOS03], les chercheurs J. Blömer, M. Otto et J.-P. Seifert proposent une variante de l'algorithme de Shamir supposée résister aux attaques par perturbation de l'opération de recombinaison CRT. La modification de cette opération ainsi que l'utilisation du calcul pathogène [YKLM01] permettent à l'algorithme BOS de résister à toutes les attaques par perturbation connues jusqu'alors. La suite de ce paragraphe explique le principe de l'algorithme.
Soient t_1 et t_2 deux entiers aléatoires de longueur κ-bits soigneusement choisis [35] :

1. On précalcule et stocke en mémoire les valeurs suivantes :
$$t_1 \cdot p,\ t_2 \cdot q,\ t_1 \cdot t_2 \cdot N, \tag{11.12}$$
$$\begin{aligned} d_1 &\equiv d \bmod \varphi(t_1 \cdot p),\ e_1 \equiv {d_1}^{-1} \bmod \varphi(t_1 \cdot p), \\ d_2 &\equiv d \bmod \varphi(t_2 \cdot q),\ e_2 \equiv {d_2}^{-1} \bmod \varphi(t_2 \cdot q), \end{aligned} \tag{11.13}$$
2. Ensuite, on calcule la signature :
$$S^* = CRT\ (S_p^*, S_q^*) \bmod (t_1 \cdot t_2 \cdot N),\ \text{avec}\ \begin{cases} S_p^* \equiv m^{d_1} \bmod (t_1 \cdot p) \\ S_q^* \equiv m^{d_2} \bmod (t_2 \cdot q) \end{cases} \tag{11.14}$$
3. Enfin, l'algorithme renvoie la signature :
$$S \equiv (S^*)^{c_1 c_2} \bmod N\ avec \begin{cases} c_1 \equiv (m - (S^*)^{e_1} + 1) \bmod (t_1 \cdot p) \\ c_2 \equiv (m - (S^*)^{e_2} + 1) \bmod (t_2 \cdot q) \end{cases} \tag{11.15}$$

Remarquons que dans le cas d'une exécution sans erreur, $c_1 = c_2 = 1$ et par suite, $S = S^* \bmod N$. D'un point de vue plus pratique, cette méthode présente le même inconvénient que la contre-mesure de Shamir : l'exposant secret d n'est pas toujours stocké tel que sur les périphériques cryptographiques.

Enfin, D. Wagner a montré dans [Wag04] que cette contre-mesure reste vulnérable face aux attaques par perturbation. L'attaque consiste à perturber la lecture du message m, sur un octet, au moment de calculer S_p^*. Cette perturbation produit une faute transitoire sur la valeur m puisque la valeur stockée en mémoire reste inchangée. Par conséquent, la signature réalisée par l'algorithme BOS sera erronée :
$$\hat{S} \equiv (\hat{S^*})^{\hat{c_1}} \bmod N \tag{11.16}$$

35. D'après [JC05], t_1 et t_2 doivent satisfaire les propriétés suivantes : (i) $pgcd(t_1, t_2) = 1$, (ii) $pgcd(d, \varphi(t_1)) = pgcd(d, \varphi(t_2)) = 1$, (iii) $t_1 = t_2 = 3 \bmod 4$ (iv) t_1 et t_2 ne sont pas des carrés, (v) $t_2 \nmid (t_1.p).[(t_1.p)^{-1} \bmod (t_2.q)]$

sachant que $\hat{S^*} \equiv (S^*) \bmod q$ mais que $\hat{S^*} \not\equiv (S^*) \bmod p$. En faisant des hypothèses sur la valeur erronée du message, l'attaquant calcule la valeur de $\hat{c_1}$ correspondante et tente de factoriser N en calculant $pgcd\ ((\hat{S}^e - m^{\hat{c_1}}) \bmod N, N)$. Selon D. Wagner [Wag04], la probabilité de succès de cette attaque est de 4% pour un RSA de 1024 bits et des nombres aléatoires de longueur $\kappa = 80$ bits. Cependant, J. Blömer et M. Otto proposent de modifier légèrement leur précédente contre-mesure pour résister à cette attaque [BO06]. La modification consiste à ajouter de l'aléa dans le calcul de c_1 et c_2 de façon à ce que leur valeur dépende le moins possible de la faute :

$$\begin{cases} c_1 \equiv (m - (S^*)^{e_1}) \cdot R_1 + 1 \bmod (t_1 \cdot p) \\ c_2 \equiv (m - (S^*)^{e_2}) \cdot R_2 + 1 \bmod (t_2 \cdot q) \end{cases} \tag{11.17}$$

où R_1 et R_2 sont deux aléas de taille κ bits. À l'heure actuelle, on ne connaît pas d'attaque contre cette dernière version de l'algorithme BOS.

11.2.3.3 Algorithme de Ciet & Joye

Suite aux attaques de D. Wagner sur l'algorithme *BOS*, M. Ciet et M. Joye ont aussi proposé une version améliorée de l'algorithme BOS [JC05]. Son principe est rappelé ci-dessous.
Soient r_1 et r_2 deux entiers aléatoires de longueur κ-bits soigneusement choisis, r_3 un entier aléatoire de longueur l-bits :

1. On précalcule et stocke en mémoire les valeurs suivantes :

$$p^* = r_1 \cdot p,\ q^* = r_2 \cdot q,\ i_q^* = (q^*)^{-1} \bmod p^*,\ N = p \cdot q, \tag{11.18}$$

2. Ensuite, on calcule les paramètres CRT :

$$\begin{aligned} S_p^* &\equiv m^{d_p} \bmod p^*,\ s_2 \equiv m^{d_p} \bmod \varphi(r_2) \bmod r_2 \\ S_q^* &\equiv m^{d_q} \bmod q^*,\ s_1 \equiv m^{d_p} \bmod \varphi(r_1) \bmod r_1 \end{aligned} \tag{11.19}$$

3. Ces paramètres sont ensuite recombinés avec le Théorème des Restes Chinois pour calculer :

$$S^* = CRT(S_{p^*}, S_{q^*})$$

4. Enfin, l'algorithme renvoie la signature :

$$S \equiv (S^*)^\gamma \bmod N \text{ avec } \begin{cases} c_1 \equiv (S^* - s_1 + 1) \bmod r_1 \\ c_2 \equiv (S^* - s_2 + 1) \bmod r_2 \\ \gamma = \lfloor \dfrac{(r_3 \cdot c_1 + (2^l - r_3) \cdot c_2)}{2^l} \rfloor \end{cases} \tag{11.20}$$

Notons une nouvelle fois que, pour un calcul sans erreur :

$$\gamma = c_1 = c_2 = 1 \tag{11.21}$$

Une analyse de sécurité de cet algorithme est réalisée par les concepteurs dans [JC05]. M. Ciet et M. Joye soulignent que cet algorithme est résistant à l'attaque de D. Wagner. Par ailleurs, ils mettent en évidence l'importance de protéger spécialement les valeurs calculées à l'étape 1 contre les fautes. Nous montrerons à *FDTC 2008* que malgré cette précaution, ce schéma reste vulnérable face à des perturbations transitoires (voir Section 11.3.2.3).

11.2.4 Contre-mesures combinées SPA et DFA

Protéger les implantations d'algorithmes cryptographiques contre les perturbations n'est pas suffisant pour garantir leur sécurité physique. En effet, l'exécution de ces algorithmes sur un composant dénué de contre-mesures peut laisser fuir des signaux compromettants. Ces signaux pourraient être ensuite analysés malicieusement par un attaquant voulant extraire des informations secrètes. Le cas des attaques par analyse élémentaire de l'exécution d'une exponentiation modulaire en est un exemple concret [KJJ99] (*cf.* Section 1.3.4.3). En effet, pour les implantations classiques de l'exponentiation modulaire, le flot d'exécution dépend directement de l'exposant manipulé. Par conséquent, si un attaquant est capable de déterminer le flot d'exécution d'une exponentiation, il pourra en déduire directement la valeur de l'exposant. Afin de combler cette vulnérabilité potentielle, J.-S. Coron a proposé l'algorithme *Square & Multiply Always* (*cf.* Section 1.3.4.3). Dans cet algorithme, le flot d'exécution est toujours le même, quelle que soit la valeur de l'exposant. Malheureusement, M. Joye et S.-M. Yen ont prouvé que cette variante introduisait une nouvelle vulnérabilité face aux perturbations [YJ00] : les *safe-errors* . Le principe consiste a exploiter des erreurs sur les multiplications modulaires n'ayant aucune conséquence sur le résultat final du calcul. En effet, dans la variante *Square & Multiply Always*, des multiplications sont ajoutées pour *"équilibrer"* l'exécution, mais ces multiplications transparentes du point de vue du résultat de l'exponentiation modulaire. Ainsi, sachant que c'est l'une de ces multiplications qui a été perturbée et, connaissant l'instant où la faute est provoquée et, l'attaquant peut déduire la valeur du bit d'exposant traité à cet instant. En répétant cette attaque sur plusieurs exécutions de l'exponentiation, il peut alors déterminer entièrement l'exposant privé.

Par conséquent, il est devenu nécessaire d'imaginer des contre-mesures contre les attaques par analyse élémentaire n'introduisant pas d'éventuelles vulnérabilités exploitables par des perturbations. Dans ce cadre, nous avons distingué deux méthodes visant à apporter une sécurité combinée.

11.2.4.1 La vérification de cohérence

Le principe de vérification de cohérence s'inspire des propriétés de l'exponentiation binaire de Montgomery (*cf.* Algorithme 6) qui permet de calculer simultanément $(m^d \bmod p, m^{d+1} \bmod p)$. L'exponentiation de Montgomery est alors utilisée pour calculer chacune des demi exponentiations, puis le résultat est recombiné pour calculer la signature S et la valeur de vérification. La signature est alors comparée à la valeur de vérification via une *vérification de cohérence* permettant de s'assurer qu'aucune erreur ne s'est glissée durant l'exécution. Ce principe a d'abord été introduit par C. Giraud à *FDTC 2005* [Gir05b, Gir05c]. L'algorithme dérivé est rappelé ci-dessous.

Soit k un entier aléatoire de taille 32 bits.

1. Calcul des paramètres CRT par un algorithme d'exponentiation qui résiste aux analyses élémentaires et aux *safe-errors* :

$$\begin{aligned} s_p &\equiv m^{d_p} \bmod (k \cdot p),\ s'_p \equiv m^{d_p - 1} \bmod (k \cdot p) \\ s_q &\equiv m^{d_q} \bmod (k \cdot q),\ s'_q \equiv m^{d_p - 1} \bmod (k \cdot q) \end{aligned} \tag{11.22}$$

2. Recombinaison CRT par l'algorithme modifié $CRT_{blinded}$:

$$CRT_{blinded}(s_p, s_q) = s_p \bmod (p \cdot q) + p \cdot (i_p \cdot (s_q - s_p) \bmod (k \cdot p)) \tag{11.23}$$

On obtient alors S et S' tels que :

$$\begin{aligned} S &= CRT_{blinded}(s_p, s_q) \\ S' &= CRT_{blinded}(s'_p, s'_q) \end{aligned} \tag{11.24}$$

3. Enfin, on vérifie que le calcul s'est correctement déroulé :
   (a) Si $S \equiv (m \cdot S') \bmod N$ & (paramètres, p, q, i_p,non perturbés), renvoyer S,
   (b) Sinon, renvoyer `Erreur : Faute détectée`.

Une étude avancée de la sécurité de l'algorithme, vis-à-vis des perturbations et de l'analyse de signaux compromettants, est proposée dans [Gir07]. Enfin, A. Bosher, R. Naciri et E. Prouff ont proposé dans [BNP07] une variante de cet algorithme. Cependant, R. Naciri montra, quelques semaines plus tard, que l'implantation de leur algorithme pouvait être cassée, en une seule exécution, en combinant une attaque par analyse élémentaire avec une analyse différentielle de perturbation.

11.2.4.2 Les chaînes d'additions auto-immunes

En s'inspirant de la méthode précédente, M. Rivain proposa d'aller plus loin dans le principe de vérification de cohérence et proposa, à *CT-RSA 2009*, une contre-mesure combinée basée sur les *chaînes d'addition* [Riv09]. Le principe général consiste à calculer simultanément $(A, B) = (m^{d_p} \bmod p, m^{\varphi(p)-d_p} \bmod p)$ et vérifier que $A \cdot B \stackrel{?}{\equiv} 1 \bmod p$. Afin de calculer simultanément les deux puissances du message, M. Rivain a proposé une méthode pour construire une *chaîne d'addition double*. En d'autres termes, construire une suite de taille $n \in \mathbb{N}$ de telle sorte que, comme $d_p \leqslant \varphi(p) - d_p$) :

1. $(a_0, b_0) = (d_p, 2\varphi(p) - d_p)$
2. $(a_n, b_n) = (0, 1)$
3. $\forall i, 1 \leqslant i \leqslant n-1$, (a_{i+1}, b_{i+1}) est obtenu à partir de (a_i, b_i) en retranchant 1, en divisant par deux et/ou en retranchant à un élément la valeur de l'autre.

On note alors w_p la séquence binaire codant la construction de cette chaîne. Cette séquence est de taille n et sert alors à coder la double exponentiation modulaire calculée par la fonction *DoubleExpMod*. Soient $c_p = m^{2\varphi(p)-d_p} \bmod p$ (resp. $c_q = m^{2\varphi(q)-d_q} \bmod q$) les valeurs de vérification, alors, on peut décrire la contre-mesure de M. Rivain appliquée au RSA-CRT de la manière suivante :

1. Construire $w_p \leftarrow$ AddChaineDouble$(d_p, 2\varphi(p) - d_p)$,
2. Calculer $(S_p, c_p) \leftarrow$ DoubleExpMod$(\dot{m}, w_p, p)$,
3. Construire $w_q \leftarrow$ AddChaineDouble$(d_q, 2\varphi(q) - d_q)$,
4. Calculer $(S_q, c_q) \leftarrow$ DoubleExpMod$(\dot{m}, w_q, q)$,
5. Recombinaison CRT : $S = CRT(S_p, S_q)$,
6. Vérifier que $S \cdot c_p \equiv 1 \bmod p$ et $S \cdot c_q \equiv 1 \bmod q$ avant de retourner S, sinon retourner l'erreur

En général, réaliser une exponentiation modulaire basée sur des chaînes d'additions n'est pas très intéressant car il est difficile de trouver des chaînes de taille minimale. Cependant, M. Rivain propose dans [Riv09] une méthode heuristique garantissant un coût moyen pour l'exponentiation modulaire double, seulement 10% supérieur à celui d'une exponentiation modulaire

classique. Par ailleurs, une étude détaillée de la sécurité de cette contre-mesure est aussi proposée dans [Riv09], permettant de montrer sa résistance face aux différentes attaques par perturbation connues. Enfin, en utilisant la méthode de branchement conditionnel protégée contre les perturbations [DGRS09] pour réaliser la comparaison finale (*cf.* Étape 6), cet algorithme offre une protection au *second ordre*, c'est-à-dire contre les attaques en *double-fautes* [KQ07].

11.3 Perturbation d'une implantation protégée de RSA-CRT

11.3.1 Contexte de l'attaque

Dans sa thèse soutenue en 2007, C. Giraud dressait un état de l'art des différentes contre-mesures qui ont été proposées pour protéger le crypto-système RSA en mode CRT. Parmi ces contre-mesures, seulement deux d'entre elles ne présentaient pas de vulnérabilités connues face aux perturbations : l'algorithme Ciet & Joye [JC05] et la contre-mesure combinée SPA/DFA de Giraud. Dans le cadre de la thèse, nous nous sommes plus particulièrement intéressé à l'étude de la sécurité de l'algorithme Ciet & Joye vis-à-vis des fautes. Dans cette optique, nous avons imaginé l'attaque par perturbation décrite dans ce chapitre.

Modèle de perturbation. Le modèle de perturbation que nous avons choisi s'inspire de celui utilisé dans l'attaque de D. Wagner contre l'algorithme *BOS* (*cf.* Section 11.2.3.2 et [Wag04]). Dans ce modèle, nous considérerons les perturbations transitoires d'un octet de S_{p^*}, résultat de l'exponentiation modulo p^*. Nous avons choisi de considérer de telles perturbations sur S_{p^*} car, selon [JC05], il n'est pas nécessaire de protéger cette variable contre les perturbations. Sans recommandations particulières, cette variable pourra donc être perturbée facilement sur la plupart des implantations de l'algorithme Ciet & Joye. D'autre part, la perturbation aléatoire d'un octet est un modèle réaliste ayant déjà mené vers des applications concrètes, notamment dans le domaine des cartes à puce [Wag04, Gir05a].

D'un point de vue mathématique, cette perturbation aléatoire d'un octet peut être représentée de la manière suivante. Soient, $i \in [\![0; \frac{(n/2)+\kappa}{8} - 1]\!]$ et $\varepsilon \in [\![-128; 127]\!] \setminus \{0\}$, la valeur perturbée $\hat{S}_{p^*}$ peut s'écrire :

$$\hat{S}_{p^*} = S_{p^*} + \varepsilon \cdot 2^{8i} \tag{11.25}$$

Afin d'avoir une vision plus concrète de l'effet de la perturbation, celle-ci peut encore être schématisée comme suit.

Enfin, dans notre modèle, ni la position de la faute i dans la zone mémoire contenant S_{p^*}, ni

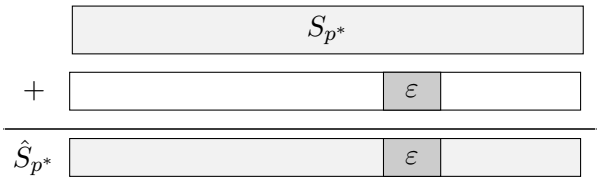


FIGURE 11.1 – Perturbation d'un octet aléatoire du résultat de l'exponentiation modulo p^*

la valeur de ε ne sont connues par l'attaquant. Dans un souci de concision, nous présenterons le principe de notre attaque en considérant seulement la perturbation de S_{p^*}. Mais on peut noter qu'une analyse similaire peut permettre d'exploiter les perturbations de S_{q^*}.

11.3.1.1 Exemple d'exécution perturbée

Maintenant que nous avons délimité le cadre de notre étude de l'algorithme Ciet & Joye vis-à-vis des perturbations, voyons comment la modification malicieuse du résultat de l'exponentiation modulo p^* infecte la signature retournée. Dans la suite, nous noterons $\hat{x}$ la valeur infectée par la faute qui est associée à une variable x.

Comme stipulé dans le modèle, la faute est injectée sur le résultat de l'exponentiation modulo p^*, par conséquent, en suivant l'exécution de l'algorithme Ciet & Joye (*cf.* Section 11.2.3.3), la première opération infectée est la recombinaison CRT :

$$\hat{S^*} = CRT(\hat{S}_{p^*}, S_{q^*}) \tag{11.26}$$

La valeur intermédiaire $\hat{S^*}$ étant infectée à son tour, la faute se transmet alors à l'une des valeurs de vérification c_1 :

$$\begin{cases} \hat{c_1} \equiv (\hat{S^*} - s_1 + 1) \bmod r_1 \\ c_2 \equiv (\hat{S^*} - s_2 + 1) \bmod r_2 \end{cases} \tag{11.27}$$

On peut toutefois noter que la valeur c_2 reste inchangée car la faute ne touche que le résultat du calcul modulo p^*. De ce fait :

$$\begin{aligned} \hat{S^*} \bmod q^* &\equiv S_{q^*} \\ &\equiv S^* \bmod q^* \end{aligned}$$

De plus, l'exposant final γ, calculé à partir des valeurs de vérification, est aussi touché par la faute :

$$\hat{\gamma} = \lfloor \frac{(r_3.\hat{c_1} + (2^l - r_3).c_2)}{2^l} \rfloor \tag{11.28}$$

Enfin, on obtient en sortie de l'algorithme Ciet & Joye, la signature erronée $\hat{S}$ suivante :

$$\hat{S} \equiv (\hat{S^*})^{\hat{\gamma}} \bmod N \tag{11.29}$$

Ce schéma de propagation de l'erreur n'est pas surprenant compte tenu du principe sur lequel repose l'algorithme Ciet & Joye. En effet, cet algorithme est basé sur le calcul pathogène qui consiste à détecter l'erreur et l'utiliser afin de rendre la signature erronée inexploitable par le calcul du *pgcd*. Ce principe est bien illustré dans le calcul précédent. En effet, bien que l'erreur soit injectée sur le résultat de l'exponentiation modulo p^*, ce qui pourrait la rendre potentiellement exploitable, l'erreur influence ensuite le calcul de l'exposant γ. C'est cet exposant qui est alors utilisé pour masquer la signature avant qu'elle ne soit retournée. À première vue, cette étape masque complètement le déséquilibre engendré par la perturbation initiale, ce qui rend la signature erronée inexploitable par les méthodes connues. Dans l'analyse suivante, nous allons montrer que, pour certaines erreurs, cette contre-mesure n'est pas suffisante pour empêcher l'extraction de la clé privée. Puis, à partir du modèle de perturbation, nous estimerons la probabilité de produire de telles perturbations.

11.3.2 Analyse différentielle de la perturbation

11.3.2.1 Analyse différentielle

Pour commencer l'analyse, nous allons comparer le résultat d'une signature erronée $\hat{S}$, suivant le modèle, à la signature correcte S obtenue à partir d'un même message m. On peut d'abord

remarquer que, bien que la faute vise à perturber la valeur de S_{p^*} seulement, l'exponentiation finale à la puissance $\hat{\gamma}$ engendre que :

$$\begin{cases} \hat{S} \not\equiv S \bmod p \\ \hat{S} \not\equiv S \bmod q \end{cases} \tag{11.30}$$

Ce qui, comme nous le soulignions précédemment, rend toute exploitation par le *pgcd* impossible. Mais, regardons de plus près l'expression de $\hat{S}$. En effet, $\hat{S} \equiv (\hat{S^*})^{\hat{\gamma}} \bmod N$ où $\hat{S^*}$ est le résultat de la recombinaison CRT d'un résultat correct S_{q^*} et de la valeur perturbée $\hat{S}_{p^*}$. En d'autres termes :

$$\begin{cases} \hat{S^*} \equiv \quad \hat{S}_{p^*} \bmod p \not\equiv \quad S_{p^*} \bmod p \\ \hat{S^*} \equiv \quad S_{q^*} \bmod q \end{cases} \tag{11.31}$$

En utilisant l'exposant public, à la manière de l'amélioration de Lenstra (*cf.* Section 11.2.1), on peut encore écrire :

$$\begin{cases} {\hat{S^*}}^e \not\equiv \dot{m} \bmod p \\ {\hat{S^*}}^e \equiv \dot{m} \bmod q \end{cases} \tag{11.32}$$

Or, d'après l'algorithme de Ciet & Joye, la signature renvoyée ici est $\hat{S} = (\hat{S^*})^{\hat{\gamma}} \bmod N$. En combinant cette information avec le résultat précédent, nous avons :

$$\begin{cases} \hat{S}^e \not\equiv \dot{m}^{\hat{\gamma}} \bmod p \\ \hat{S}^e \equiv \dot{m}^{\hat{\gamma}} \bmod q \end{cases} \tag{11.33}$$

Par conséquent, si l'attaquant peut retrouver la valeur de l'exposant $\hat{\gamma}$, il pourra exploiter la faute car $(\hat{S}^e - \dot{m}^{\hat{\gamma}})$ est un multiple de q mais pas de p. Aussi, en utilisant la fonction *pgcd*, il pourra s'attendre à obtenir $pgcd((\hat{S}^e - \dot{m}^{\hat{\gamma}}) \bmod N, N) = q$, un des deux facteurs de N. Dans ce cas, la difficulté du problème de la factorisation de N repose sur la difficulté de retrouver $\hat{\gamma}$. Évidemment, ce cas a été pris en compte par les concepteurs qui ont fait en sorte que, lorsqu'une faute est détectée, $\hat{\gamma}$ soit une valeur aléatoire de taille κ bits. Aussi, il suffirait que κ soit suffisamment grand pour qu'il ne soit pas possible de retrouver $\hat{\gamma}$ dans un temps raisonnable. Dans la partie suivante, nous allons montrer que pour certaines erreurs, la valeur de $\hat{\gamma}$ pourra être bornée dans un espace suffisamment restreint pour qu'un attaquant puisse la retrouver à moindre coût.

Comment retrouver l'exposant $\hat{\gamma}$? Pour répondre à cette question, commençons par étudier plus précisément la propagation de l'erreur dans le calcul de l'exposant $\hat{\gamma}$. En étudiant l'équation (11.20), on peut noter que γ dépend d'un aléa r_3 de taille l bits, mais aussi des valeurs de vérification c_1 et c_2. Justement, l'injection d'une faute suivant le modèle vient perturber le résultat – $\hat{S}_{p^*}$ – de l'une des demi-exponentiations. Par conséquent, les valeurs de vérification peuvent s'exprimer en fonction de l'erreur ε :

$$\begin{cases} \hat{c}_1 \equiv \quad (\hat{S^*} - s_1 + 1) \bmod r_1 \equiv \quad (1 + \varepsilon \cdot 2^{8i}) \bmod r_1 \\ c_2 \equiv \quad (\hat{S^*} - s_2 + 1) \bmod r_2 \equiv \qquad\qquad 1 \bmod r_2 \end{cases} \tag{11.34}$$

L'équation précédente nous montre que, comme on pouvait s'y attendre, seul c_1 détecte une faute. D'autre part, cette valeur de vérification est le reste de la division euclidienne de l'erreur,

incrémentée de 1, par l'aléa r_1. Aussi, afin de contourner l'influence de l'aléa, et par la même occasion, limiter l'espace de recherche pour $\hat{c}_1$, nous nous intéresserons au cas où l'erreur est comprise entre :

$$0 \leqslant |1 + \varepsilon \cdot 2^{8i}| \leqslant \frac{r_1 - 1}{2} \tag{11.35}$$

Dans ce cas, aucune réduction modulaire n'intervient dans le calcul de $\hat{c}_1$. Cette valeur de vérification dépend alors directement de l'erreur résiduelle ε. Par cette hypothèse supplémentaire nous pouvons limiter considérablement l'espace des valeurs possibles pour $\hat{c}_1$. Nous étudierons la probabilité que cette hypothèse soit satisfaite, suivant notre modèle de faute, à la section 11.3.2.2.

Dans le cas où l'équation (11.35) est satisfaite, nous avons $\hat{c}_1 = 1 + \varepsilon \cdot 2^{8i}$ et, comme $c_2 = 1$, la valeur de $\hat{\gamma}$ peut encore être exprimée :

$$\hat{\gamma} = \lfloor \frac{(r_3 \cdot \hat{c}_1 + (2^l - r_3) \cdot c_2)}{2^l} \rfloor = 1 + \lfloor \frac{r_3 \cdot \varepsilon \cdot 2^{8i}}{2^l} \rfloor \tag{11.36}$$

Le numérateur correspond à un produit entre l'aléa r_3 et l'erreur sur un octet ε ayant subi un décalage vers la gauche de $8i$ bits. Afin d'avoir une idée plus précise de la construction de l'exposant $\hat{\gamma}$, sous les hypothèses que nous avons adoptées, la figure 11.2 illustre la distribution des bits du produit dans le dénominateur de $\hat{\gamma}$. Pour finir de calculer $\hat{\gamma}$, on applique à ce produit

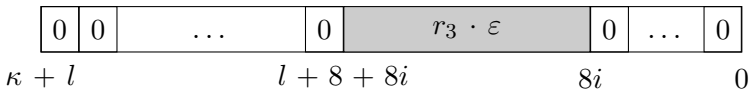


FIGURE 11.2 – Distribution des bits d'erreur dans le dénominateur de $\hat{\gamma}$

une division entière par 2^l de façon à normaliser la taille de $\hat{\gamma}$. Cette opération peut se traduire par un décalage de l bits vers la droite de la valeur calculée précédemment. Si l'on découpe le calcul de $\hat{\gamma}$ différemment, celui-ci peut être interprêté comme un produit entre l'erreur introduite ε et l'aléa r_3 sur lequel on applique un décalage de $(8i - l)$ bits. Et, suivant la position de l'erreur i, ce décalage final n'aura pas la même influence sur la distribution des bits de $\hat{\gamma}$:

1. $(l - \kappa) \geqslant 0$. Dans ce cas $\forall i \in [\![0; \frac{\kappa}{8} - 1]\!]$, alors $(8i - l) < 0$. Cela signifie que la partie du dénominateur $r_3 \cdot \varepsilon$, qui peut être considérée comme un aléa de taille $l + 8$ bits, subit un décalage vers la droite. Par conséquent, la valeur de $\hat{\gamma}$ sera composée d'un aléa de taille $(l + 8) + (8i - l) = 8(i + 1)$ bits localisé sur les bits de poids faible. Ce cas est illustré par la figure 11.3.

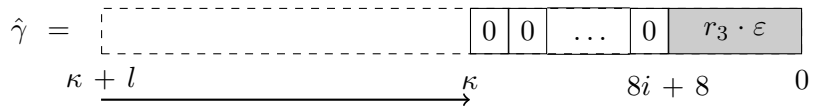


FIGURE 11.3 – Distribution des bits de $\hat{\gamma}$ pour un décalage $l > 8i$

2. L'autre cas est $(l - \kappa) < 0$. Alors, si $(8i - l) < 0$, la distribution des bits $\hat{\gamma}$ sera similaire à ce qui est décrit dans le cas précédent, à savoir un aléa localisé sur les poids faibles. Autrement, lorsque $(8i - l) > 0$, $\hat{\gamma}$ reste une valeur aléatoire de taille $8(i + 1)$ bits, mais localisée sur les bits de poids fort du vecteur comme illustré par la figure 11.4. On pourra d'ailleurs noter ici que la valeur de $\hat{\gamma}$ est toujours impaire.

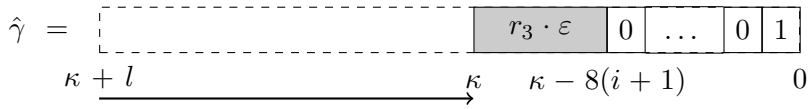


FIGURE 11.4 – Distribution des bits de $\hat{\gamma}$ pour un décalage $l < 8i$ et $l < \kappa$

Par conséquent la forme de $\hat{\gamma}$ dépend fortement de la position de la perturbation et, dans une moindre mesure, des paramètres définissant les longueurs d'aléa dans l'implantation de l'algorithme Ciet & Joye. Plus précisément, l'indice de position i influence la taille d'aléa contenue dans la valeur de $\hat{\gamma}$. Évidemment, la valeur de l'exposant $\hat{\gamma}$ reste aléatoire mais, si i est suffisamment petit (*i.e.* $i \leqslant 5$), l'exposant pourra être retrouvé par recherche exhaustive.

Algorithme d'attaque L'attaquant commence par définir une taille limite B_f pour la recherche exhaustive puis, pour une signature erronée $\hat{S}$ d'un message m obtenue suivant le modèle de faute. Il teste alors toutes les valeurs candidates $1 \leqslant \hat{\gamma}' < 2^{B_f}$ jusqu'à ce qu'il réussisse à factoriser le module public N en calculant $pgcd((\hat{S}^e - \dot{m}^{\hat{\gamma}'}) \bmod N, N)$. Si aucune valeur de $\hat{\gamma}'$ ne permet de retrouver un facteur de N, cela peut signifier que l'équation (11.35) n'est pas satisfaite. Il faut alors obtenir une nouvelle signature erronée et recommencer l'analyse jusqu'à obtenir un facteur de N. L'analyse précédente est résumée par l'algorithme 11. Enfin, notre étude est complétée, dans le paragraphe suivant, par une estimation théorique du nombre de signatures erronées à collecter pour espérer factoriser le module.

Algorithme 11 Attaque par perturbation de l'implantation Ciet & Joye de RSA-CRT

ENTRÉES : m, N, κ, l, $\hat{S}$, limite de recherche exhaustiveB_f
SORTIE : un facteur de N

{Quel que soit le signe de $(l - \kappa)$, le cas LSB est traité}
Pour $\hat{\gamma}' = 1$ **à** $2^{B_f} - 1$ **faire**
  $p' \leftarrow pgcd((\hat{S}^e - \dot{m}^{\hat{\gamma}'}) \bmod N, N)$
  Si $p' \neq 1$ **alors**
    Retourner p'
  Fin Si
Fin Pour
{Si γ reste inconnu et $l < \kappa$, on cherche sur les MSB}
Si $l < \kappa$ **alors**
  Pour $i = 1$ **à** $2^{B_f} - 1$ **faire**
    {Les bits sont décalés vers les poids forts}
    $\hat{\gamma}' \leftarrow (i \ll (\kappa - B_f - 1)) + 1$
    $p' \leftarrow pgcd((\hat{S}^e - \dot{m}^{\hat{\gamma}'}) \bmod N, N)$
    Si $p' \neq 1$ **alors**
      Retourner p'
    Fin Si
  Fin Pour
Fin Si
{Ici, l'algorithme termine son exécution sans factoriser N}
Retourner -1

11.3.2.2 Performances

Afin d'évaluer le nombre de signatures à obtenir pour espérer factoriser N par la méthode précédente, nous avons supposé que la valeur de l'erreur ε ainsi que sa position i sont uniformément distribuées dans leur espace respectif. En pratique, si l'attaquant peut influencer la position du registre de S_{p^*} perturbé, il pourra réduire significativement le nombre de signatures erronées à collecter. D'après l'analyse précédente, nous avons fait deux hypothèses importantes sur la perturbation d'une signature RSA pour qu'il soit possible de factoriser N à moindre coût :

1. Aucune réduction modulaire n'est exécutée lors du calcul de la valeur de vérification c_1. Cette condition peut être traduite l'équation (11.35). Sachant que r_1 est un aléa de taille κ et que $|1 + \varepsilon \cdot 2^{8i}| \leqslant 2^{8i+7}$, la probabilité que la condition précédente soit satisfaite peut être approximée par :

$$\Pr\left[|1 + \varepsilon \cdot 2^{8i}| < \frac{r_1 - 1}{2}\right] \approx \Pr\left[8(i+1) < \kappa\right] \tag{11.37}$$

   Ainsi, l'évaluation de la probabilité précédente ne dépend que de la position i de la perturbation d'un octet de S_{p^*}. En supposant que i est uniformément distribué dans $[\![0; \frac{(n/2)+\kappa}{8} - 1]\!]$, on obtient donc :

$$\Pr[8(i+1) < \kappa] = \frac{2 \cdot (\kappa - 8)}{n + 2\kappa} \tag{11.38}$$

2. L'autre hypothèse concerne la possibilité de retrouver $\hat{\gamma}$ par recherche exhaustive. Dans l'analyse précédente, nous avons montré que, quelle que soit la localisation de la partie significative de $\hat{\gamma}$, la condition précédente implique que la partie significative est de taille $8(i+1)$ bits. Afin de retrouver ces bits, l'attaquant fixe une taille limite B_f pour la recherche, de telle sorte que cette taille soit adapté à la puissance de calcul dont l'attaquant dispose. Par conséquent, pour que l'attaquant puisse retrouver $\hat{\gamma}$ par une recherche exhaustive :

$$8(i+1) \leqslant B_f \tag{11.39}$$

   Comme précédemment, sachant que l'on considère les perturbations aléatoires sur un octet de S_{p^*}, alors la position de l'octet perturbé i appartient à $[\![0; \frac{(n/2)+\kappa}{8} - 1]\!]$. Si i est uniformément distribué dans cet espace, alors la probabilité pour $\hat{\gamma}$ puisse être retrouvé par recherche exhaustive peut s'exprimer :

$$\Pr[8(i+1) < B_f] = \frac{2 \cdot (B_f - 8)}{n + 2\kappa} \tag{11.40}$$

En regardant les deux expressions précédentes, on remarque que le calcul de la probabilité de chacun des évènements revient à calculer la probabilité que la valeur de la perturbation soit bornée respectivement, par la taille de l'aléa r_1, fixée par les développeurs, et par la taille de la recherche exhaustive B_f, fixée par l'attaquant. Or, l'attaque ne peut aboutir que si les deux évènements précédents se réalisent simultanément, aussi, la probabilité de réussir l'analyse pour une signature erronée $\hat{S}$ obtenue suivant notre modèle de faute est :

$$\Pr\left[\hat{S} \text{ soit exploitable}\right] = \Pr\left[8(i+1) < \kappa \text{ et } 8(i+1) < B_f\right] \tag{11.41}$$

$$= \Pr\left[8(i+1) < min(\kappa, B_f)\right] \tag{11.42}$$

Par conséquent, pour un RSA-CRT de taille $n = 1024$ bits, implanté avec la méthode Ciet & Joye [JC05] pour un paramètre $\kappa = 80$ bits et $B_f = 40$ bits, la probabilité qu'une signature erronée permette de factoriser N est de $5,8\%$. De plus, la probabilité d'avoir une signature exploitable dépend du minimum entre le paramètre d'aléa κ et de la capacité de calcul de l'attaquant B_f, rallonger la taille des aléas n'est pas un moyen de se protéger contre notre attaque. Par ailleurs, diminuer la taille d'aléa κ, ce qui minimise la probabilité d'avoir une signature exploitable, n'est pas une solution non plus. En effet, la taille de $\hat{\gamma}$ pourrait ne plus être suffisante pour dissuader un attaquant de retrouver sa valeur par une recherche exhaustive, sans considérer quelque modèle de perturbation.

Si l'on regarde maintenant le nombre moyen de signatures erronées à obtenir pour espérer factoriser N, nous avons évalué que 13 signatures sont suffisantes pour espérer factoriser N avec une probabilité légèrement supérieure à 50%. Avec 83 signatures, la probabilité de réussite est de l'ordre de 99%. Ces estimations ont été confirmées par notre implantation GMP de la méthode sur un PC démontrant un peu plus le caractère pratique de notre attaque.

11.3.2.3 Proposition de contre-mesures

L'attaque que nous avons proposée dans cette partie utilise une faille dans la construction de l'exposant de masquage γ. Cette faille peut même être imputée à la manière dont sont calculés les valeurs de vérification c_1 et c_2. Le principe de la première contre-mesure que nous proposons consiste à forcer l'exécution d'une réduction modulaire pendant leur calcul respectif, lorsqu'une faute est détectée. Ceci peut être fait, à moindre coût, en utilisant un aléa α de taille κ bits de telle sorte que $\alpha > r_1$ et $\alpha > r_2$:

$$\begin{cases} c_1 \equiv (\alpha \cdot (S^* - s_1) + 1) \mod r_1 \\ c_2 \equiv (\alpha \cdot (S^* - s_2) + 1) \mod r_2 \end{cases} \tag{11.43}$$

De cette manière, lorsqu'aucune erreur n'est détectée $c_1 = c_2 = 1$. Dans le cas contraire, la condition sur α provoque automatiquement un masquage de l'erreur par une multiplication par un aléa et une réduction modulaire.

L'autre proposition de contre-mesure consiste à remplacer systématiquement l'opération finale d'exponentiation modulaire à la puissance γ par la variante proposée par M. Ciet et M. Joye dans le même article [JC05]. En effet, celle-ci consiste à tirer un aléa r de taille $(n+\kappa)$ bits et retourner en guise de signature :

$$S = (\gamma \cdot S^* \oplus (\gamma - 1) \cdot r) \mod N \tag{11.44}$$

11.4 Conclusion

Ce chapitre met en évidence toute la difficulté d'élaborer des contre-mesures efficaces pour protéger les implantations d'algorithmes cryptographiques. Dans cette optique, nous avons détaillé l'évolution parallèle des attaques et des contre-mesures élaborées pour protéger le RSA-CRT, largement déployé sur les systèmes embarqués. Plus particulièrement, nous avons proposé une méthode pour attaquer une implantation protégée de l'algorithme de signature RSA en mode CRT. Sous un modèle de faute réaliste, nous avons montré qu'il est possible de retrouver un exposant privé RSA de 1024 bits à partir de 83 signatures perturbées. Aussi, nous suggérons fortement d'utiliser la variante proposée dans [JC05] pour remplacer l'opération de masquage

finale de l'algorithme Ciet & Joye. Enfin, nous proposons un tableau récapitulatif donnant un aperçu rapide des différentes contre-mesures proposées pour implanter le RSA en mode CRT ainsi que l'étude de leur résistance face à différents modèles de perturbation (*cf.* Tableau 11.1).

La principale information que l'on peut tirer de ce tableau est que, depuis la découverte des attaques par perturbations à la fin des années 90 [BDL97], près d'une douzaine de proposition de contre-mesures se sont succédées pour protéger la signature RSA en mode CRT. Cette moyenne de près d'une contre-mesure par an met en évidence à la fois l'évolution rapide des méthodes d'attaque mais aussi la difficulté d'élaborer des contre-mesures durables. La première difficulté que nous avons recensée consiste à déterminer précisément l'ensemble des menaces contre lesquelles on souhaite se prémunir. Cette étude préliminaire nécessaire doit prendre en compte à la fois les moyens que peut mettre en œuvre l'attaquant ainsi que la valeur de ce que l'on souhaite protéger. En effet, il n'est pas intéressant d'utiliser des contre-mesures nécessitant des moyens dépassant largement la valeur du secret à protéger. Par ailleurs, comme le montre l'exemple du RSA-CRT, il est difficile d'anticiper l'évolution des méthodes d'attaques. Aussi, il est intéressant d'imaginer des contre-mesures pouvant être facilement modifiées pour prendre en compte les nouvelles failles et les colmater. Ceci est d'autant plus important dans les systèmes embarqués, comme les cartes à puce, où il est préférable de diffuser des mises à jour logicielles que de devoir changer le matériel.

Enfin, cette succession d'attaques et de contre-mesures montre aussi le dynamisme de la communauté pour prendre de court les attaquants potentiels à la rechercher d'éventuelles failles. En effet, le RSA en mode CRT est largement déployé pour assurer la sécurité de transactions bancaires notamment. De plus, la presse internationale a relayé ces dernières années les résultats spectaculaires d'attaques contre des systèmes de paiement électronique utilisant soit des cartes à puce, soit des serveurs [36]. Aussi, il relève d'un enjeu économique important de pouvoir proposer des solutions sécurisées embarquant les meilleures contre-mesures possibles.

36. A notre connaissance la dernière attaque en date à avoir été relayée en Février 2010 est celle de R. Anderson permettant de réaliser des transactions bancaires sans connaître le code PIN [MDAB10]. Cette attaque publiée sous le nom *"Chip and PIN is Broken"* a été relayée notamment par ZDNet ou *The Telegraph*. On peut signaler encore l'exploitation des vulnérabilités induites par le mécanisme de branchement prédictif des micro processeurs modernes proposé par J.-P. Seifert *"On the Power of Simple Branch Prediction Analysis"* [AcKS07] et relayée par le quotidien français *Le Monde* sous le titre plus que vendeur : *"Les puces ne garantissent pas la sécurité des échanges en ligne"*.

Nom de la contre-mesure	Principe de base	Avantage(s) ou Inconvénient(s) majeurs	Meilleure attaque connue	Performances
Doublement de la signature	Comparer le résultat de 2 exécutions de la même signature	− Doublement du temps d'exécution, doublement de la surface pour une implantation parallèle.		
Vérification de la signature	Utiliser l'exposant public e	+ Plus intéressant si e est *"petit"*, − Nécessite de connaître e.		
Astuce de Shamir [Sha97]	Extension aléatoire des modules	− Nécessite de connaître d	Attaque de *Aumüller et al.* [ABF+02]	1 faute avant la recombinaison CRT
Algorithme de Yen *et al.* (1) [YKLM01]	Calcul pathogène	− Nécessite de connaître d, coût des précalculs.	Attaque de Yen *et al.* [YKM06]	1 faute sur k_p (ou bien k_q)
Algorithme de Yen *et al.* (2) [YKLM01]	Calcul pathogène	− Nécessite de connaître d, coût des précalculs.	Attaque de Yen *et al.* [YKM06]	1 faute sur k_p (ou bien k_q)
Algorithme de *BOS* (1) [BOS03]	Calcul pathogène & extension aléatoire de module	− Coût des précalculs (inversions modulaires)	Attaque de Wagner [Wag04]	25 fautes sur un octet de m
Algorithme Ciet & Joye [JC05]	Calcul pathogène & extension aléatoire de module	+ Efficacité de l'implantation, − Utiliser la variante seulement.	Notre attaque [BCG08a]	83 fautes sur S_p (ou bien S_q)
Contre-mesure unifiée SPA et DFA de Giraud [Gir05b]	Extension de module & Vérification de cohérence	+ Contre-mesure unifiée, − 2 multiplications/bit d'exposant	Néant	
Algorithme de *BOS* (2) [BO06]	*Idem BOS* (1) & randomisation des vérifications	− Coût des précalculs (inversions modulaires)	Néant	
Contre-mesure unifiée SPA et , DFA de Boscher *et al.* [BNP07]	*Square & Multiply Always* & Vérification de cohérence	+ Contre-mesure unifiée − Peut-être attaquée ...	Attaque de Naciri	1 faute sur S_p (ou bien S_q)
Algorithme de Vigilant [Vig08]	Extension des modules par des carrés d'aléas	+/− Définir la taille de l'aléa offre un compromis entre détection d'erreurs et performances	Néant [37]	
Chaînes d'addition auto-immunes [Riv09]	Exponentiations par chaînes d'additions & Vérification de cohérence	+ 1.65 multiplications/bit d'exposant	Néant	

TABLE 11.1 – Récapitulatif des méthodes de protection des signatures RSA-CRT contre les perturbations

Cinquième partie

Conclusion & Perspectives

Ce mémoire de thèse présente la réflexion que nous avons menée autour des attaques par perturbation. Bien que ce domaine de recherche soit très spécifique, son étude complète requiert néanmoins des compétences dans des disciplines variées comme la micro-électronique, les mathématiques et l'informatique. Outre sa pluridisciplinarité, le domaine des attaques par perturbation répond à un réel besoin industriel. Dans l'industrie de la cartes à puce, ces attaques sont considérées comme une menace réelle, et protéger les cartes contre les perturbations relève d'un enjeu économique important.

Dans l'approche globale de ce sujet, nous nous sommes mis dans la peau d'un attaquant pouvant perturber des composants cryptographiques. Dans un premier temps nous avons étudié la sécurité des implantations du RSA face à la perturbation de ses éléments publics. Nous avons ensuite étendu ces attaques à d'autres implantations d'algorithmes de signatures électroniques. Ensuite, nous avons essayé d'exploiter les perturbations d'implantations d'algorithmes de chiffrement à flot. Sous un modèle de faute classiquement considéré, nous avons montré que les perturbations des implantations de Grain-128 peuvent laisser fuir de l'information critique. Puis, nous avons considéré un modèle de perturbation plus original, la substitution d'une opération, pour publier la meilleure attaque physique connue sur les implantations de Rabbit. Au delà de ce résultat, nous espérons que ce point de vue sera utilisé pour étendre l'étude de la sécurité physique d'implantations d'algorithme de chiffrement à flot.

Certes, il est intéressant d'imaginer les attaques les plus efficaces possibles. Mais n'oublions que ces attaques doivent surtout mettre en évidence les parties critiques d'implantations. Ainsi, il sera plus facile de proposer des contre-mesures adaptées. Typiquement, notre attaque contre une implantation protégée de la signature RSA, en mode CRT, a montré que l'opération de masquage final est inappropriée car certaines perturbations peuvent la rendre faible. Cependant, cette vulnérabilité pourra être fixée en utilisant une autre opération, pourtant moins gourmande. Cet exemple met en évidence une propriété essentielle d'un bonne contre-mesure. Celle-ci doit, non seulement protéger efficacement contre toutes les attaques connues, mais surtout, elle doit être facilement modifiable pour rester résistante à de nouvelles méthodes d'analyse. Pour cette raison l'utilisation de contre-mesures logicielles est particulièrement adaptée. En effet, il semble plus facile de fixer des failles de sécurité en proposant des mises à jour qu'en rappelant une série de composants ou de cartes.

L'exploitation d'éléments non-critiques d'algorithmes cryptographiques, comme des clés publiques, soulève une autre question : dans quelle mesure un processus peut-il avoir confiance en sa propre exécution ? Dans ce cas extrême modélisant un attaquant ayant tous les droits, il semble difficile de trouver des méthodes pouvant vérifier l'intégrité d'une exécution tout en garantissant l'intégrité de cette méthode elle-même. Nous pensons qu'une piste de recherche intéressante serait d'étudier la possibilité de construire des crypto-systèmes pouvant être modifiés sans révéler d'information sur le secret. Ce principe a été partiellement initié par le principe du calcul pathogène mais, pourrait être intégré dans la construction même d'un crypto-système pour être plus efficace.

Pour conclure cette thèse, je tiens à dire que l'étude des attaques par perturbation m'a permis de découvrir les tenants et les aboutissants de la cryptographie appliquée. J'espère aussi, par l'étude de modèles de perturbations originaux, avoir contribué modestement à l'histoire des attaques par perturbation. Une histoire dont le courant pourrait être lui-même perturbé ...

Index

Bibliographie

[ABF+02] C. Aumüler, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attack on RSA with CRT : Concrete Results and Practical Countermeasures. In B.S. Kaliski Jr., Ç.K. Koç, and C. Parr, editors, *Cryptographic Hardware and Embedded Systems (CHES 2002)*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.

[AcKS07] O. Aciiçmez, Ç.K. Koç, and J.-P. Seifert. On the Power of Simple Branch Prediction Analysis. In *ACM Symposium on Information and Computer Communications Security (ASIACCS 2007)*, pages 312–320. ACM Press, 2007.

[Adl79] L.M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Found. Comp. Sci. Symp. (FOCS 1979)*, pages 55–60. IEEE, 1979.

[AGS07] O. Aciiçmez, S. Gueron, and J.-P. Seifert. New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures. In S. D. Galbraith, editor, *IMA Int. Conf.*, volume 4887 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2007.

[Ajt98] M. Ajtai. The Shortest Vector Problem inL_2 is NP-Hard for Randomized Reductions. In *STOC*, pages 10–19. ACM Press, 1998.

[AK96] R. Anderson and M. Kuhn. Tamper Resistance - A cautionary Note. In *USENIX Workshop on Electronic Commerce*, pages 1–11, 1996.

[AM06] F. Armknecht and W. Meier. Fault Attacks on Combiners with Memory. In B. Preneel and S.E. Tavares, editors, *Selected Areas in Cryptography (SAC 2005)*, volume 3897 of *Lecture Notes in Computer Science*. Springer, 2006.

[AMMA08] R.E. Atani, W. Meier, S. Mirzakuchaki, and S.E. Atani. Design and Implementation of DPA Resistive Grain-128 Stream Cipher Based on SABL Logic. *International Journal of Computers, Communications & Control*, III :293–298, 2008.

[ANS10] ANSSI. Mécanismes cryptographiques – règles et recommandations concernant le choix et le dimmensionnement des mécanismes cryptographiques de niveau de robustesse *standard*, January 2010.

[A/S03a] Cryptico A/S. Algebraic analysis of Rabbit. White paper, 2003.

[A/S03b] Cryptico A/S. Analysis of the key setup function in Rabbit. White paper, 2003.

[A/S03c] Cryptico A/S. Hamming weights of the g-function. White paper, 2003.

[A/S03d] Cryptico A/S. Periodic properties of Rabbit. White paper, 2003.

[A/S03e] Cryptico A/S. Second degree approximations of the g-function. White paper, 2003.

[A/S03f] Cryptico A/S. Security analysis of the IV-setup for Rabbit. White paper, 2003.

[Aum07] J.-P. Aumasson. On a Bias of Rabbit. In *State of the Art of Stream Ciphers (SASC 2007)*, 2007.

[Bab86] L. Babai. On Lovász lattice reduction and the nearest point problem. *Combinatorica*, 6 :1–13, 1986.

[Bau08] A. Bauer. *Vers une généralisation rigoureuse des méthodes de Coppersmith pour la recherche de petites racines de polynômes*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, September 2008.

[BCC+09] A. Berzati, Cécile Canovas, G. Castagnos, B. Debraize, L. Goubin, A. Gouget, P. Paillier, and S. Salgado. Fault Analysis of Grain-128. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST 2009)*, San Francisco (USA), 2009. IEEE Computer Society.

[BCDG09] A. Berzati, C. Canovas, J.-G. Dumas, and L. Goubin. Fault Attacks on RSA Public Keys : Left-To-Right Implementations are also Vulnerable. In M. Fischlin, editor, *RSA Cryptographer's Track (CT-RSA 2009)*, volume 5473 of *Lecture Notes in Computer Science*, pages 414–428, San Francisco (USA), 2009. Springer.

[BCDGar] A. Berzati, C. Canovas-Dumas, and L. Goubin. *Fault Analysis in Cryptography*, chapter A Survey of Differential Fault Analysis against Classical RSA Implementations. Springer, 2011 (To appear).

[BCG08a] A. Berzati, C. Canovas, and L. Goubin. (In)security Against Fault Injection Attacks on CRT-RSA Implementations. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2008)*, pages 101–107, Washington DC (USA), 2008. IEEE Computer Society.

[BCG08b] A. Berzati, C. Canovas, and L. Goubin. Perturbating RSA Public Keys : an Improved Attack. In *Cryptographic Hardware and Embedded Systems (CHES 2008)*, Lecture Notes in Computer Science, Washington DC (USA), 2008. Springer-Verlag.

[BCG09] A. Berzati, C. Canovas, and L. Goubin. Differential Fault Analysis of Rabbit : Toward a Secret Key Leakage. In *10th International Conference on Cryptology in India (Indocrypt 2009)*, New Delhi (India), 2009.

[BCMCC06] E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier. Why One Should Also Secure RSA Public Key Elements. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 324–338. Springer-Verlag, 2006.

[BDG10] A. Berzati, C. Dumas, and L. Goubin. Public Key Perturbation of Randomized RSA Implementations. In *Cryptographic Hardware and Embedded Systems (CHES 2010)*, 2010.

[BDJ+96] F. Bao, R.H. Deng, A. Jeng, A.D. Narasimhalu, and T. Ngair. Another New Attack to RSA on Tamperproof Devices. 1996.

[BDJ+98] F. Bao, R.H. Deng, A. Jeng, A.D. Narasimhalu, and T. Ngair. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In M. Lomas and B. Christianson, editors, *Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 115–124. Springer-Verlag, 1998.

[BDL97] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[BDL01] D. Boneh, R.A. DeMillo, and R.J. Lipton. "On the Importance of Eliminating Errors in Cryptographic Computations". *Journal of Cryptology*, 14(2) :101–119, 2001.

[BECN+04] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. Cryptology ePrint Archive, Report 2004/100, 2004.

[BGN05] E. Biham, L. Granboulan, and P. Nguyen. Impossible Fault Analysis of RC4 and Differential Analysis of RC4. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption (FSE 2005)*, volume 3557, pages 359–367. Springer, 2005.

[BM06] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks Against AES. In Springer-Verlag, editor, *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4236 of *Lecture Notes in Computer Science*, pages 201–215. Springer-Verlag, 2006.

[BMM00] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Ellitic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology (CRYPTO 2000)*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer-Verlag, 2000.

[BNP07] A. Boscher, R. Naciri, and E. Prouff. CRT RSA Algorithm Protected Against Fault Attacks. In *Information Security Theory and Practices (WISTP 2007)*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–243. Springer-Verlag, 2007.

[BO06] J. Blömer and M. Otto. Wagner's Attack on a secure CRT-RSA Algorithm Reconsidered. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, volume 4236 of *Lecture Notes in Computer Science*, pages 13–23. Springer-Verlag, 2006.

[BOS03] J. Blömer, M. Otto, and J.-P. Seifert. A New CRT-RSA Algorithm Secure Against Bellcore Attack. In *ACM Conference on Computer and Communication Security (CCS 2003)*, pages 311–320. ACM Press, 2003.

[BR94] M. Bellare and P. Rogaway. Optimal Asymetric Encryption – How to encrypt with RSA. In A. De Santis, editor, *Advances in Cryptology – EUROCRYPT 1994, International Conference on the Theory and Application of Cryptographic Techniques*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[BR96] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures : How to Sign with RSA and Rabin. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT 1996, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, 1996.

[BS96] E. Biham and A. Shamir. The next stage of differential fault analysis : How to break completely unknown cryptosystems. Preprint, 1996.

[BS97] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology (CRYPTO 1997)*, 1997.

[BS00] A. Biryukov and A. Shamir. Cryptanalitic Time/memory/data Tradeoffs For Stream Cipher. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[BV96] D. Boneh and R. Venkatesan. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. In *Advances in Cryptology*

(CRYPTO 1996), volume 1109 of *Lecture Notes in Computer Science*, pages 129–142. Springer-Verlag, 1996.

[BVCZ05] M. Boesgaard, M. Vesterager, T. Christiensen, and E. Zenner. The stream cipher Rabbit. eStream Report 2005/024, the ECRYPT stream cipher project, 2005.

[BVP+03] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius. Rabbit : A High-Performance Stream Cipher. In T. Johansson, editor, *Fast Software Encryption (FSE 2003)*, volume 2887 of *Lecture Notes In Computer Science*, pages 307–329. Springer, 2003.

[Can09] G. Canivet. *Analyse des effets d'attaques par fautes et conception sécurisée sur plate-forme reconfigurable*. PhD thesis, Institut National Polytechnique de Grenoble - INPG, 2009.

[CJ05] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the presence of permanent and transient faults. *Designs, Codes and Cryptography*, 36(1) :33–43, 2005.

[CKPS00] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt 2000, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[cKRSP94] ç.K. Koç, R.L. Rivest, A. Shamir, and W.L. Price. High-Speed RSA Implementation, 1994.

[Cla07] C. Clavier. *De la sécurité physique des crypto-systèmes embarqués*. PhD thesis, Université de Versailles Saint-Quentin, 2007.

[CMV+09] G. Canivet, P. Maistri, F. Valette, J. Clédière, M. Renaudin, and R. Leveugle. Glitch and Laser Fault Attacks onto a Secure AES Implementation on a SRAM-Based FPGA. *Journal of Cryptology : Special Issue on Hardware and Security*, 2009.

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New-York Inc., 1993.

[Cop96] D. Coppersmith. Finding Small Roots of a Bivariate Integer Equation. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT 1996, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.

[Cop97] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, 10(4) :233–260, 1997.

[Cor99] J.-S. Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES 1999)*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.

[CP05] C. De Cannière and B. Preneel. Trivium – A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher. Technical report, 2005.

[DGRS09] E. Dottax, C. Giraud, M. Rivain, and Y. Sierra. On Second-Order Fault Analysis Resistance for CRT-RSA Implementations. In O. Markowitch, A. Bilas, J.-H. Hoepman, C.J. Mitchell, and J.-J. Quisquater, editors, *Information Security*

Theory and Practices (WISTP 2009), volume 5746 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2009.

[DH76a] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transaction on Information Theory*, 22(6) :644–654, November 1976.

[DH76b] W. Diffie and M.E. Hellman. Multiuser Cryptographic Techniques. In *AFIPS National Computer Conference*, volume 45 of *AFIPS Conference Proceedings*, pages 109–112. AFIPS Press, 1976.

[DLV03] P. Dusart, G. Letourneux, and O. Vivolo. Differential Fault Analysis on AES. In M. Yung, Y. Han, and J. Zhou, editors, *Applied Cryptography and Network Security (ANCS 2003)*, volume 2846 of *Lecture Notes in Computer Science*, pages 293–306. Springer, 2003.

[DRTV07] J.-G. Dumas, J.-L. Roch, E. Tannier, and S. Varrette. *Théorie des codes - Compression, cryptage, correction*. 2007.

[Dus98] P. Dusart. *Autour de la fonction qui compte le nombre de nombres premiers*. PhD thesis, Université de Limoges, 1998.

[ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Fin94] H. Finney. An RC4 cycle that can't happen. 1994.

[FKJM+06] P.-A. Fouque, S. Kunz-Jacques, G. Martinet, F. Muller, and F. Valette. Power Attack on Small RSA Public Exponent. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 339–353. Springer, 2006.

[FM01] S. Fluhrer and D. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. In Bruce Schneier, editor, *Workshop on Fast Software Encryption (FSE 2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2001.

[FMS01] S.R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In S. Vaudenay and A.M. Youssef, editors, *Selected Areas in Cryptography (SAC 2001)*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001.

[FV03] P.-A. Fouque and F. Valette. The Doubling Attack – *why Upwards Is Better than Downwards*. In C. D. Walter, Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES 2003)*, volume 2779, pages 269–280. Springer, 2003.

[Gir04] C. Giraud. A survey on fault attacks. In *CARDIS 2004*, Smart Card Research and Advanced Applications IV, pages 159–176, 2004.

[Gir05a] C. Giraud. DFA on AES. In V. Rijmen, H. Dobbertin, and A. Sowa, editors, *Fourth Conference on the Advanced Encryption Standard (AES4)*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer-Verlag, 2005.

[Gir05b] C. Giraud. Fault-Resistant RSA Implementation. In L. Breveglieri and I. Koren, editors, *Fault Diagnosis and Tolerance in Cryptography*, pages 142–151, 2005.

[Gir05c] C. Giraud. Procédé de traitement de données impliquant une exponentiation modulaire et un dispositif associé, March 2005. Numéro de publication : FR0503083, WO2006103341.

[Gir07] C. Giraud. *Attaques de Cryptosystèmes Embarqués et Contre-Mesures Associées*. PhD thesis, Université de Versailles Saint-Quentin, 2007.

[GK04] C. Giraud and E.W. Knudsen. Fault Attacks on Signature Schemes. In V. Varadharajan, H. Wang, and J. Pieprzyk, editors, *Proceedings of Information Security and Privacy (ACISP 2004)*, volume 3108 of *Lecture Notes in Computer Science*, pages 478–491. Springer-Verlag, 2004.

[GKW05] M. Gomulkiewicz, M. Kutilwoski, and P. Wlaz. Synchronization Fault Analysis for Breaking A5/1. In Sotiris E. Nikoletseas, editor, *Experimental and Efficient Algorithms (WEA 2005)*, volume 3503 of *Lecture Notes in Computer Science*, pages 415–427. Springer, 2005.

[Hab92] D.H. Habing. The Use of Lasers to Simulate Radiation-Induced Transients in Semiconductors Devices and Circuits. In *IEEE Transactions on Nuclear Science*, volume 39, pages 1647–1653, 1992.

[HGS01] N.A. Howgrave-Graham and N.P. Smart. Lattice Attacks on Digital Signature Schemes. *Design, Codes and Cryptography*, 23 :283–290, 2001.

[HJMM06] M. Hell, T. Johansson, A. Maximov, and W. Meier. A Stream Cipher Proposal : Grain-128. In *IEEE International Symposium on Information Theory*, pages 1614–1618, 2006.

[HR08] M. Hojsik and B. Rudolf. Differential Fault Analysis of Trivium. In Kaisa Nyberg, editor, *Fast Software Encryption (FSE 2008)*, volume 5086 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2008.

[HS04] J. Hoch and A. Shamir. Fault Analysis of Stream Ciphers. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems (CHES 2004)*, volume 3156 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2004.

[HS09] N. Heninger and H. Shacham. Reconstructing RSA Private Keys from Random Key Bits. In S. Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2009.

[JC05] M. Joye and M. Ciet. Practical Fault Countermeasures for Chinese Remaindering Based RSA. In L. Breveglieri and I. Koren, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2005)*, pages 124–132, 2005.

[JLQ99] M. Joye, A. Lenstra, and J.-J. Quisquater. "Chinese Remaindering Based Cryptosystems in the Presence of Faults". *Journal of Cryptology*, 12(4) :241–245, 1999.

[JM06] E. Jochemsz and A. May. A Strategy for Finding Roots of Multivariate Polynomials with New Applications in Attacking RSA Variants. In X. Lai and K. Chen, editors, *Advances in Cryptology - ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2006.

[Joy09] M. Joye. Protecting RSA Against Fault Attacks : The Embedding Method. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*, pages 41–45. IEEE Computer Society, 2009.

[JPY01] M. Joye, P. Paillier, and S.-M. Yen. Secure Evaluation of Modular Functions. In R.J. Hwang and C.K. Wu, editors, *2001 International Workshop on Cryptology and Network Security*, pages 227–229, Taipei,Taiwan, 2001.

[JQBD97] M. Joye, J.-J. Quisquater, F. Bao, and R.H. Deng. RSA-types Signatures in the Presence of Transient Faults. In M. Darnell, editor, *Cryptography and Coding, 6th IMA International Conference*, volume 1355 of *Lecture Notes in Computer Science*, pages 155–160. Springer-Verlag, 1997.

[KAF+10] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D.A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit rsa modulus. Cryptology ePrint Archive, Report 2010/006, 2010.

[KBPJJ08] C.H. Kim, P. Bulens, C. Petit, and J.-J.Quisquater. Fault Attaks on Public Key Elements : Application to DLP-Based Schemes. In S. F. Mjölsnes, S. Mauw, and S. K. Katsikas, editors, *European PKI workshop Public Key Infrastructure (EuroPKI 2008)*, volume 5057 of *Lecture Notes In Computer Science*, pages 182–195. Springer, 2008.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology (CRYPTO 1999)*, volume 1666, pages 388–397. Springer-Verlag, 1999.

[Koc96] P. Kocher. Timing attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology (CRYPTO 1996)*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KQ07] C.H. Kim and J.-J. Quisquater. Fault Attacks for CRT Based RSA : New Attacks, New Results and New Countermeasures. In *Information Security Theory and Practices, Smart Cards, Mobile and Ubiquitus Computing Systems*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer-Verlag, 2007.

[KY09] A. Kirkanski and A.M. Youssef. Differential Fault Analysis of Rabbit. In *Selected Areas in Cryptography (SAC 2009)*, Lecture Notes in Computer Science, pages 200–217. Springer, 2009.

[Len96] A. Lenstra. Memo on RSA Signature Generation in the presence of Faults, 1996.

[LHL93] A.K. Lenstra and Jr. H.W. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.

[LLL86] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalem*, 261(4) :515–534, 1986.

[May03] A. May. *New RSA Vulnerabilities Using Lattice Reduction Methods*. PhD thesis, Univeristy of Paderborn, 2003.

[MDAB10] S. Murdoch, S. Drimer, R. Andersson, and M. Bond. Chip and PIN is broken. In D. Evans and G. Vigna, editors, *IEEE Symposium on Security & Privacy (SSP 2010)*. IEEE Computer Society, 2010.

[Mon85] P. Montgomery. Modular Multiplication without trial divison. *Mathematics of Computation*, 44 :519–521, 1685.

[Mon87] P. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48 :243–264, 1987.

[MOVR97] A.J. Menezes, P.C. Van Oorschot, S. A. Vanstone, and R.L. Rivest. Handbook of Applied Cryptography, 1997.

[Mui06] J.A. Muir. Seifert's RSA Fault Attack : Simplified Analysis and Generalizations. Cryptology ePrint Archive, Report 2005/458, 2006.

[MW78] T. May and M. Woods. A new physical mechanism for soft errors in dynamic memories. In *Proceedings of the 16-th International Reliability Physics Symposium*, 1978.

[NNTW05] D. Naccache, P.Q. Nguyen, M. Tunstall, and C. Whelan. Experimenting with Faults, Lattices and the DSA. In S. Vaudenay, editor, *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 16–28. Springer-Verlag, 2005.

[NS02] P.Q. Nguyen and I.E. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *Journal of Cryptology*, 15(3) :151–176, 2002.

[oSN77] National Institute of Standards and Technology (NIST). FIPS PUB 46 : The Data Encryption Standard, January 1977.

[oSN94] National Institute of Standards and Technology (NIST). FIPS PUB 186 : Digital Signature Standard (DSS), 1994.

[oSN00] National Institute of Standards and Technology (NIST). FIPS PUB 186-2 : Digital Signature Standard (DSS), January 2000.

[oSN01a] National Institute of Standards and Technology (NIST). FIPS PUB 180-2 : Secure Hash Standard, August 2001.

[oSN01b] National Institute of Standards and Technology (NIST). FIPS PUB 186-2 + CHANGE NOTICE 1 : Digital Signature Standard (DSS), October 2001.

[oSN01c] National Institute of Standards and Technology (NIST). FIPS PUB 197 : Advanced Encryption Standard, November 2001.

[oSN07] National Institute of Standards and Technology (NIST). FIPS PUB 180-3 : Secure Hash Standard, June 2007.

[oSN09] National Institute of Standards and Technology (NIST). FIPS PUB 186-3 : Digital Signature Standard (DSS), June 2009.

[Ott04] M. Otto. *Fault Attacks and Countermeasures*. PhD thesis, University of Paderborn, December 2004.

[PP04] S. Paul and B. Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In B.K. Roy and W. Meier, editors, *Workshop on Fast Software Encryption (FSE 2004)*, volume 3017 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2004.

[QC82] J.-J. Quisquater and C. Couvreur. Fast Decipherment Algorithm for RSA Public-Key Cryptosystem. *Electronics Letters*, 18(21)(21) :905–907, 1982.

[QS02] J.-J. Quisquater and D. Samyde. Eddy Current for Magnetic Analysis with Active Sensor. In *e-Smart 2002*, 2002.

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Thoery 12*, 1 :128–138, 1980.

[Riv09] M. Rivain. Securing RSA Against Fault Analysis by Double Addition Chain Exponentiation. In M. Fischlin, editor, *RSA Cryptographer's Track (CT-RSA 2009)*, volume 5473 of *Lecture Notes in Computer Science*, pages 459–480. Springer, 2009.

[RSA78] R.L. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signature and Public-Key Cryptosystems. In *Communications of the ACM*, volume 21, pages 120–126, 1978.

[SA02] S.P. Skorobogatov and R.J. Anderson. Optical Fault Induction Attacks. In B.S. Kaliski Jr., Ç.K. Koç, and C. Parr, editors, *Cryptographic Hardware and Embedded Systems (CHES 2002)*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer-Verlag, 2002.

[Sch96] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc, 2nd edition edition, 1996.

[SE94] C.P. Schnorr and M. Euchner. Lattice Basis Reduction : Improved practical algorithms and solving subset sum problems. In *Math. Programming*, volume 66, pages 181–199, 1994.

[Sei05] J.-P. Seifert. On Authenticated Computing and RSA-Based Authentication. In *ACM Conference on Computer and Communications Security (CCS 2005)*, pages 122–127. ACM Press, 2005.

[Sha97] A. Shamir. "Improved Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attacks". *Presented at the Rump Session of Eurocrypt'97*, 1997.

[Sha04] A. Shamir. Stream Ciphers : Dead or Alive ? In P. J. Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, page 78. Springer, 2004.

[Sho] V. Shoup. Number Theory C++ Library (NTL).

[Sho05] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[Sko06] S.P. Skorobogatov. Optically Enhanced Position-Locked Power Analysis. In *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 2006.

[Ste97] J. Stern. *La science du secret*. 1997.

[Sti95] D. Stinson. *Cryptography : Theory and Practice*. CRC Press Inc, 1995.

[SWD96] O. Schirokauer, D. Weber, and T. Denny. Discrete Logarithms : The effectiveness of the Index Calculus Method. In *Algorithmic Number Theory (ANTS-II 1996)*, volume 1122 of *Lecture Notes in Computer Science*, pages 337–362. Springer, 1996.

[Vig08] D. Vigilant. RSA with CRT : A new Cost Effective Solution to Twart Fault Attacks. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems (CHES 2008)*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.

[Wag04] D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In *Proceedings of the 11th ACM Conference on Computer Security (CCS 2004)*, pages 92–97. ACM, 2004.

[YHL08] L. Yi, W. Huaxiong, and S. Ling. Cryptanalysis of Rabbit. In T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, editors, *11th International Conference on Information Security (ISC 2008)*, volume 5222 of *Lecture Notes In Computer Science*, pages 204–214. Springer-Verlag, 2008.

[YJ00] S.-M. Yen and M. Joye. Checking Before Output May not be Enough Against Fault-Based Cryptanalysis. In *IEEE Transactions on Computers*, pages 367–370, 2000.

[YK04] S.-M. Yen and D. Kim. Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection. In *Fault Diagnosis and Tolerance in Cryptography (FDTC 2004)*, pages 381–385, 2004. IEEE Computer Society.

[YKLM01] S.-M. Yen, D. Kim, S. Lim, and S. Moon. RSA Speedup with Residue Number System Immune Against Hardware Fault Cryptanalysis. In K. Kim, editor, *Information Security and Cryptology (ICISC 2001)*, volume 2288 of *Lecture Notes in Computer Science*, pages 397–413. Springer-Verlag, 2001.

[YKM06] S.-M. Yen, D. Kim, and S. Moon. Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, volume 4236 of *Lecture Notes in Computer Science*, pages 53–61. Springer-Verlag, 2006.

[YMH02] S.-M. Yen, S. Moon, and J.-C. Ha. Hardware Fault Attack on RSA with CRT Revisited. In C.H. Lim and P.J. Lee, editors, *Information Security and Cryptology (ICISC 2002)*, volume 2587 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2002.

[Zie79] J. Ziegler. Effect of Cosmic Rays on Computer Memories. *Science*, 206 :776–788, 1979.

Résumé

Avec l'avènement des attaques par canaux auxiliaires, à la fin des années 90, les preuves de sécurité algébriques ne sont plus suffisantes pour garantir la sécurité de crypto-systèmes embarqués. L'une de ces attaques, la *Differential Fault Analysis*, propose d'exploiter les perturbations malicieuses de composants cryptographiques pour en extraire des données secrètes. L'objet de cette thèse est d'étendre le champ d'application de l'analyse de perturbations en proposant de nouvelles attaques basées sur des modèles de faute innovants mais réalistes.

Alors qu'il est rapidement devenu nécessaire de protéger les clés privées contre les perturbations, de récents travaux ont démontré que la perturbation d'éléments publics pouvait aussi engendrer une fuite d'information critique. Dans ce cadre, nous nous intéresserons particulièrement aux implantations classiques de deux crypto-systèmes asymétriques des plus répandus : le RSA et le DSA. Nous étudierons leur comportement vis-à-vis de perturbations intervenant pendant leur exécution, ce qui n'avait jamais été fait auparavant.

Dans un second temps, nous avons suivi l'émergence de nouveaux algorithmes de chiffrement à flot. La structure mathématique de ces nouveaux algorithmes étant désormais plus forte, nous avons voulu évaluer la robustesse de leur implantation face aux perturbations. A ce titre, nous nous sommes intéressés à deux des finalistes du projet *eSTREAM* : Grain-128 et Rabbit. Enfin, cette thèse soulignera la difficulté de protéger les implantations de crypto-systèmes contre les perturbations en prenant l'exemple du RSA-CRT.

Mots-clés: Analyse de perturbation, cryptographie asymétrique, perturbation d'éléments publics, chiffrement à flot

Abstract

Since the advent of side channel attacks, at the end of the 90's, classical cryptanalysis is no longer sufficient to ensure the security of embedded cryptosystems. Among side channel attacks, *Differential Fault Analysis* is a powerful way to recover secret information from malicious perturbations of a cryptographic hardware. The purpose of the thesis is to extend the scope of fault attacks by providing brand new attacks based on innovative but realistic fault models.

Whereas private keys have been rapidly protected against perturbations, recent works addressed the issue of protecting also non-critical elements, such as public keys, since their perturbation may leak secret information. We will investigate on this area by focusing on classical implementations of two very popular cryptosystems : RSA and DSA. In details, we will detail how to exploit faults on the public modulus that occurred during their execution. To the best of our knowledge, these cryptographic algorithms have never been studied according to such a fault model.

We have also followed the emergence of new stream ciphers. Since their mathematical structure are stronger, we wanted to evaluate the robustness of their implementation against malicious faults. In this context we studied two of the *eSTREAM* finalists : Grain-128 and Rabbit.

Finally, this thesis emphasizes the difficulty for elaborating efficient countermeasure against faults by describing what have been done for CRT-RSA.

Keywords: Fault analysis, asymmetric cryptography, public key perturbation, stream ciphers