Glossaire

3GPP	***3**rd **G**eneration **P**artnership **P**roject* Projet de standardisation pour les systèmes de communications mobiles de troisième génération.
A3/A8	Schéma générique pour le calcul des fonctions d'authentification (A3) et de génération de clé de session (A8) utilisé dans le système GSM. Par extension, on désigne également par A3/A8 tout algorithme instanciant ce schéma générique.
AES	***A**dvanced **E**ncryption **S**tandard* Algorithme de chiffrement par bloc standardisé en 2001 en remplacement de l'algorithme DES obsolète
CBC	***C**ipher **B**lock **C**haining* Mode d'opération d'un chiffrement par bloc qui consiste, préalablement à son chiffrement, à XOR-er chaque bloc du message par le chiffré du bloc précédent.
CFA	***C**ollision **F**ault **A**nalysis* Technique d'attaque physique par analyse de faute qui consiste à extraire de l'information à partir d'une collision obtenue à l'aide d'une faute provoquée.
CMOS	***C**omplementary **M**etal **O**xide **S**emi-conductor* Famille de composants électroniques à faible consommation électrique. L'essentiel de la consommation de courant de ces composants ne se produit que lors d'un basculement d'un état logique à son état complémentaire.
COMP128	Exemple d'algorithme A3/A8 pour l'authentification et l'échange de clé sur un réseau GSM. Cet algorithme, initialement confidentiel, a été dévoilé en 1998 par Marc Briceno et cassé dans la foulée par Ian Goldberg et David Wagner [BGW98]. Il ne devrait plus être utilisé.
CPA	***C**orrelation **P**ower **A**nalysis* Technique d'attaque physique par analyse du courant qui consiste à corréler la consommation avec les valeurs des données manipulées. La CPA est parfois appelée CEMA dans le cas de l'analyse du rayonnement électromagnétique.
CPU	***C**entral **P**rocessing **U**nit* Unité centrale de traitement de l'information qui constitue le cœur calculatoire d'un micro-processeur.

CRT — ***C**hinese **R**emainder **T**heorem*
Méthode, basée sur le théorème du même nom, permettant de rendre le calcul d'une signature RSA, en théorie, quatre fois plus rapide que la méthode standard.

DES — ***D**ata **E**ncryption **S**tandard*
Algorithme de chiffrement par bloc utilisé du milieu des années 1970 jusque vers le début des années 2000. Devenu obsolète en raison d'une taille de clé insuffisante, il a été remplacé par une version plus forte, le Triple-DES, mais surtout par l'AES, nouveau standard depuis 2001.

DESX — Variante du DES proposée par Ron Rivest utilisant deux clés supplémentaires servant à XOR-er l'entrée et la sortie de l'algorithme standard.

DFA — ***D**ifferential **F**ault **A**nalysis*
Technique d'attaque physique par analyse de faute qui consiste à extraire de l'information à partir de la différentielle observée entre un chiffré de référence et une version obtenue à l'aide d'une faute provoquée.

DPA — ***D**ifferential **P**ower **A**nalysis*
Technique d'attaque physique par analyse du courant qui consiste à mettre en évidence une faible différence de consommation moyenne, en fonction de la valeur d'un bit d'une donnée manipulée. La DPA est parfois appelée DEMA dans le cas de l'analyse du rayonnement électromagnétique.

ECB — ***E**lectronic **C**ode **B**ook*
Mode d'opération d'un chiffrement par bloc qui consiste à chiffrer les blocs du message indépendamment les uns des autres.

ECM — ***E**lliptic **C**urve **M**ethod*
Méthode de factorisation d'entiers dont la complexité dépend principalement de la taille du facteur recherché plutôt que de celle de l'entier factorisé.

EEPROM — ***E**lectrically-**E**rasable **P**rogrammable **R**ead-**O**nly **M**emory*
Type de mémoire morte pouvant être effacée électriquement et programmée. Ce type de mémoire, dont une technologie plus récente est appelée mémoire FLASH, sert par exemple à stocker les données d'un utilisateur sur une carte à puce.

FDH — ***F**ull **D**omain **H**ash*
Schéma de signature basé sur le RSA dans lequel la sortie de la fonction de hachage a la même taille que le module.

GSM — ***G**lobal **S**ystem for **M**obile communications*
Norme numérique de seconde génération pour la téléphonie mobile.

IBE — ***I**dentity-**B**ased **E**ncryption*
Chiffrement à clé publique permettant d'utiliser une chaîne de caractères quelconque, par exemple l'adresse mail du destinataire, comme clé publique.

IDEA — ***I**nternational **D**ata **E**ncryption **A**lgorithm*
Algorithme de chiffrement par bloc inventé par Xuejia Lai et James Massey [LM91].

IFA	***I**neffective **F**ault **A**nalysis* Technique d'attaque physique par analyse de faute qui consiste à extraire de l'information à partir d'une faute provoquée n'ayant pas d'effet sur le résultat d'une opération arithmétique ou logique.
LFSR	***L**inear **F**eedback **S**hift **R**egister* Registre à décalage à rétro-action 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 Du nom de ses inventeurs, LLL [LLL82] est un algorithme de réduction de réseau souvent utilisé dans la cryptanalyse de schémas de chiffrement à clé publique.
LSB	***L**east **S**ignificant **B**it* Bit de droite, le moins significatif, d'un octet.
LUT	***L**ook-**U**p **T**able* Table de valeurs utilisée pour l'implémentation d'une boîte de substitution (S-Box).
MILENAGE	Schéma générique de calcul de plusieurs fonctions cryptographiques utilisées dans le standard 3GPP pour les systèmes de communications mobiles de troisième génération.
MSB	***M**ost **S**ignificant **B**it* Bit de gauche, le plus significatif, d'un octet.
NBS	***N**ational **B**ureau of **S**tandard* Organisme états-unien de normalisation, de support et de recherche sur les nouvelles technologies. Le NBS a changé de nom en 1988 pour devenir le NIST.
NIST	***N**ational **I**nstitute of **S**tandards and **T**echnology* Agence du Département du 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.
NVM	***N**on-**V**olatile **M**emory* Mémoire conservant les données qu'elle contient même après mise hors-tension. La ROM, l'EEPROM et la FLASH sont des mémoires de ce type.
OAEP	***O**ptimal **A**symmetric **E**ncryption **P**adding* Schéma de padding introduit en 1994 par MIHIR BELLARE et PHIL ROGAWAY [BR95]. Utilisé pour le RSA, RSA-OAEP est prouvé sûr dans le modèle de l'oracle aléatoire.
PFDH	***P**robabilistic **F**ull **D**omain **H**ash* Variante de FDH considérant une famille de fonctions de hachage. La signature est calculée d'après une fonction de hachage choisie aléatoirement dans cette famille. L'aléa utilisé est joint à la signature.

PGP — ***P**retty **G**ood **P**rivacy*
Logiciel tout public de communication électronique sécurisée utilisant la cryptographie asymétrique. PGP a été développé par Philip Zimmermann et rendu disponible en libre téléchargement en 1991.

PIN — ***P**ersonal **I**dentification **N**umber*
Code personnel d'identification. Le contrôle de la connaissance du PIN permet de n'autoriser l'utilisation d'une carte bancaire ou d'une carte SIM qu'à son possesseur légitime.

PKCS — ***P**ublic-**K**ey **C**ryptography **S**tandards*
Ensemble de spécifications conçues par la société *RSA Security* pour l'implémentation des techniques de cryptographie à clé publique.

PKI — ***P**ublic-**K**ey **I**nfrastructure*
Ensemble de composants, procédures et logiciels permettant de gérer le cycle de vie des certificats numériques des clés publiques.

PRNG — ***P**seudo-**R**andom **N**umber **G**enerator*
Générateurs de nombres pseudo-aléatoires, les PRNG sont souvent basés sur des LFSR, et sont utilisés par exemple comme source d'aléa pour certaines contre-mesures vis-à-vis des attaques physiques.

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

QDI — ***Q**uasi **D**elay-**I**nsensitive*
Classe de circuits asynchrones ayant la propriété d'être insensibles aux différences de délais de propagation de ses différentes portes.

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

RIJNDAEL — Algorithme de chiffrement par bloc inventé par Vincent Rijmen et Joan Daemen [DR99, DR00, DR02] qui a été sélectionné parmi plusieurs propositions pour être adopté en 2001 comme standard AES.

RNG — ***R**andom **N**umber **G**enerator*
Générateur de nombres aléatoires. Les RNG utilisent souvent la mesure d'une grandeur physique imprédictible (bruit thermique par exemple) et sont utilisés notamment pour la génération de clés cryptographiques.

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.

RPI — ***R**andom **P**rocess **I**nterrupt*
Interruption aléatoire de processus. Les RPI permettent une désynchronisation temporelle utile comme contre-mesure vis-à-vis de certaines attaques physiques.

RSA — **Rivest Shamir Adelman**
Du nom de ses inventeurs, le RSA est un crypto-système à clé publique dont la sécurité repose sur la difficulté supposée de la factorisation des grands entiers.

S-Box — ***S**ubstitution **B**ox*
Boîte de substitution assurant la fonction de diffusion d'un algorithme de chiffrement par bloc.

SCA — ***S**ide-**C**hannel **A**nalysis*
Analyse de canaux auxiliaires. Ce type d'attaque physique permet d'extraire de l'information sur un secret (clé cryptographique, PIN,...) à partir de l'observation et de l'analyse d'une grandeur physique, le *canal auxiliaire*, dépendant de la valeur de ce secret.

SCARE — ***S**ide-**C**hannel **A**nalysis for **R**everse **E**ngineering*
Application de l'analyse de canaux auxiliaires visant la rétro-conception de tout ou partie des spécifications d'un algorithme confidentiel, plutôt que la révélation d'une clé secrète.

SEA — ***S**afe-**E**rror **A**nalysis*
Technique d'attaque physique par analyse de faute qui consiste à extraire de l'information à partir d'une faute provoquée dont l'effet n'a pas d'influence sur le résultat final d'un calcul cryptographique.

SIM — ***S**ubscriber **I**dentification **M**odule*
Carte à puce nécessaire au fonctionnement des téléphones portables de type GSM.

SNR — ***S**ignal to **N**oise **R**atio*
Rapport signal à bruit, utilisé comme indicateur de qualité d'un signal transportant une information.

SPA — ***S**imple **P**ower **A**nalysis*
Technique d'attaque physique permettant d'extraire de l'information sur un secret à partir de la mesure et de l'analyse du courant consommé lors de l'exécution d'un calcul cryptographique. La SPA est parfois appelée SEMA dans le cas de l'analyse du rayonnement électromagnétique.

SSH — ***S**ecure **SH**ell*
Programme informatique et protocole de communication sécurisant la connection à un ordinateur distant et le transfert de fichiers.

SSL — ***S**ecure **S**ocket **L**ayer*
Protocole de communication sécurisée entre deux ordinateurs fournissant les fonctions d'authentification, d'intégrité et de confidentialité. OpenSSL est une implémentation libre (*open source*) de SSL.

TA — ***T**iming **A**nalysis*
Technique d'attaque physique permettant d'extraire de l'information sur un secret à partir de la mesure de la durée d'exécution d'un calcul cryptographique.

WI-FI — Contraction de ***WI**reless **FI**delity*
Technologie de réseau informatique sans fil mise en place pour fonctionner en réseau interne et devenue un moyen d'accès à haut débit à Internet.

Sommaire

Partie II Les attaques

Partie III Les contre-mesures

Partie IV Les attaques sur algorithmes inconnus

Partie V Annexe

Première partie

Introduction

Chapitre 1

La sécurité physique

Sommaire

Nous donnons dans ce chapitre quelques notions introductives permettant au lecteur non familier avec le domaine des attaques physiques d'aborder plus sereinement les parties II à IV qui constituent le corps de ce document.

Dans un premier temps, et sans entrer dans les détails, nous décrivons le dispositif cryptographique embarqué probablement le plus répandu au monde : la carte à puce. Nous abordons ensuite la cryptographie en expliquant ses principes de base et en présentant les algorithmes les plus utilisés. Enfin, nous donnons un aperçu des grandes classes d'attaques physiques, de leurs principes et des contre-mesures destinées à les rendre difficiles ou à les contrer.

1.1 Qu'est-ce qu'une carte à puce ?

Une carte à puce est un objet en plastique contenant un circuit micro-électronique possédant la plupart des fonctionnalités d'un micro-ordinateur. La portabilité de cet objet (un portefeuille peut en contenir une dizaine), et sa capacité à défendre les données et programmes qu'il contient contre les attaques de nature intrusive (abrasion chimique, observation au microscope électronique, etc...) lui procurent naturellement sa fonction essentielle de "bunker" pour le stockage de clés et l'exécution d'algorithmes cryptographiques dans les usages mobiles nécessitant un haut degré de sécurité comme par exemple :

– la sécurité des transactions (carte bancaire),
– l'identification et le contrôle d'accès (passeport électronique, badge d'entreprise),

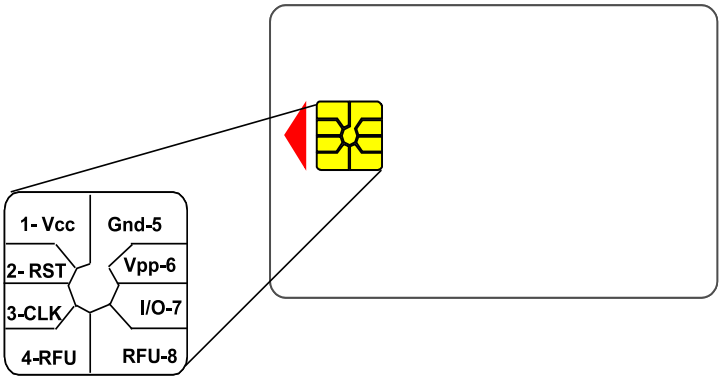


FIG. 1.1 – Les contacts d'un module de carte à puce.

- l'authentification pour l'accès à un service (téléphonie mobile, télévision à péage),
- la confidentialité des données lues ou échangées (carte de santé)
- etc...

Vue de l'extérieur, une carte à puce montre une pièce métallique dorée comprenant huit zones. Il s'agit du module servant à établir les contacts électriques entre le micro-processeur et le monde extérieur. Les huit zones correspondent à huit contacts différents (voir la Figure 1.1) dont 6 seulement ont un rôle défini par la norme ISO 7816-3 :

1. **Vcc** : tension d'alimentation (5, 3 ou 1,8 volts) fournie à la carte par le lecteur avec lequel elle s'interface,
2. **RST** : signal de remise à zéro de la carte,
3. **CLK** : horloge fournie par le lecteur à la carte, qui est nécessairement utilisée pendant les phases de communication entre la carte et le lecteur même si la carte peut posséder son propre oscillateur,
4. **RFU** : réservé pour un usage futur,
5. **Gnd** : potentiel de référence (masse),
6. **Vpp** : tension de programmation (21 volts) qui servait par le passé à alimenter une pompe de charge utilisée pour les écritures en EEPROM (inutilisé de nos jours),
7. **I/O** : ligne transportant de manière bi-directionnelle les données échangées entre la carte et le lecteur,
8. **RFU** : réservé pour un usage futur.

Sous le module se trouve le micro-processeur. C'est un composant dont la surface ne doit pas excéder 25 mm^2. On en trouve actuellement dont l'architecture interne peut être à 8, 16 ou 32 bits de granularité de calcul ou d'exécution. Il comprend différentes parties parmi lesquelles on peut trouver :

- la CPU qui est l'unité centrale de traitement,
- la mémoire RAM pour stocker des résultats intermédiaires de calcul en cours de fonctionnement,
- la mémoire ROM qui contient essentiellement le code programme du système d'exploitation et des applications de la carte,

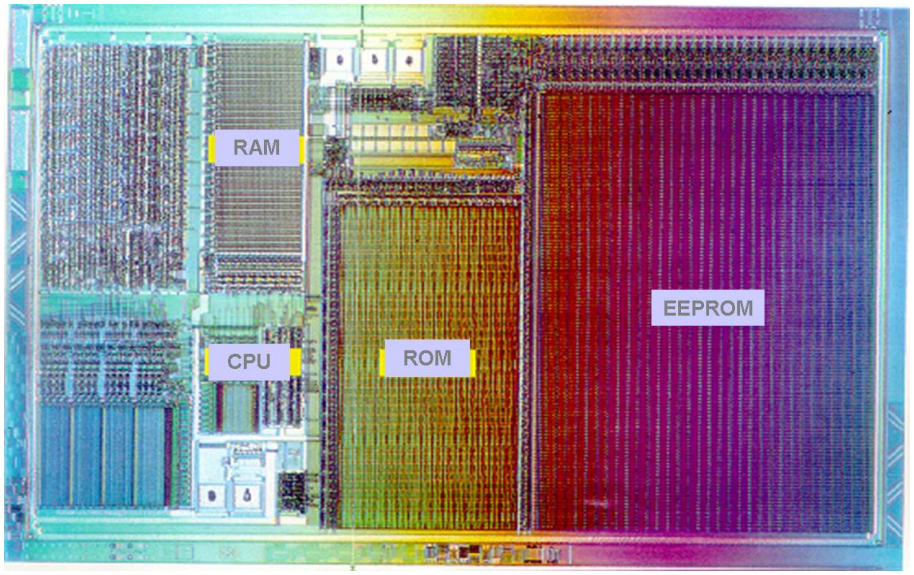


FIG. 1.2 – Les différentes parties d'un micro-processeur de carte à puce.

- la mémoire non volatile (EEPROM ou mémoire Flash) qui permet de stocker, comme sur un disque dur, les données spécifiques à l'utilisateur, y compris lorsque la carte n'est pas alimentée,
- un générateur d'aléa (RNG) indispensable pour la génération de clés cryptographiques et le bon fonctionnement de certaines contre-mesures,
- un ou plusieurs crypto-processeurs permettant d'effectuer rapidement des calculs cryptographiques spécialisés (DES, AES, multiplication modulaire, etc...)
- des capteurs de sécurité destinés à détecter des conditions anormales de fonctionnement.

La Figure 1.2 montre la photo d'un modèle relativement ancien de puce et identifie certaines de ses zones.

Nous renvoyons le lecteur à l'ouvrage [DJRV00] pour une présentation plus détaillée de la carte à puce.

1.2 Les algorithmes cryptographiques

1.2.1 Cryptographie symétrique

La cryptographie est utilisée depuis l'antiquité pour cacher le sens du contenu d'une communication entre deux interlocuteurs. Depuis ces temps anciens où l'on attribue à CÉSAR une technique simple consistant à appliquer à chaque symbole du message clair une rotation de valeur fixe à l'intérieur de l'alphabet utilisé, jusque vers la fin des années 1970, les algorithmes de chiffrement reposaient tous sur le principe que l'émetteur et le destinataire d'un message confidentiel devaient chacun utiliser une même valeur secrète de clé. Cette clé K paramètre une fonction de chiffrement $\mathrm{E}_K(\cdot)$ qui sert à produire une version inintelligible du message clair M. Le chiffré $C = \mathrm{E}_K(M)$ obtenu est alors transmis sur un canal de communication non sûr au destinataire qui lui appliquera la transformation inverse $\mathrm{D}_K(\cdot) = \mathrm{E}_K^{-1}(\cdot)$ qui permet de déchiffrer C pour retrouver le message inital $M = \mathrm{D}_K(C)$. La Figure 1.3 illustre ce principe de chiffrement, appelé cryptographie *symétrique* (on parle aussi de cryptographie *à clé secrète*) du fait

FIG. 1.3 – Le principe de la cryptographie symétrique.

que la valeur de la clé utilisée par l'émetteur pour chiffrer est la même que celle utilisée par le destinataire pour déchiffrer.

La cryptographie symétrique comprend deux sous-familles de fonctions de chiffrement : les algorithmes de *chiffrement à flot* et les algorithmes de *chiffrement par bloc*. Les algorithmes de chiffrement à flot sont essentiellement des générateurs pseudo-aléatoires permettant de dériver de la clé K une séquence de bits, dite *séquence chiffrante*, de longue période. La clé joue ici le rôle de graine pour ce générateur de bits, et la séquence chiffrante sera utilisée pour masquer le flot de bits du message à la façon du chiffrement à masque jetable de VERNAM. Les algorithmes de chiffrement par bloc utilisent la clé K pour transformer des blocs de texte clair de longueur fixe (typiquement 8 ou 16 octets) en des blocs de chiffré de même longueur. Lorsque le message est plus grand que la taille d'un bloc, celui-ci est découpé en plusieurs blocs qui sont chiffrés indépendamment (mode dit "ECB")[1] ou de manière chaînée (par exemple, mode dit "CBC").

Pendant très longtemps, les utilisateurs de la cryptographie symétrique ont utilisé des fonctions de chiffrement elles-mêmes confidentielles, estimant que le secret de la transformation opérée sur le message participait à la sécurité du système. Ce n'est qu'en 1883 qu'AUGUSTE KERCKHOFFS publie son essai *La Cryptographie militaire* dans lequel il propose le principe qui portera son nom et qui énonce que la sécurité d'un système cryptographique ne doit pas reposer sur le caractère secret du système de codage. Seule une courte quantité d'information, la *clé*, nécessite d'être gardée secrète.

La compréhension qu'un algorithme de chiffrement peut être révélé permettra bien plus tard, lorsque l'augmentation et la diffusion des moyens de calcul automatisé auront suscité un besoin de cryptographie à usage civil, la naissance du premier standard public de chiffrement. C'est ainsi que fut adopté en 1976 l'algorithme DES (en anglais, *Data Encryption Standard*), standard qui participera à une explosion de l'usage de la cryptographie dans le monde. Après 25 ans de règne sur la cryptographie symétrique, et devenu obsolète vers le début des années 90 en raison d'une taille de clé trop faible par rapport à l'évolution des moyens de calcul, le DES sera remplacé en 2001 par l'AES (en anglais, *Advanced Encryption Standard*).

Bien que de nombreux autres algorithmes de crytographie symétrique aient été définis par la communauté scientifique depuis l'apparition du DES, nous allons maintenant décrire plus en détail le DES et l'AES qui ont été et sont les plus utilisés, notamment dans les systèmes à base de cartes à puce.

[1] Notons qu'il est recommandé de ne pas utiliser le mode ECB qui peut se prêter à la manipulation du chiffré.

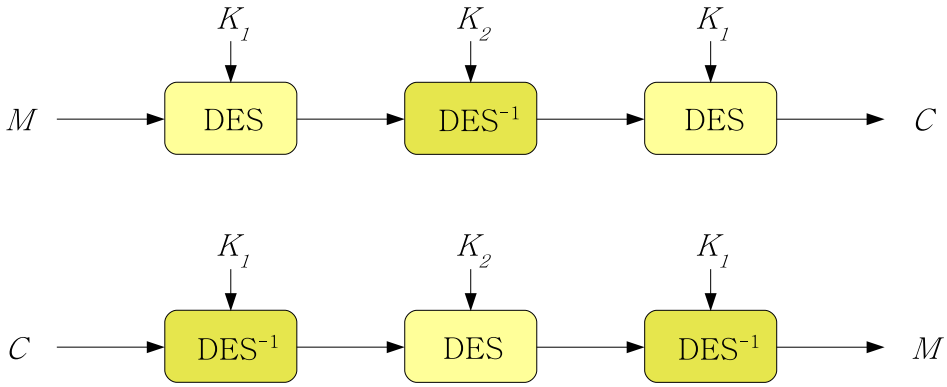


FIG. 1.4 – L'algorithme Triple-DES en mode chiffrement (en haut) et déchiffrement (en bas).

L'algorithme DES

En 1973, le *National Bureau of Standard*, organisme états-unien de standardisation, souhaite la création d'un système de chiffrement initialement destiné aux données sensibles mais non classifiées du gouvernement. Après un premier appel à proposition infructueux, cet organisme se tourne alors vers IBM qui possède un algorithme nommé *Lucifer*. Après avoir subi plusieurs modifications, dont certaines pourraient avoir été imposées par la NSA (en anglais, *National Security Agency*), cet algorithme deviendra officiellement le DES approuvé en novembre 1976 et publié comme standard en janvier 1977 [NBS77]. Le DES a été successivement réaffirmé comme standard en 1983, 1988, 1993. En 1999, il est reconduit à nouveau une quatrième fois, et ce n'est qu'en 2004 qu'il est recommandé de l'utiliser sous la forme du Triple-DES (voir Figure 1.4) qui utilise une clé (K_1, K_2) de 112 bits au lieu d'une clé K de 56 bits.

L'algorithme DES permet de chiffrer des blocs de 64 bits à l'aide d'une clé de 56 bits. Cette fonction est essentiellement structurée comme un réseau de FEISTEL encadré par une permutation de bits initiale IP et une permutation de bits finale FP, inverses l'une de l'autre. Un réseau de FEISTEL consiste en l'application itérée d'un maillon de FEISTEL, autrement appelé *tour*. Le tour numéro i transforme son entrée (L_{i-1}, R_{i-1}) en la sortie (L_i, R_i) vérifiant :

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f_{K_i}(R_{i-1}) \quad . \end{aligned}$$

La partie gauche de la sortie est obtenue par simple recopie de la partie droite de l'entrée. La moitié droite de la sortie, R_i, est le résultat de l'opération XOR entre l'entrée gauche L_{i-1} et la transformée $f_{K_i}(R_{i-1})$ de l'entrée droite par une fonction de tour bijective. Les fonctions de tour sont paramétrées par des *clés de tour* K_i, elles-mêmes dérivées de la clé K.

La Figure 1.5 représente le DES comme réseau de Feistel avec son schéma de dérivation des clés de tour. Une première permutation compressive PC1 extrait 56 des 64 bits initiaux de la clé K en ignorant les bits de poids faible de chacun de ses 8 octets. Ces 56 bits sont alors séparés en deux registres C et D de 28 bits chacun, lesquels subissent à chaque tour une rotation de leurs bits (de 1 ou 2 bits selon le numéro du tour). À chaque tour les sorties de ces registres sont regroupées et une permutation compressive PC2 en extrait les 48 bits de la clé de tour K_i.

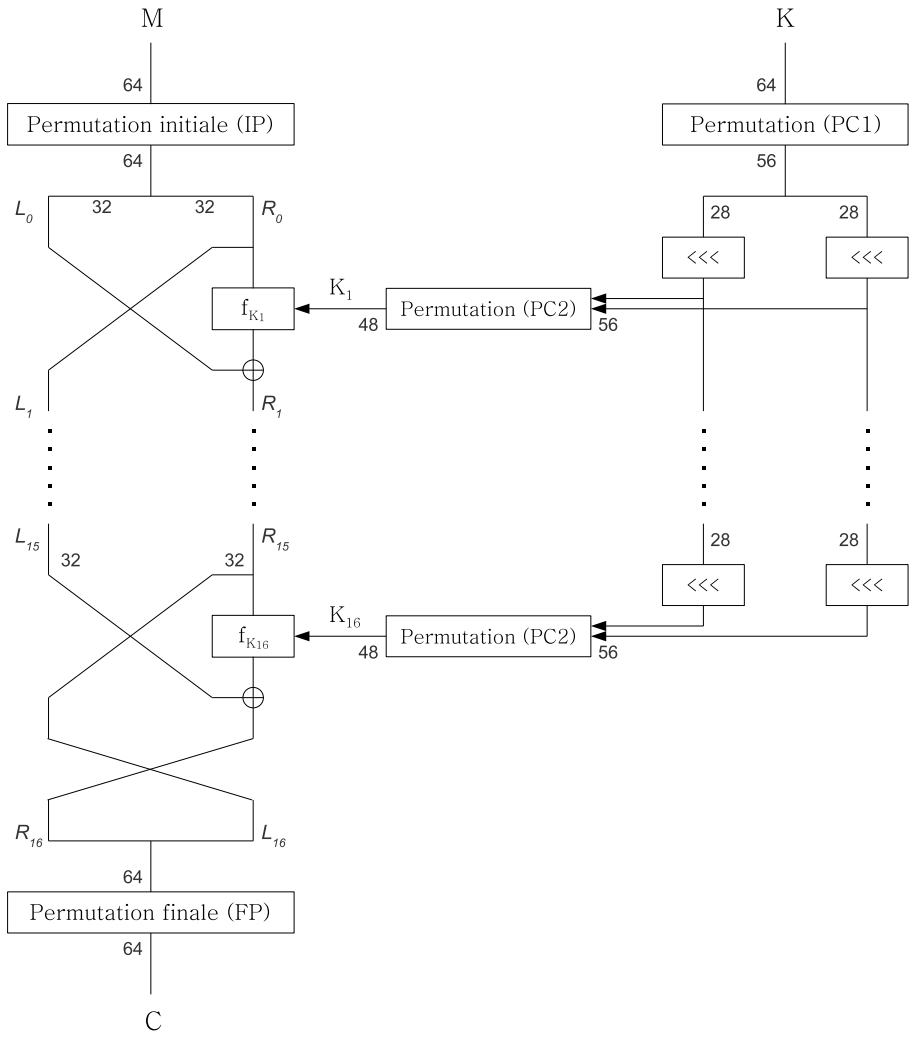


FIG. 1.5 – L'algorithme DES : un réseau de FEISTEL et son schéma de dérivation des clés de tour.

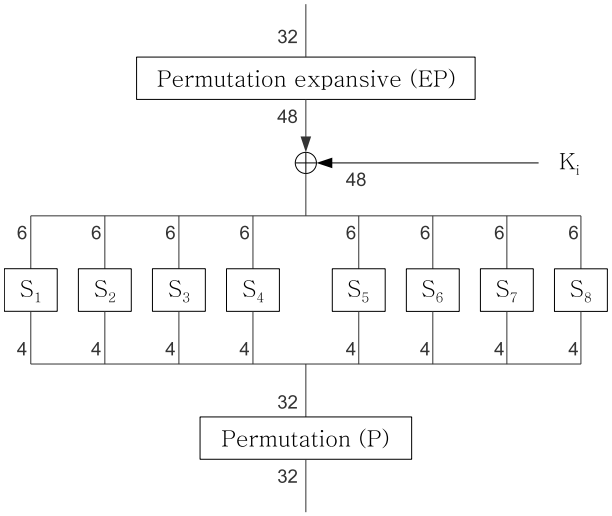


FIG. 1.6 – La fonction de tour du DES.

La fonction de tour est détaillée à la Figure 1.6. La permutation expansive commence par dupliquer 16 des 32 bits de R_{i-1} pour fournir 48 bits qui sont ensuite XOR-és avec la clé de tour K_i. Ce XOR de deux valeurs de 48 bits peut aussi s'interpréter comme huit XOR sur des valeurs de 6 bits. Les sorties de ces huit XOR sont utilisées dans des tables de substitution à 64 entrées, appelées S-Box. Ces S-Box forment la partie non linéaire de l'algorithme et assurent la fonction de *confusion* nécessaire dans tout algorithme de chiffrement[2]. Les sorties sur 4 bits des huit S-Box sont ensuite regroupées pour former 32 bits qui sont alors mélangés par la permutation P.

Comme pour tout réseau de FEISTEL, la fonction de déchiffrement DES ne diffère de celle de chiffrement que par l'ordre inversé dans lequel les clés de tour sont considérées. En l'occurrence, cela est réalisé en effectuant les rotations des registres C et D vers la droite plutôt que vers la gauche, et en supprimant celle prévue au premier tour.

L'algorithme AES

L'algorithme AES (en anglais, *Advanced Encryption Standard*) a été adopté à la suite d'un appel à candidature lancé en 1997 par le NIST[3] auprès de la communauté scientifique et industrielle en vue du remplacement du DES. Pas moins de 15 candidats ont été proposés, chacun devant répondre à un cahier des charges définissant des critères liés à la sécurité cryptographique, ainsi qu'aux performances et à la compacité de l'implémentation aussi bien sous forme logicielle que matérielle. La sélection de l'algorithme vainqueur s'est faite en deux phases et constitue un bel exemple de transparence pour l'analyse et le choix d'un standard de fonction de chiffrement. Parmi les 5 propositions finalistes, c'est l'algorithme RIJNDAEL [DR99, DR00, DR02], du nom de ses deux concepteurs VINCENT RIJMEN et JOAN DAEMEN, qui a finalement été choisi en 2001 [NIST01] pour devenir l'AES.

L'AES est un algorithme permettant de chiffrer des blocs de 128 bits (16 octets) avec des clés de taille 128, 192 ou 256 bits. Nous choisissons de décrire ici l'AES dans sa version à clé de 128

[2] La *diffusion* et la *confusion* ont été identifiés par CLAUDE SHANNON [Sha49] comme des propriétés indispensables de toute fonction de chiffrement.

[3] En 1988, le *National Bureau of Standard* devient le NIST (en anglais, *National Institute of Standards and Technology*).

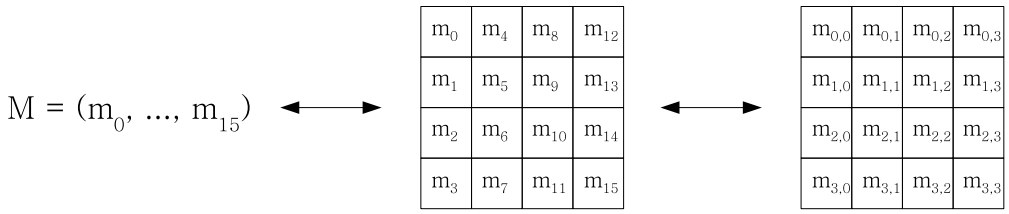


FIG. 1.7 – La représentation matricielle des valeurs intermédiaires de l'AES.

bits qui comprend 10 tours et qui est actuellement la plus utilisée.

Dans l'AES, le texte clair M, les clés de tour K_i, le chiffré C ainsi que les états intermédiaires issus des différentes transformations dont est composé chaque tour sont tous représentés comme des matrices carrées de 4 x 4 octets. La Figure 1.7 illustre, sur le message d'entrée, le principe de cette représentation matricielle.

La Figure 1.8 représente l'algorithme dans sa totalité. Chacun des neuf premiers tours est composé de quatre transformations successives appelées respectivement `SubBytes`, `ShiftRows`, `MixColumns` et `AddRoundKey`. Le dixième et dernier tour de l'AES fait exception et ne comprend pas la fonction `MixColumns`. Par ailleurs, avant d'entamer le premier tour, le message d'entrée est combiné avec une première clé de tour K_0, égale à K, par l'intermédiaire de la fonction `AddRoundKey`. Nous décrivons maintenant ces différentes transformations.

L'addition de la clé de tour, `AddRoundKey`, calcule un XOR composante par composante entre chacun des octets de la matrice représentant l'état intermédiaire courant et chacun de ceux de la matrice représentant la clé de tour. Nous invitons le lecteur intéressé par la façon dont sont dérivées les clés de tour de l'AES à se reporter à la spécification officielle de l'algorithme [NIST01].

La transformation `SubBytes` constitue la partie non linéaire de l'algorithme. Elle substitue à chaque composante de la matrice d'entrée une valeur lue dans une table à 256 entrées de 8 bits. Contrairement au DES, la S-Box S de l'AES est la même pour chacun des octets de la matrice. L'implémentation de la fonction `SubBytes` peut se faire soit par lecture de table classique, soit en profitant de ce que S est définie comme la composée de deux transformations : une pseudo-inversion dans $GF(2^8)$ pour laquelle '0' est son propre inverse, suivie d'une opération affine sur $GF(2)$. Cette dernière façon de calculer `SubBytes` est particulièrement attrayante dans les cas d'implémentations matérielles pour lesquelles il est préférable d'éviter le stockage d'une table de 256 octets.

Les deux dernières transformations de l'AES ont pour rôle d'assurer la *diffusion*. Les différents octets de la sortie de la fonction `SubBytes` sont tout d'abord permutés par la fonction `ShiftRows`. Cette transformation opère indépendamment sur chaque ligne numérotée i ($i = 0, \ldots, 3$) de la matrice et décale ses octets de i positions vers la gauche comme illustré sur la Figure 1.9. Enfin, dans la fonction `MixColumns`, chaque colonne de la matrice représentant la sortie de `ShiftRows` est considérée comme un polynôme sur $GF(2^8)$, lequel est multiplié modulo $(x^4 + 1)$ par le polynôme $c(x) = \text{'03'} \cdot x^3 + \text{'01'} \cdot x^2 + \text{'01'} \cdot x + \text{'02'}$. Cette opération linéaire est équivalente à la multiplication matricielle de chaque colonne par la matrice constante $\mathbf{C}$ suivante :

$$\mathbf{C} = \begin{pmatrix} \text{'02'} & \text{'03'} & \text{'01'} & \text{'01'} \\ \text{'01'} & \text{'02'} & \text{'03'} & \text{'01'} \\ \text{'01'} & \text{'01'} & \text{'02'} & \text{'03'} \\ \text{'03'} & \text{'01'} & \text{'01'} & \text{'02'} \end{pmatrix} .$$

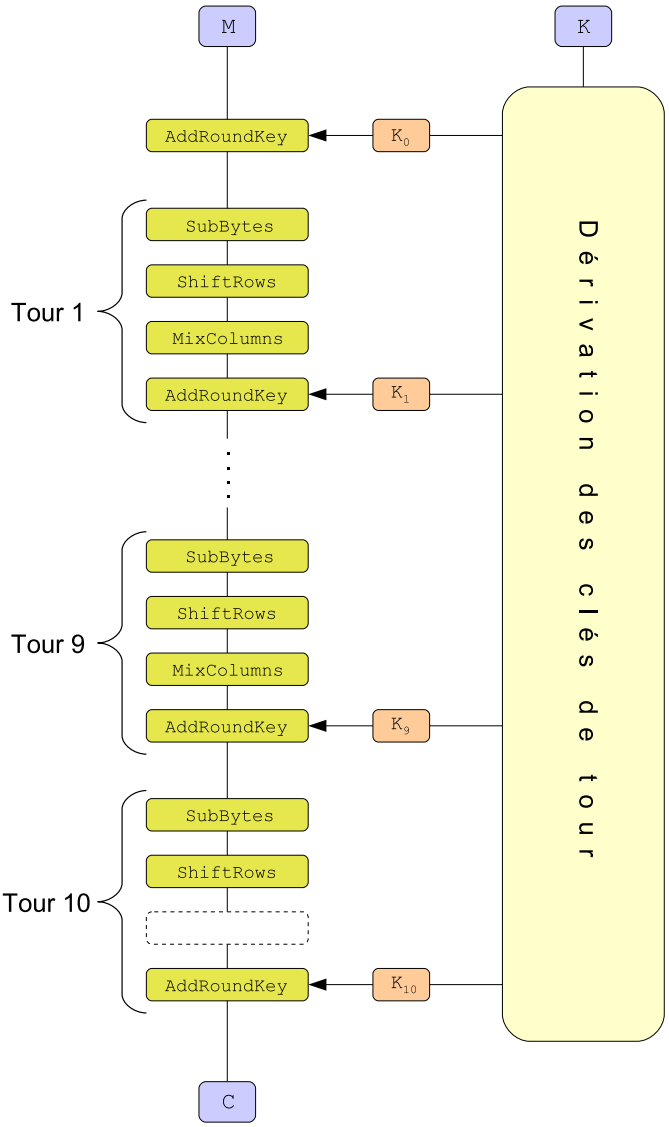


FIG. 1.8 – L'algorithme AES.

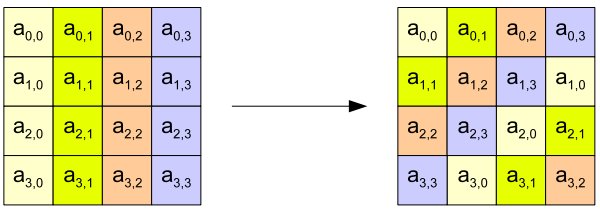


FIG. 1.9 – La transformation `ShiftRows`.

1.2.2 Cryptographie asymétrique

D'un point de vue pratique, la cryptographie symétrique présente un inconvénient majeur lié à la gestion des clés utilisées. Pour que deux interlocuteurs puissent communiquer de manière confidentielle, il leur est nécessaire de convenir au préalable d'une valeur secrète de clé. Cette phase d'échange de secret semble problématique puisqu'elle nécessite *a priori* de pouvoir disposer d'un moyen sûr de pouvoir se transmettre une information confidentielle (la clé).

En 1976, Whitfield Diffie et Martin Hellman [DH76] ont révolutionné l'usage de la cryptographie en inventant, sans toutefois pouvoir en proposer d'instanciation concrète, le concept de cryptographie *asymétrique*, autrement appelée cryptographie *à clé publique*. Le cryptographie asymétrique fait appel aux permutations à sens unique à trappe. Supposons que Bob veuille pouvoir recevoir une correspondance confidentielle. Il génère une clé K^B qui lui est dédiée. Cette clé est en réalité composée de parties $K^B = (K_p^B, K_s^B)$. La première partie K_p^B est publique et est transmise à Alice pour qu'elle puisse chiffrer à destination de Bob en évaluant la fonction à sens unique sur le message clair. K_p^B est appelée la *clé publique* de Bob même si on remarque que ce n'est pas lui qui l'utilise. La deuxième partie K_s^B doit être maintenue secrète par Bob. C'est sa *clé privée*, qui lui sert de trappe pour inverser la fonction à sens unique et ainsi déchiffrer les messages qui ont été chiffrés à son attention.

Une remarque cruciale pour ce type de crypto-système est que la clé publique de Bob n'est dédiée à aucun émetteur en particulier. C'est à dire que n'importe qui peut utiliser la clé publique de Bob pour lui envoyer des données chiffrées. Cela résout le problème évoqué pour l'échange de secret dans le cas d'un crypto-système symétrique. Il n'est plus nécessaire de disposer d'un canal sécurisé pour échanger la clé de chiffrement, ni de devoir la stocker de manière sûre. Il suffit à Bob d'inscrire sa clé publique dans un répertoire rassemblant les clés publiques de tous les utilisateurs. Il faut toutefois remarquer qu'il peut être difficile de se convaincre qu'une clé publique appartient effectivement à son possesseur supposé. Il se pourrait en effet qu'une personne mal intentionnée publie une clé publique au nom de Bob alors que c'est lui-même qui en possède la contre-partie privée. Il pourra alors déchiffrer toute donnée confidentielle qu'Alice aura cru envoyer à destination de Bob. Pour remédier à ce problème, on fait appel à des *certificats* qui attestent de l'identité du possesseur d'une clé publique. Cela met en œuvre toute une infrastructure de gestion des clés publiques (en anglais, *Public Key Infrastructure*, PKI) que nous ne détaillerons pas. Notons que le problème de gestion des clés publiques a été résolu de manière théorique dès 1984 par Adi Shamir [Sha85] qui a proposé le concept de chiffrement basé sur l'identité (en anglais, *Identity Based Encryption*, IBE). Ce n'est toutefois qu'en 1991 qu'un exemple concret d'IBE a été proposé par Ueli Maurer et Yacov Yacobi [MY91, MY93].

En pratique les crypto-systèmes à clé publique sont notablement moins rapides que les fonctions de chiffrement symétrique. Ils ne sont donc généralement pas utilisés pour chiffrer-déchiffrer de gros documents, mais plutôt pour échanger de manière sécurisée une clé de chiffrement symétrique. Chiffrer cette clé par un algorithme à clé publique ne sera pas pénalisant puisque c'est une donnée de petite taille, et une fois cette clé échangée avec son interlocuteur il sera possible d'utiliser un algorithme de chiffrement par bloc rapide (l'AES par exemple) pour chiffrer un volume important de données.

En 1978, Ronald Rivest, Adi Shamir et Leonard Adleman ont proposé le premier exemple de crypto-systèmes à clé publique [RSA78], appelé RSA du nom de ses inventeurs. La sécurité du RSA repose sur la difficulté supposée du problème de la factorisation d'entiers qui, aujourd'hui, n'a toujours pas été résolu. Plusieurs autres crypto-systèmes à clé publiques sont apparus depuis, certains reposant sur d'autres problèmes difficiles comme par exemple le problème du logarithme discret dans le groupe multiplicatif défini modulo un premier [ElG84,

ElG85, NIST00], ou dans des groupes définis par des courbes elliptiques [Mil86, Kob87]. Nous ne détaillerons ici que le fonctionnement du RSA qui est de très loin le plus couramment utilisé.

Le crypto-système RSA

Le crypto-système RSA utilise comme fonction à sens unique l'exponentiation dans les entiers modulo un nombre composé N. La trappe permettant le calcul de racines modulo N est la connaissance de la factorisation de ce module. La sécurité du RSA repose donc sur la difficulté supposée du problème de la factorisation d'entiers.

La génération d'une clé RSA se fait de la manière suivante. Étant donné un entier e appelé *exposant public*, on génère aléatoirement deux grands entiers premiers p et q tels que e soit premier avec $\varphi(N) = (p-1)(q-1)$, où $\varphi(\cdot)$ est la fonction indicatrice d'Euler. Le produit $N = pq$ est appelé le *module*, et forme avec e la partie publique $K_p = (N, e)$ de la clé. La partie privée K_s de la clé est l'*exposant privé* d, égal à l'inverse de e modulo $\varphi(N)$.

Pour chiffrer un message $0 \leqslant m < N$, Alice calcule :

$$c = m^e \bmod N$$

et l'envoie à Bob qui peut déchiffrer c en calculant $m = c^d \bmod N$. On retrouve bien le message d'origine car

$$\begin{aligned} c^d &\equiv m^{ed} \pmod N \\ &\equiv m^{1+k\varphi(N)} \pmod N \\ &\equiv m \pmod N \end{aligned} \tag{1.1}$$

d'après le théorème d'Euler qui établit que $\varphi(N)$ est un multiple de l'ordre de tout élément de $(\mathbb{Z}/N\mathbb{Z})^\star$.[4]

Le crypto-système RSA permet également de produire des signatures numériques de documents. Cette utilisation est duale de celle du chiffrement. Lorsqu'il veut signer un document m, Bob utilise sa clé privée pour générer la signature :

$$s = m^d \bmod N\ .$$

Quiconque souhaite vérifier l'authenticité d'une signature s supposée produite par Bob sur un document m utilisera la clé publique de Bob pour inverser la fonction de signature. Il est alors possible de calculer $m' = s^e \bmod N$ et vérifier que m' est bien égal à m. Dans la pratique, on évite de signer directement la valeur du message m. On utilise au contraire une fonction de hachage $\mathcal{H}$ pour calculer une empreinte $h = \mathcal{H}(m)$ du document à signer. C'est cette empreinte h qui sera signée par Bob. L'inversion de la signature fournira une valeur h' que le vérifieur comparera à $\mathcal{H}(m)$. Cet emploi d'une fonction de hachage permet d'éviter de calculer la fonction RSA sur de gros documents. Elle est généralement utilisée conjointement avec un bon schéma de padding (par exemple RSA-OAEP ou RSA-PSS), pour contrer la malléabilité de l'exponentiation modulaire.

Pour rendre impossible en pratique la factorisation du module N, les deux premiers p et q doivent être suffisamment grands. De nos jours, on utilise des modules de taille au moins égale à 1024 bits, parfois même égale à 2048 bits.

[4] En la considérant séparément modulo p et modulo q, on peut prouver que l'Equation (1.1) est également vérifiée dans le cas où $m \notin (\mathbb{Z}/N\mathbb{Z})^\star$.

Algorithme 1.1 Méthode basique d'exponentiation modulaire

Entrée : $x, d = (d_{k-1}, \ldots, d_0)_2, N$
Sortie : $y = x^d \bmod N$

1. $y \leftarrow 1$
2. **pour** $i \leftarrow k-1$ à 0
3. $\quad y \leftarrow y \cdot y \bmod N$
4. $\quad$ **si** $(d_i = 1)$ **alors**
5. $\quad\quad y \leftarrow y \cdot x \bmod N$
6. $\quad$ **fin si**
7. **fin pour**
8. **retourne** y

FIG. 1.10 – Exponentiation par la méthode *élever au carré et multiplier*.

L'exponentiation modulaire sur des grands entiers est une opération coûteuse en temps de calcul. Parmi les nombreuses méthodes possibles, nous présentons à la Figure 1.10 un algorithme qui réalise l'opération $y = x^d \bmod N$ par la méthode de base dite *élever au carré et multiplier*. Notant k la taille en bit de l'exposant, le nombre moyen de multiplications modulaires est ici de $1,5k$. Plus généralement, quelle que soit la méthode utilisée, au moins k multiplications sont nécessaires pour calculer une exponentiation. Par ailleurs, une multiplication modulaire nécessite habituellement $\mathcal{O}(k^2)$ opérations élémentaires au niveau bit[5], et la complexité de l'exponentiation est donc en $\mathcal{O}(k^3)$.

Deux voies sont possibles pour réduire la durée de l'exponentiation. La première est de réduire le nombre de multiplications en choisissant un petit exposant. Cela est possible dans le cas de l'exponentiation publique pour laquelle la valeur classique $(2^{16}+1)$ de l'exposant e permet de limiter considérablement la durée d'un chiffrement ou d'une vérification de signature.

Pour ce qui concerne l'exponentiation privée utilisée pour le déchiffrement ou la signature, JEAN-JACQUES QUISQUATER et CHANTAL COUVREUR [QC82] ont proposé une technique astucieuse basée sur le *théorème chinois des restes* (en anglais, *Chinese Remainder Theorem*, CRT). Cette technique est appelée *mode CRT* par opposition au *mode standard*. Pour des modules équilibrés, la clé privée d est alors remplacée par cinq éléments de taille deux fois plus petite, p, q, $d \bmod (p-1)$, $d \bmod (q-1)$ et $p^{-1} \bmod q$, qui vont permettre de remplacer le calcul d'une exponentiation modulaire sur des opérandes de taille k par celui de deux exponentiations sur des opérandes de taille $k/2$.

Pour calculer une signature $s = m^d \bmod N$ par cette technique, on commence par évaluer s indépendamment modulo p et modulo q :

$$\begin{aligned} s_p &= s^d \bmod p &&= (s \bmod p)^{d \bmod (p-1)} \bmod p \\ s_q &= s^d \bmod q &&= (s \bmod q)^{d \bmod (q-1)} \bmod q \,. \end{aligned}$$

D'après le théorème chinois des restes, il est alors possible de retrouver s en combinant s_p

[5] Il existe des algorithmes de multiplication avec une meilleure complexité asymptotique, tels que les méthodes de KARATSUBA ($\mathcal{O}(k^{\log(3)/\log(2)})$) ou de SCHÖNHAGE et STRASSEN ($\mathcal{O}(k \log k \log\log k)$), mais leur utilisation n'est pas rentable pour des nombres de taille cryptographique.

et s_q. Cela peut se faire par exemple en utilisant la méthode de Garner :

$$s = s_p + p \cdot \big((s_q - s_p) \cdot p^{-1} \bmod q\big) \ .$$

En raison de la complexité cubique de l'exponentiation, et dans le cas où la durée de la recombinaison est négligeable devant celle des calculs de s_p et s_q, la signature est quatre fois plus rapide en mode CRT qu'en mode standard.

1.3 L'analyse des canaux auxiliaires

Il est très important que les fonctions cryptographiques sur lesquelles reposent les fonctions sécuritaires d'un système soient solides. Elles doivent démontrer un niveau de sécurité théorique ne laissant place à aucune attaque réalisable en un temps raisonnable qu'un adversaire pourrait vouloir mener en analysant les chiffrés ou les paires clair-chiffré. Il est également important d'être vigilant quant à l'utilisation de ces fonctions pour éviter certains pièges. Par exemple l'emploi du RSA doit impérativement faire appel à une bonne fonction de padding pour contrer la malléabilité de l'exponentiation modulaire.

Il est apparu récemment que les précautions prises quant à la fonction mathématique utilisée et à son emploi, quoiqu'évidemment nécessaires, ne suffisent pas à garantir, *concrètement*, la sécurité d'un système. La mise en œuvre de la cryptographie requiert une implémentation *physique* qui produira nécessairement des grandeurs observables d'une autre nature que les entrées et les sorties de la fonction mathématique.

C'est l'exploitation de ces grandeurs physiques que l'on nomme l'*analyse des canaux auxiliaires* (en anglais, *Side Channel Analysis*, SCA). Nécessitant une interaction avec le dispositif cryptographique (ou au moins son observation), les attaques permises par ce type d'analyse concernent donc essentiellement les crypto-systèmes embarqués, par exemple dans les cartes à puce. Le premier exemple publié d'une analyse de canal auxiliaire exploitait le temps d'exécution d'une commande [Koc96], mais d'autres grandeurs physiques ont rapidement été analysées par la suite comme la consommation de courant dès 1998 [KJJ98], ou le rayonnement électromagnétique[6] dès 2001 [GMO01]. Il est également possible d'envisager d'autres types de fuite d'information. Il a ainsi été présenté à Ches '07 un poster proposant d'exploiter la variation de température externe d'un composant électronique en fonction de son activité, démonstration pratique à l'appui.

1.3.1 Les attaques

L'analyse du temps d'exécution

Paul Kocher [Koc96] a donné en 1996 un premier exemple d'analyse de canal auxiliaire en proposant d'exploiter des variations de la durée de certaines opérations, comme par exemple une multiplication modulaire, en fonction des valeurs de ses opérandes. Il montre qu'il est alors possible de retrouver les clés utilisées dans différents crypto-systèmes à clé publique basés sur l'exponentiation modulaire.

Les attaques par analyse du temps d'exécution n'ont pas donné lieu par la suite à beaucoup d'autres publications (voir malgré tout [HH99, DKL+00]) car il est apparu qu'il était souvent assez facile d'assurer qu'une implémentation logicielle sur carte à puce s'exécute en temps constant.

[6] L'information apportée par le rayonnement électromagnétique est parfois de nature différente et plus riche que celle contenue dans la consommation de courant. Cependant les techniques d'analyse et les contre-mesures vis-à-vis de ces techniques sont essentiellement les mêmes.

Néanmoins, un regain d'intérêt est apparu pour une variation de ce type d'attaque qui concerne l'exploitation (déjà entrevue comme possible dans [Koc96]) de différences de temps d'accès à des données présentes ou non en mémoire cache.

Notons que les analyses du temps d'exécution ne se limitent pas aux algorithmes cryptographiques mais peuvent aussi concerner plus généralement toute routine sensible d'un système d'exploitation de carte à puce. Il existe par exemple une commande qui permet de comparer le *code personnel d'identification* (en anglais, *Personal Identification Number*, PIN) stocké dans la carte avec la valeur présentée par l'utilisateur. Il est évident qu'une telle comparaison pourrait être la cible d'une attaque par mesure de temps si la durée de cette commande devait être plus ou moins longue en fonction de la valeur du PIN.

L'analyse simple du courant

L'*analyse simple du courant* (en anglais, *Simple Power Analysis*, SPA) consiste à mesurer et à observer la consommation de courant d'une carte à puce. Cette consommation est proportionnelle à la tension observée aux bornes d'une résistance connectée en série à l'alimentation de la carte. La tension est mesurée à l'aide d'un oscilloscope dont on peut déclencher l'acquisition sur des évènements liés à la communication entre le lecteur et la carte (envoi de la commande par exemple).

La consommation de courant d'un micro-processeur résulte pour l'essentiel de la somme des contributions des consommations de ses différentes portes. Cette consommation reflète donc l'activité interne du composant et il n'est pas étonnant qu'elle dépende aussi bien des instructions exécutées que des données manipulées.

La Figure 1.11 montre un exemple dans lequel deux traces de courant révèlent l'activité électrique lors d'une portion de calcul d'un DES[7]. On note que les deux traces se superposent presque parfaitement jusqu'à ce qu'elles divergent franchement sur la partie droite de la figure. À cet instant l'un des deux processus a exécuté une instruction supplémentaire. Si la présence de cette instruction supplémentaire, qui se traduit par des pics vers le bas facilement reconnaissables, dépend d'une valeur liée à la clé (ce qui est le cas ici sur cette implémentation non protégée), alors il est possible d'envisager une attaque basée sur cette observation.

La dépendance de la consommation vis-à-vis des instructions exécutées peut être exploitée d'autres manières. Par exemple, lorsqu'une exponentiation privée RSA est implémentée avec la méthode basique *élever au carré et multiplier* décrite à la Figure 1.10, il peut être possible de déterminer les bits de l'exposant privé si la multiplication et l'élévation au carré consomment différemment. Un autre exemple est donné sur la Figure 1.12 qui montre comment la consommation de courant peut révéler la structure d'un algorithme et de son implémentation. On y distingue les différentes opérations effectuées pendant un tour du DES. Cette opportunité de distinguer par SPA à quels instants se produisent des évènements précis peut être mise à profit par l'adversaire pour améliorer l'efficacité de nombreux types d'attaques physiques.

Il est possible de remarquer sur la Figure 1.11 qu'il existe des instants où les deux traces ne se superposent pas parfaitement, même durant la phase pendant laquelle les instructions exécutées sont identiques. Ceci est révélateur de la dépendance entre la consommation de courant et la valeur des données traitées.

La Figure 1.13 illustre encore mieux cette dépendance. Elle représente les consommations correspondant à la manipulation d'une variable prenant six valeurs différentes. Cette portion

[7] Pour l'ensemble des figures de cette section, l'axe des abscisses représente le temps et l'axe de ordonnées représente l'amplitude de la consommation de courant.

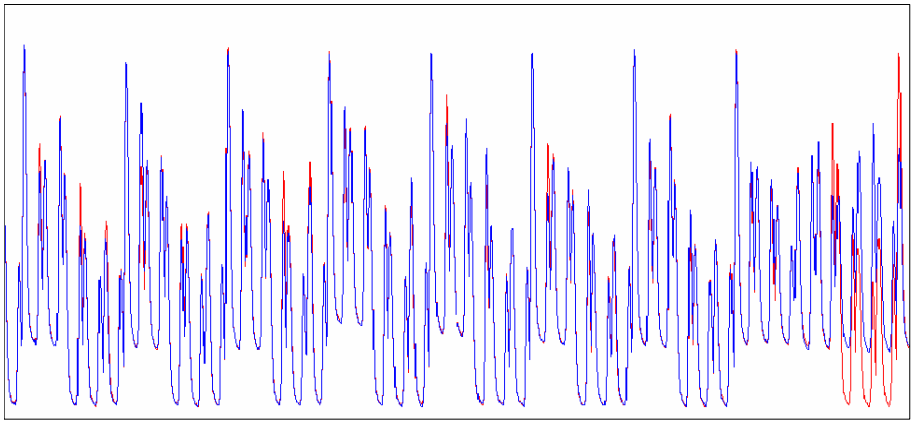


FIG. 1.11 – La consommation de courant dépend des instructions exécutées.

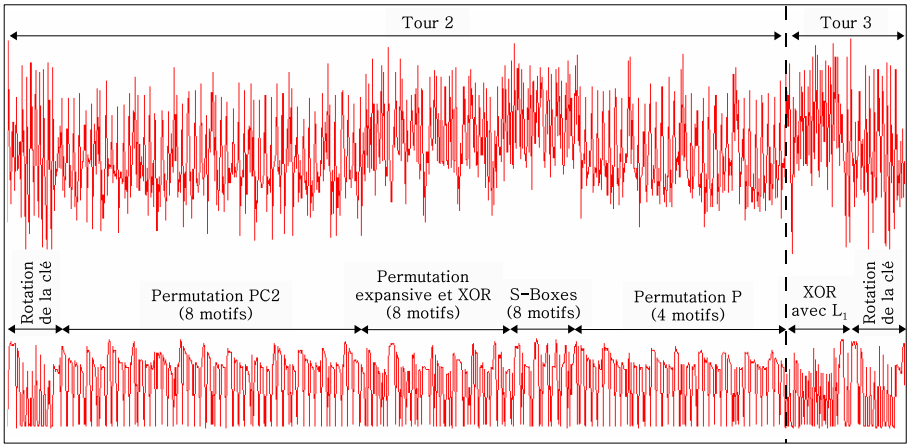


FIG. 1.12 – La consommation de courant révèle la structure des algorithmes.

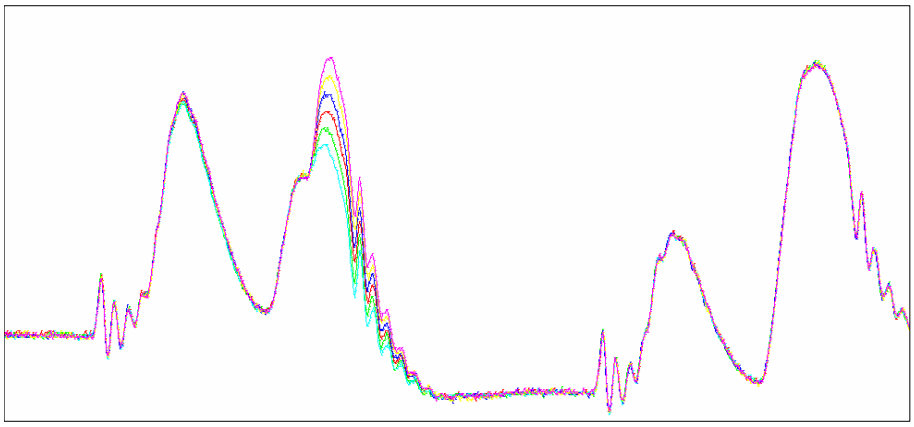


FIG. 1.13 – La consommation de courant dépend des données manipulées.

de deux cycles d'horloge présente un instant de signature auquel les six consommations se distinguent bien les unes des autres, de manière assez régulièrement espacées. Il est possible d'exploiter la dépendance de la consommation en fonction des données notamment dans une classe d'attaques dérivées de la SPA et appelées *attaques par dictionnaire* (en anglais, *template attacks*) [CRR03, RO04, APSQ06, GLP06].

Les analyses statistiques du courant

L'analyse simple du courant exploite des différences de consommation visibles sur une ou quelques traces de courant, chacune pouvant éventuellement être produite par moyenne de plusieurs exécutions avec les mêmes données d'entrée afin d'atténuer le bruit de mesure. C'est de la comparaison de ces courbes qu'est obtenue de l'information sur un secret manipulé par la carte.

Les analyses statistiques du courant effectuent un traitement sur un grand nombre de traces obtenues en faisant varier le message en entrée de l'algorithme. Ce traitement statistique permet essentiellement de réaliser un test d'hypothèse sur une petite portion de la clé, appelée *sous-clé* dans le cas des algorithmes de chiffrement par bloc. Ce test d'hypothèse s'appuie sur la consommation de courant produite par la manipulation d'une donnée intermédiaire de l'algorithme qui ne dépend que du message d'entrée (ou du chiffré de sortie) et de la valeur de la sous-clé. Nous présentons maintenant plusieurs techniques d'analyse statistique du courant.

Historiquement, la première de ces méthodes a été introduite dès 1998 par Paul Kocher, Joshua Jaffe et Benjamin Jun [KJJ98, KJJ99] et s'appelle l'*analyse différentielle du courant* (en anglais, *Differential Power Analysis*, DPA). Elle considère un bit arbitraire d'une valeur intermédiaire de l'algorithme ne dépendant que du message d'entrée et de la valeur d'une sous-clé. Un tel bit est appelé *bit de sélection* ou encore *bit cible*. Pour chaque supposition sur la valeur de la sous-clé, il est possible d'effectuer une partition des traces de courant en deux ensembles : l'un des ensembles contient les traces pour lesquelles la valeur du bit de sélection vaut 0, et l'autre contient les traces pour lesquelles il vaut 1.

Il est important de souligner que la valeur calculée pour le bit de sélection dépend à la fois du message d'entrée et de la supposition faite sur la valeur de la sous-clé. C'est une prédiction de ce qu'est la valeur réelle de ce bit. Cette prédiction est systématiquement correcte pour la bonne supposition de la sous-clé, et est supposée correcte par chance une fois sur deux lorsque la valeur supposée de la sous-clé n'est pas la bonne.

Une fois l'ensemble des courbes de courant ainsi partitionné en deux groupes, on calcule la courbe de courant moyenne de chacun des groupes, et on les soustrait l'une à l'autre. Il en résulte une *courbe de DPA* associée à chacune des suppositions sur la sous-clé.

Si la valeur supposée de la sous-clé est correcte, alors le bit de sélection reflète exactement sa valeur dans la carte au moment de l'exécution et la courbe de DPA montre une différence de courant significative à chaque instant où ce bit a été manipulé. Cela se traduit par un "pic" à chacun de ces instants. En revanche, si la supposition est incorrecte, les courbes de chaque paquet sont sensément indépendantes de ce qui s'est réellement passé dans la carte pour ce bit, et les courbes de DPA qui en résultent sont présumées être plates à un bruit près. La Figure 1.14 présente à mi-hauteur une courbe de DPA sur le début d'un DES qui montre cinq pics prononcés pour la bonne supposition de sous-clé. Les courbes du haut et du bas correspondent à la consommation du DES.

Il est ainsi possible d'identifier la bonne valeur de la sous-clé comme étant celle qui produit le pic le plus haut parmi toutes les courbes de DPA. En procédant de la sorte successivement pour toutes les sous-clés, on retrouve progressivement la majeure partie ou la totalité de la clé.

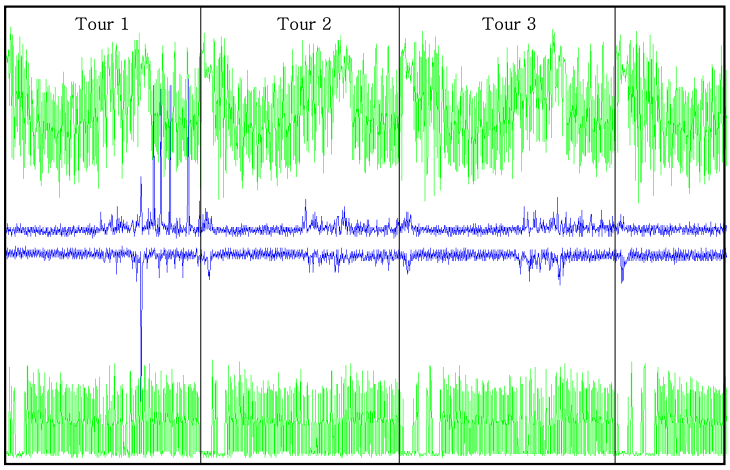


FIG. 1.14 – Une courbe de DPA sur le début du DES.

Un avantage pratique de la DPA est de ne pas nécessiter d'hypothèse forte sur le modèle de consommation en fonction des données. Il est simplement suffisant que la consommation de courant lorsqu'un mot de donnée est manipulé soit différente en moyenne selon que l'un des bits de ce mot vaille systématiquement 0 ou systématiquement 1.

En revanche la DPA présente l'inconvénient d'être sujette au problème dit des *pics fantômes*. Contrairement à ce que voudrait la théorie présentée ci-dessus, il peut arriver que certains pics soient présents sur des courbes de DPA correspondant à certaines suppositions incorrectes de la sous-clé. Ces pics fantômes, dont l'amplitude peut tout à fait rivaliser avec les "vrais" pics (ceux apparaissant sur le courbe de DPA pour la bonne sous-clé), constituent bien évidemment un problème pour l'identification correcte de la valeur de la sous-clé. Des courbes de DPA sur l'AES pour trois suppositions différentes de sous-clés sont présentées sur la Figure 1.15. Pour chacune, le pic le plus prononcé est dirigé vers le bas. Ces trois pics ont sensiblement la même amplitude. On comprend bien, sur cet exemple, qu'il est difficile de déterminer si l'un d'eux correspond à la bonne valeur de la sous-clé.

La raison de l'existence de ces pics fantômes tient dans ce que l'approche adoptée conjugue deux erreurs de modélisation. La première est qu'il est faux de considérer que la répartition des courbes de courant dans chaque paquet pour une supposition de sous-clé incorrecte est indépendante de celle correspondant à la bonne supposition. Il peut résulter de cette approximation que certaines mauvaises partitions ressemblent malgré tout fortement à la bonne. La deuxième erreur de modélisation consiste à ignorer la contribution, dans la consommation de courant, des bits autres que le bit de sélection et qui se trouvent pourtant dans le même mot machine. Par exemple, sur une architecture à 8 bits, les contributions des sept bits accompagnant le bit de sélection dans l'octet manipulé sont considérées identiques dans chacun des deux paquets de courbes. Cette hypothèse hasardeuse n'est pas nécessairement vérifiée en pratique, et la contribution conjuguée de ces sept bits sur la consommation peut tout à fait contrebalancer l'influence du bit de sélection. Nous invitons le lecteur intéressé par cette discussion sur les limitations de la DPA à se reporter au Chapitre 3 ou à l'excellente étude de CÉCILE CANOVAS et JESSY CLÉDIÈRE [CC05][8] qui présentent une analyse plus détaillée de ce sujet.

[8] Ce travail présente une étude systématique et détaillée des pics de DPA ou de CPA obtenus sur le DES sous les modèles de consommation usuels de poids et de distance de Hamming. Ils ignorent cependant la contribution

Nous présentons maintenant deux alternatives à la DPA, chacune issue d'une approche visant à éviter ses hypothèses trop simplificatrices et à prendre en considération toute l'information dont dispose l'attaquant qui se révèle être pertinente pour la fonction de consommation à l'instant de formation du pic.

La première méthode est appelée *analyse du courant par corrélation* (en anglais, *Correlation Power Analysis*, CPA). Elle a été proposée par Éric Brier, Francis Olivier et l'auteur [BCO03, BCO04] (voir également Chapitre 3) et suppose que l'attaquant a préalablement identifié un modèle de consommation. La CPA repose sur deux idées essentielles. La première est que, connaissant le message d'entrée et une valeur supposée de la sous-clé, l'attaquant dispose de toute l'information nécessaire pour prédire *l'intégralité* du mot machine pertinent pour l'attaque[9]. La deuxième est que, disposant d'un modèle de consommation, l'attaquant peut transformer sa prédiction de la donnée manipulée en prédiction de la consommation occasionnée par la manipulation de cette donnée. Il est donc possible, pour chaque supposition de la sous-clé, d'établir une série de prédictions (une pour chaque courbe acquise) de la consommation de courant à l'instant pertinent pour l'attaque. Au bruit de mesure et à une éventuelle imprécision de modèle près, cette série reflète nécessairement les consommations mesurées dans le cas où la supposition sur la sous-clé est correcte. Un calcul du coefficient de corrélation linéaire entre les consommations mesurées et les consommations prédites révélera alors facilement l'exactitude de la supposition sur la sous-clé. En effet, si une erreur est commise sur cette sous-clé, le fait d'avoir tenu compte de tous les bits du mot machine pour fonder la prédiction de la consommation rend alors beaucoup plus improbable que dans le cas de la DPA l'émergence d'un pic fantôme de corrélation.

Nous mentionnons maintenant deux modèles classiques de consommation utiles pour la CPA. Ces deux modèles sont basés sur le fait que les plus grandes variations de la consommation de courant en fonction d'une donnée se produisent lorsque cette donnée est présentée sur le bus externe de la CPU. À cet instant, chaque ligne de bus pour laquelle se produit une transition de 0 à 1 ou de 1 à 0 consomme une certaine énergie nécessaire pour réaliser ce basculement d'état. En supposant que les contributions des bits sont additives, et que l'énergie requise est la même quel que soit le bit considéré et quel que soit le sens de la transition, on obtient une consommation globale qui est une fonction linéaire de la distance de Hamming entre la valeur de la donnée considérée et celle préalablement présente sur le bus et appelée *état de référence*. Sur certains composants, le bus est dit *préchargé*, c'est-à-dire que l'état de référence est remis à zéro (ou à son complément) avant chaque écriture. Dans ce cas de figure, l'état de référence ne joue aucun rôle et le modèle de consommation se réduit à une fonction linéaire en le poids de Hamming de la donnée manipulée.

Ces deux modèles de consommation sont les plus couramment adoptés car ils s'ajustent souvent assez bien aux consommations effectivement mesurées. La CPA n'est cependant pas limitée à ces modèles, et est facilement adaptable à des modèles plus élaborés prenant par exemple en considération des contributions différenciées pour chacun des bits, voire même asymétriques en fonction du sens de la transition.

Notre expérience de la CPA sur divers composants montre que cette méthode réduit l'amplitude des pics fantômes presque à néant dans la plupart des cas[10], et permet ainsi de prendre

des 4 bits adjacents à la sortie de la S-Box dans le cas où ceux-ci sont utilisés pour coder la sortie d'une autre S-Box.

[9] Ceci est à nuancer quelque peu dans le cas du DES où l'attaquant peut avoir à faire une supposition sur la façon dont est implémentée une éventuelle "compression" des S-Box.

[10] On notera toutefois certains cas rares mis en évidence dans [CC05] d'indistinguabilité parfaite entre deux valeurs supposées de la sous-clé.

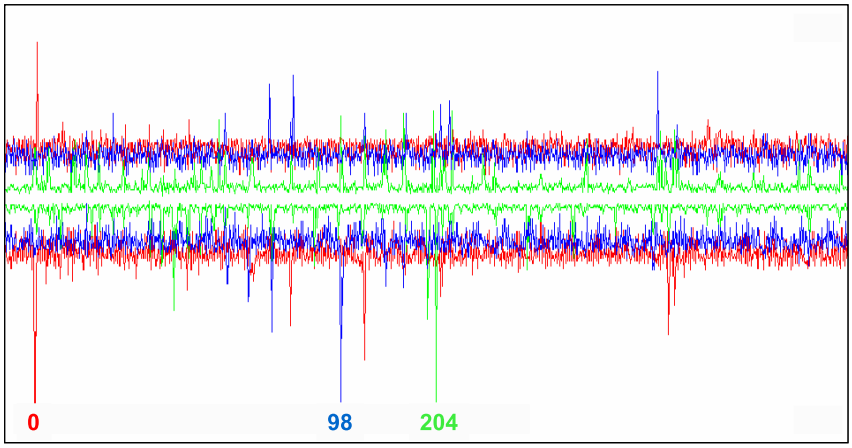


FIG. 1.15 – Courbes de DPA sur l'AES pour différentes suppositions de sous-clé.

une décision correcte quant à la valeur de la sous-clé avec considérablement moins de courbes que la DPA.

Une autre méthode intéressante pour palier aux inconvénients de la DPA a été proposée par RÉGIS BÉVAN et ERIK KNUDSEN [BK03]. Tout comme pour la CPA cette méthode présuppose un modèle de consommation. La prédiction de l'intégralité de la valeur intermédiaire est également combinée avec le modèle pour obtenir une prédiction de la consommation réelle. Sous l'hypothèse d'un bruit de mesure gaussien, les auteurs identifient la valeur correcte de la sous-clé comme étant celle maximisant la vraisemblance des consommations mesurées. Découverte avant la CPA, il est très probable que cette méthode procure une insensibilité aux pics fantômes équivalente. En revanche, et contrairement à la CPA, nous ignorons comment cette technique se comporte dans le cas où une partie des bits du mot machine manipulé n'ont pas pu être prédits.

1.3.2 Les contre-mesures

L'analyse du temps d'exécution

Pour parer l'analyse du temps d'exécution, la contre-mesure la plus évidente consiste à implémenter les fonctions cryptographiques et autres routines sensibles *en temps constant*, c'est-à-dire de manière telle que le temps d'exécution de la fonction ne varie pas, ou tout au moins que sa variation ne dépende en rien de la valeur du secret. C'est presque toujours possible, mais il existe des cas difficiles où la source de variation temporelle n'est pas inhérente à l'implémentation logicielle mais provient par exemple du fonctionnement interne d'un crypto-processeur ou encore d'un mécanisme de mémoire cache.

L'analyse simple du courant

Une bonne manière de se protéger contre les attaques SPA permettant de distinguer, soit entre l'exécution de deux instructions différentes, soit entre la présence et l'absence d'une instruction, est de produire une implémentation *en code constant*. Il faut en effet veiller à ce que la suite des instructions exécutées soit la plus régulière possible, ou tout au moins qu'elle non plus ne dépende pas de la valeur du secret.

Citons par exemple deux méthodes utilisables en cryptographie asymétrique. La première concerne la multiplication scalaire d'un point sur une courbe elliptique. Cette opération, analogue à l'exponentiation modulaire du RSA, est sujette au même type de SPA s'il est possible de distinguer entre l'opération de doublement et celle d'addition de points (les analogues respectifs du carré et de la multiplication pour l'exponentiation). Les formules pour calculer ces opérations sont de natures différentes, mais divers auteurs ont réussi à proposer des formules unifiées permettant de calculer le doublement et l'addition avec la même séquence d'instruction.

La deuxième méthode, proposée par Benoît Chevallier-Mames, Mathieu Ciet et Marc Joye [CCJ04], est plus fondamentale et permet de transformer une séquence d'instructions *a priori* irrégulière en la répétition d'un motif unique. Sous l'hypothèse que deux exécutions de ce motif sont indistingables par SPA, cette *atomicité* de l'implémentation permet de "gommer" de la trace de courant les détails qu'aurait pu exploiter un attaquant pour retrouver la clé. Les auteurs montrent comment appliquer efficacement leur technique à l'exponentiation modulaire et à la multiplication scalaire.

S'il est relativement simple d'éviter qu'un attaquant puisse mener une SPA au niveau *instruction*, il n'est pas aisé de se protéger contre une SPA au niveau *donnée*. Sous l'hypothèse forte qu'un attaquant est capable de "lire" la valeur d'une donnée quelconque, ou même seulement son poids de Hamming, à l'aide d'une seule trace de courant, on peut affirmer qu'il est impossible de se protéger de manière logicielle contre ce type de SPA. Nous comprenons donc qu'il est nécessaire de rendre impossible, ou au moins très difficile, la détermination de la valeur d'une donnée à partir de l'analyse de la consommation de courant. Les techniques utilisées pour cela font appel à des dispositifs au niveau matériel pouvant avoir comme objectif de :

1. rendre la consommation la plus indépendante possible des données manipulées par une conception équilibrée au niveau des portes logiques [THH+05, PM05, BGLT06, CZ06],
2. brouiller la consommation par l'ajout d'un bruit aléatoire,
3. désynchroniser l'exécution du processus.

La première technique est de loin la plus prometteuse car elle traite le problème à la base en visant à supprimer la fuite d'information. Elle permet donc également de se protéger contre la DPA.

L'ajout d'un bruit aléatoire n'est suffisant que s'il est accompagné d'une autre contre-mesure, particulièrement dans le cas où il est possible de moyenner les traces de courant à données constantes. De plus, son efficacité contre la DPA est négligeable.

Enfin, différents dispositifs ont été proposés pour assurer la désynchronisation du processus. Une méthode appelée *interruption aléatoire de processus* (en anglais, *Random Process Interrupt*, RPI), consiste à interrompre l'exécution du code à des instants aléatoires. Un cycle d'horloge inutile est alors inséré pour décaler d'autant la partie utile de l'exécution. Le haut de la Figure 1.16 montre un exemple de deux traces de courant dont l'une est ainsi décalée d'un cycle par l'apparition d'un RPI au niveau du repère horizontal. Il est néanmoins envisageable de pouvoir traiter les courbes contenant des RPI afin de les éliminer. Le bas de la Figure 1.16 illustre le résultat alors obtenu. Une autre façon de désynchroniser le processus consiste à utiliser une horloge interne variable basée sur un oscillateur instable. Le haut de la Figure 1.17 montre un exemple de l'activation d'un tel mécanisme. Sur la partie gauche les courbes sont superposées mais elles divergent dès que l'horloge interne est utilisée. Cette contre-mesure est assez efficace, y compris contre la DPA, même s'il est facilement possible de rendre deux courbes localement synchrones, comme illustré sur la figure du bas, à l'aide d'un traitement à base de reconnaissance de forme.

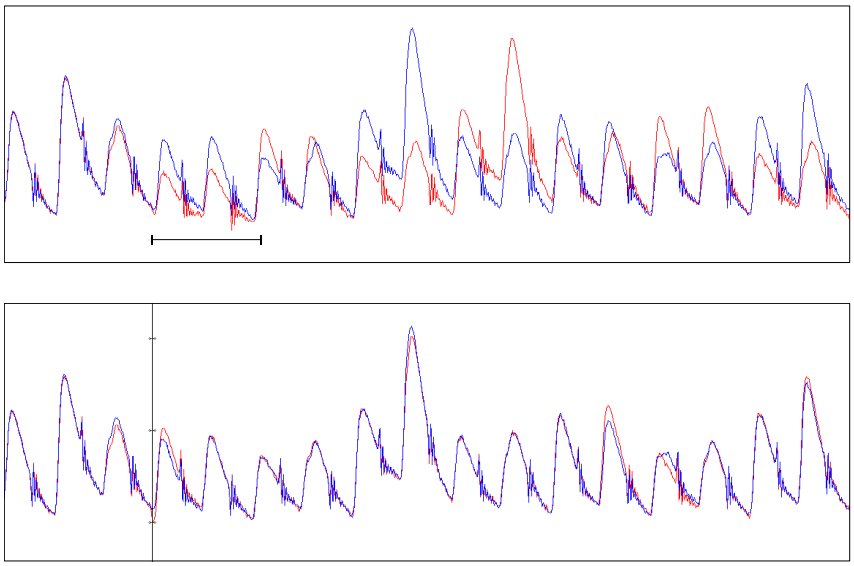


FIG. 1.16 – Exemple d'interruption aléatoire de processus.

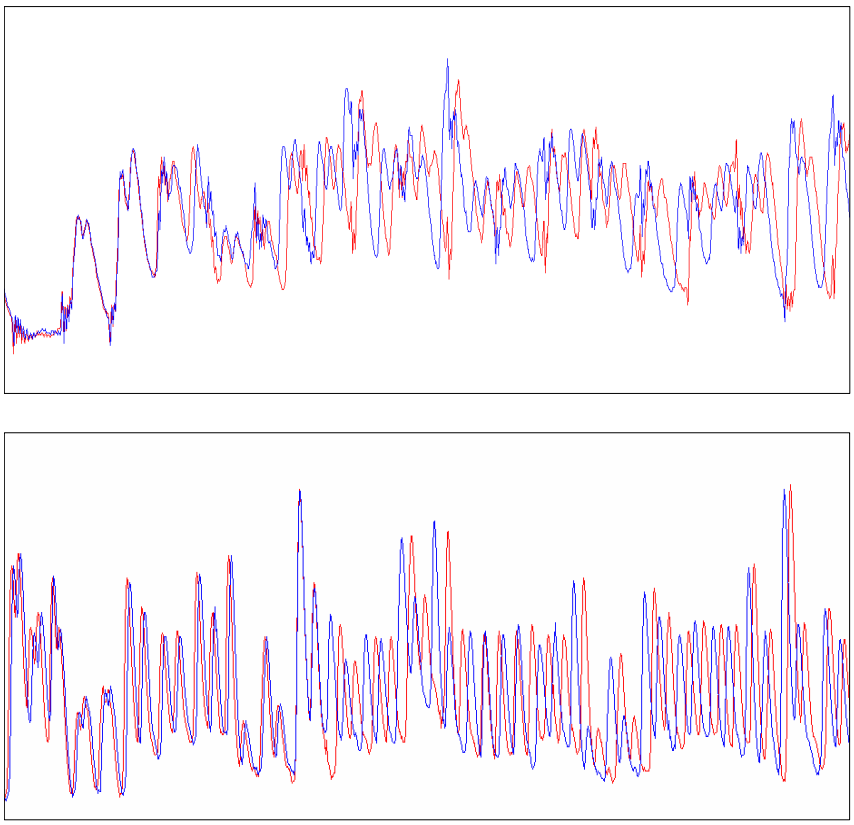


FIG. 1.17 – Exemple de désynchronisation provoquée par une horloge instable.

Les analyses statistiques du courant

Le succès d'une analyse statistique du courant requiert deux conditions. D'une part il est nécessaire que les processus considérés soient synchrones pour que la manipulation de la donnée intermédiaire prédite par l'attaquant se produise au même instant sur chaque trace, d'autre part il faut que cette donnée prédite soit effectivement manipulée par la carte.

Les protections contre les analyses statistiques du courant vont viser à ce que l'une au moins de ces conditions ne soit pas vérifiée.

Désynchronisation : Le premier type de contre-mesure a donc pour objectif de désynchroniser les différentes exécutions de l'algorithme. Cela est rendu possible, soit grâce à des mécanismes hardware comme les RPI ou les horloges internes instables déjà cités comme exemples de contre-mesures SPA, soit par programmation logicielle. Deux méthodes non exclusives de désynchronisation logicielle sont possibles. La première consiste à insérér des portions de code inutiles à des instants aléatoires. Pour dérouter d'autant plus l'attaquant, ces portions de code seront elles-mêmes aléatoires, et de durée aléatoire. Cette méthode est le pendant logiciel des RPI. La deuxième idée consiste, chaque fois que cela est possible, à exécuter des processus indépendants en ordre aléatoire. Par exemple, on peut calculer les S-Box du DES ou de l'AES dans un ordre quelconque plutôt que dans leur ordre naturel.

Ces protections ne sont pas absolues mais peuvent grandement gêner l'attaquant. Elles seront avantageusement associées à des contre-mesures plus fondamentales comme celle présentées maintenant.

Randomisation des données : Le principe général des contre-mesures fondamentales contre les analyses statistiques est simple. Il s'agit de rendre aléatoire à chaque exécution la valeur intermédiaire prédite par l'attaquant. La vraie valeur manipulée, et donc la consommation qui en résulte, est alors indépendante de celle prédite et l'analyse statistique de la consommation de courant échoue. Bien sûr, il est nécessaire que cette randomisation des valeurs intermédiaires n'empêche pas de pouvoir effectivement calculer au final la fonction cryptographique. Dans le cas des algorithmes asymétriques, cela se réalise généralement de manière naturelle par l'emploi d'un ou de plusieurs masques arithmétiques. Le cas des chiffrements par bloc est plus particulier. Par exemple si une fonction f est XOR-linéaire, il suffit de masquer le message M en entrée par un XOR avec une valeur aléatoire R, de calculer $f(M \oplus R)$, puis de démasquer en sortie pour obtenir $f(M \oplus R) \oplus f(R) = f(M) = C$. Bien entendu, dans la réalité les fonctions ne sont pas entièrement linéaires, mais on peut vouloir procéder de manière analogue si la majeure partie de l'algorithme est linéaire comme c'est le cas pour le DES. Il faut alors traiter de manière particulière les parties non linéaires que sont les S-Box. Pour cela, et à chaque exécution, on doit recalculer de nouvelles tables S^* définies, à partir de S et des valeurs des masques r_e et r_s en entrée et en sortie de la S-Box, par :

$$S^*(x) = S(x \oplus r_e) \oplus r_s \ .$$

Plusieurs contre-mesures de ce type ont été publiées parmi lesquelles citons la méthode de la *duplication* [GP99] proposée par Louis Goubin et Jacques Patarin, et celle du *masque adaptatif* [AG01] découverte par Mehdi-Laurent Akkar et Christophe Giraud. La thèse de doctorat de Christophe Giraud [Gir07] détaille la mise en œuvre du masque adaptatif sur l'AES qui nécessite des précautions particulières par rapport au DES.

Ces différentes contre-mesures protègent contre les attaques statistiques présentées à la sous-section précédente. En théorie, elles ne sont pas efficaces contre des variantes dites *d'ordre*

supérieur de ces techniques. On dit qu'une analyse statistique du courant est *d'ordre n* si elle effectue un traitement statistique sur une fonction de n différents points de chaque trace de courant. Il semble difficile de concevoir une contre-mesure fondamentale qui protège contre les analyses d'ordre supérieur. Par exemple, Kai Schramm et Christof Paar ont publié à Ct-rsa '06 [SP06] une méthode permettant, pour tout n, de protéger l'AES contre les analyses statistiques d'ordre n. Malheureusement, Jean-Sébastien Coron, Emmanuel Prouff et Matthieu Rivain ont démontré à Ches '07 [CPR07] que cette contre-mesure est vulnérable, pour tout $n \geqslant 3$, à une analyse statistique d'ordre 3. La conception d'une contre-mesure efficace contre les analyses statistiques d'ordre supérieur semble être un problème ouvert.

1.4 L'exploitation de fautes

Les analyses de canaux auxiliaires sont des attaques passives. Elles ne requièrent pas de l'attaquant qu'il interfère avec le déroulement du calcul cryptographique autrement qu'en mesurant une grandeur physique laissant fuir de l'information sur l'état interne du processus.

Supposons maintenant qu'un adversaire soit capable d'interférer avec l'exécution d'un programme pour la perturber d'une manière ou d'une autre. Cela ouvre la voie à des attaques actives par *exploitation de fautes* qui tirent parti de l'effet de la perturbation pour inférer de l'information autrement inaccessible.

Un exemple de telles attaques a été décrit pour la première fois en 1996 par Dan Boneh, Richard DeMillo et Richard Lipton [BDL97]. Cette attaque, qui n'était à l'époque que théorique, est aujourd'hui connue sous le nom d'attaque *Bellcore* d'après le nom du laboratoire où travaillaient ses découvreurs.

Les moyens d'injection de faute

On connaît différentes façons de perturber le fonctionnement d'une carte à puce. Parmi les moyens les plus utilisés, on peut citer par exemple :

les glitchs : Ce sont des variations courtes et franches (pour ne pas être détectées par les capteurs de sécurité) de la tension d'alimentation, ou de la fréquence d'horloge, appliquée à la carte. Lorsque la tension est perturbée, le micro-processeur peut mal interpréter, mal exécuter, voire sauter une instruction. Des variations de l'horloge peuvent également provoquer des erreurs de lecture.

l'illumination : Soumettre la surface d'une puce, préalablement mise à nu, à une source lumineuse intense peut avoir pour effet de modifier une donnée (ou une adresse, ou un code d'opération) ou de dérouter le processus. Ce type d'injection de fautes peut être réalisé par un moyen très rudimentaire comme par exemple un flash d'appareil photo, ou beaucoup plus perfectionné et efficace comme dans le cas d'un banc laser. Cette dernière source permet une grande précision aussi bien spectrale, que spatiale (le faisceau laser peut être focalisé sur une petite surface du circuit) ou temporelle (l'impulsion peut atteindre une durée très courte).

la variation de température : Des températures extrêmes peuvent provoquer des modifications aléatoires du contenu de cellules RAM, ou des erreurs de lecture de la mémoire non volatile.

les impulsions magnétiques : Un champ magnétique intense appliqué près de la surface du silicium crée des courants locaux pouvant provoquer des fautes.

Le lecteur intéressé pourra consulter [GT04] où sont donnés plus de détails concernant les types d'injecteurs et où il est discuté de considérations pratiques concernant leurs mises en œuvre.

Les modèles d'effet de la faute

Il existe une diversité des effets de la faute en fonction de l'injecteur utilisé et du composant visé. Les modèles d'effet de faute les plus souvent utilisés pour la description d'attaques sont les suivants :

modification aléatoire : L'effet de la faute est de modifier une donnée de manière aléatoire et, *a priori*, inconnue de l'attaquant. On suppose généralement que la faute modifie intégralement un mot machine, mais certains auteurs font l'hypothèse qu'une faute peut ne modifier qu'un seul bit.

écrasement : Dans ce modèle, la faute modifie une donnée pour lui faire prendre une valeur fixe souvent supposée égale à 0 ou à son complément.

déroutement : Ce type de faute correspond à une modification de la séquence d'instruction normalement exécutée. Dans le cas d'un saut d'instruction, celle-ci est ignorée et le processus continue normalement à l'instruction suivante. Il peut aussi arriver qu'une modification du compteur programme fasse sauter l'exécution à une adresse difficilement prévisible.

Pour chacun de ces modèles, il est important de définir quelle précision temporelle et/ou spatiale l'attaquant est capable d'atteindre. Cela permet notamment de déterminer s'il lui est possible de choisir précisément le mot machine (ou le bit) ou l'instruction qui subit la faute.

Remarquons que le modèle de faute le moins exigeant suppose une modification aléatoire d'un mot machine que l'attaquant n'a pas la possibilité de choisir. Il est particulièrement intéressant de concevoir des attaques reposant sur ce modèle, car elles seront alors parmi les plus simples à réaliser en pratique.

1.4.1 Les attaques

Dans l'ensemble du code programme d'une carte à puce, il existe de nombreuses opérations de routines sensibles qui peuvent être la cible d'injections de fautes. Bien qu'elles offrent alors à l'attaquant autant de possibilités d'obtenir une connaissance ou un accès indu, nous ne décrirons dans la suite que des exemples d'attaques sur des algorithmes cryptographiques.

Les analyses différentielles de fautes

Les *analyses différentielles de fautes* (en anglais, *Differential Fault analysis*, DFA), exploitent l'effet différentiel que produit une faute sur la sortie d'un algorithme. Dans cette catégorie, nous mentionnerons l'attaque *Bellcore* sur l'exponentiation privée RSA calculée en mode CRT, la DFA sur le DES, ainsi que des attaques de type DFA sur l'AES. Par ailleurs, signalons que le Chapitre 7 de ce mémoire présente une description détaillée d'une DFA sur l'algorithme de chiffrement par bloc IDEA.

Attaque *Bellcore* : La première attaque par faute publiée, l'attaque *Bellcore*, concerne la signature RSA calculée en mode CRT. On suppose qu'un attaquant dispose d'une carte à puce de laquelle il a pu obtenir la signature s d'un message m et une signature fautée s' du même message, et où la faute est supposée avoir perturbé le résultat d'une (seule) des deux exponentiations

modulaires. Supposant sans perte de généralité que la faute a perturbé le calcul de $s_p = s^d \bmod p$, la signature fautée s' dont dispose l'attaquant vérifie :

$$\begin{aligned} s' &\not\equiv s \equiv s_p \pmod{p} \\ s' &\equiv s \equiv s_q \pmod{q} . \end{aligned}$$

Il en résulte que $s' - s$ est un multiple de q sans être un multiple de p, ce qui permet de révéler q par un simple calcul de pgcd :

$$q = \text{pgcd}(s' - s, N) .$$

Notons que Marc Joye, Arjen Lenstra et Jean-Jacques Quisquater [JLQ99] ont décrit une variante de cette attaque qui permet de retrouver q en calculant $q = \text{pgcd}(s'^e - m, N)$, c'est à dire sans avoir besoin de solliciter la carte deux fois avec le même message. Cette variante trouve tout son intérêt dans le cas où le calcul de la signature utilise un padding aléatoire avec aléa joint comme par exemple RSA-PFDH.

DFA sur le DES : Eli Biham et Adi Shamir ont introduit le terme DFA en 1997 en présentant plusieurs attaques par fautes sur le DES [BS97]. Nous décrivons ici celle qui est la plus connue et qui repose sur des hypothèses minimales quant à l'effet de la faute et la puissance de l'attaquant.

On suppose qu'un attaquant est capable de provoquer une erreur de calcul n'importe où durant l'avant-dernier tour du DES. Notons que l'attaquant n'a pas l'obligation de localiser sa faute sur ce tour précis de manière déterministe car la structure du DES est ainsi faite qu'il lui sera possible d'identifier *a posteriori* s'il a effectivement perturber le calcul durant ce tour.

Disposant d'un couple (C, C') de deux chiffrements, l'un normal, l'autre fauté, d'un même message M, il est possible d'en déduire $R_{15} = L_{16}$ et $R'_{15} = L'_{16}$ qui sont respectivement les valeurs intacte et fautée en entrée du dernier tour. On peut également calculer $R_{16} \oplus R'_{16}$ qui est égal à la différentielle en sortie du dernier tour puisque l'on suppose que la faute n'a pas perturbé $R_{14} = L_{15}$. Pour chaque S-Box active du dernier tour, c'est à dire dont les entrées sont différentes, il est possible d'éliminer toutes les hypothèses de sous-clé pour lesquelles on n'obtient pas la différentielle de sortie attendue. En répétant cette procédure pour différents couples (C, C'), on détermine progressivement une unique valeur possible pour les 48 bits de la clé du tour 16. Les huit bits de clé encore inconnus se retrouvent par recherche exhaustive.

DFA sur l'AES : Plusieurs analyses différentielles de fautes sur l'AES ont été publiées. Nous mentionnons ici celles qui nous semblent les plus intéressantes car ne supposant que des modèles de faute et d'attaquant raisonnables.

Pierre Dusart, Gilles Letourneux et Olivier Vivolo [DLV03a, DLV03b] ont trouvé une attaque dans le modèle de modification aléatoire d'octet, et où la faute peut survenir n'importe où entre l'avant-dernier et le dernier `MixColumns`. Par la suite, Gilles Piret et Jean-Jacques Quisquater [PQ03] ont proposé une attaque plus efficace ne nécessitant que deux fautes en les injectant un tour plus tôt.

La première DFA sur l'AES dans ce modèle de faute a néanmoins été trouvée par Christophe Giraud [Gir03, Gir04]. Cette attaque casse l'AES à clé de 128 bits en ciblant l'expansion de la clé et nécessite de pouvoir injecter des fautes pendant la dérivation des dernière et avant-dernière clés de tour. Si l'octet affecté par la faute n'est pas choisi, l'attaque nécessite 248 fautes en moyennes pour retrouver la clé. Si l'attaquant peut choisir la position de l'octet fauté, alors 31 fautes suffisent. Cette attaque a été améliorée par la suite pour ne nécessiter que 22 [CY03], puis 12 fautes [PT06]. Récemment, Junko Takahashi, Toshinori Fukunaga et Kimihiro

Yamakoshi [TFY07] ont analysé en détail et de manière systématique l'effet d'une faute sur la dérivation des clés des deux derniers tours, et ont découvert une attaque révélant l'intégralité de la clé à l'aide de 7 fautes seulement.

L'exploitation des identités de sortie

La DFA infère de l'information à partir d'une différentielle en sortie de l'algorithme. À l'inverse, il existe une classe d'attaque par fautes qui exploite le fait que deux chiffrés sont identiques.

Décrivons tout d'abord deux techniques appartenant à cette classe d'attaque, la CFA et l'IFA. Elles supposent toutes deux un modèle de faute par écrasement, c'est-à-dire tel que la valeur prise par la donnée modifiée est fixe, égale à 0 par exemple.

Dans l'*analyse de faute par collisions* (en anglais, *Collision Fault Analysis*, CFA), un attaquant exécute l'algorithme sur une entrée M et provoque l'écrasement d'une valeur intermédiaire proche du début de l'algorithme. Il obtient ainsi un chiffré fauté C^*, puis trouve, par recherche exhaustive, le message M^* dont le chiffré non fauté est précisément égal à C^*. L'attaquant inférera alors de l'information sur la clé à partir de la connaissance de ce que la valeur intermédiaire lors du chiffrement de M^* valait elle aussi 0.

Une variante de la CFA est l'*analyse de faute sans effet* (en anglais, *Ineffective Fault Analysis*, IFA). Ici, l'attaquant est à la recherche d'une entrée pour laquelle les chiffrés normal et fauté sont identiques. Dans ce cas également, cela ne se produira que si la valeur intermédiaire au niveau de la faute vaut 0 pour l'exécution normale.

Une autre technique exploitant une identité de sortie s'appelle l'*analyse des erreurs sûres* (en anglais, *Safe-Error Analysis*) [JQYY02, YJ00, YKLM02]. Ce type d'attaque présente l'avantage de ne supposer qu'un modèle de faute aléatoire. L'exemple le plus typique est une attaque contre une variante de l'algorithme d'exponentiation *élever au carré et multiplier* (voir Figure 1.10) dans laquelle on effectue toujours une multiplication même lorsque celle-ci n'est pas nécessaire. On insère pour cela des multiplications fictives lorsque le bit de l'exposant vaut zéro. En fautant le calcul d'une multiplication, l'attaquant pourra déterminer si celle-ci était utile ou fictive (et donc si $d_i = 1$ ou 0) d'après l'observation que la signature retournée est modifiée ou non.

1.4.2 Les contre-mesures

Nous présentons maintenant différentes contre-mesures vis-à-vis des analyses de fautes, et discutons de leur efficacité en fonction de l'attaque considérée.

Plusieurs contre-mesures permettent de protéger une implémentation tout à la fois contre les analyses de canaux auxiliaires et contre les fautes.

Par exemple, tout mécanisme de désynchronisation, matériel ou logiciel, prévu pour gêner les SCA aura aussi pour effet de rendre plus difficile une injection de faute réussie. Ceci est particulièrement vrai pour les attaques nécessitant de cibler une instruction particulière. En revanche, l'attaque de *Bellcore* ou la DFA sur le DES ne sont pas très sensibles à cette protection.

Le masquage aléatoire des données intermédiaires, qui protège contre les analyses statistiques du courant, peut aussi contrer certaines attaques par fautes, précisément celles qui supposent une faute imposant à la donnée manipulée une valeur fixe connue de l'attaquant (par exemple, CFA et IFA). On peut toutefois contourner cette contre-mesure si la faute permet de fixer la valeur de plusieurs données chacune masquée avec le même aléa. Ce type d'attaque par fautes, que l'on pourrait qualifier d'ordre supérieur, et dont des exemples sont décrits aux Sections 10.4 et 10.5, devient envisageable lorsque la faute permet de dérouter l'exécution normale du processus attaqué.

Certaines protections sont en revanche tout à fait spécifiques aux attaques par fautes. Par exemple, une manière de contrer la DFA sur un algorithme cryptographique est de dupliquer le calcul. On effectue deux fois le chiffrement à l'identique et on ne retourne le chiffré que si les deux exécutions ont produit la même valeur. Alternativement, on peut aussi effectuer un chiffrement suivi d'un déchiffrement (ou une signature suivie d'une vérification) et ne retourner le chiffré (ou la signature) qui si l'on retombe bien sur la donnée d'entrée initiale. Cette contre-mesure protège contre la DFA mais est très pénalisante en temps d'exécution. De plus elle ne permet pas d'éviter les exploitations d'identités de sortie, sauf si elle est accompagnée d'un mécanisme rendant la carte inopérante après un certain nombre de fautes détectées.

Chapitre 2

Résumé de nos travaux

Nous donnons ici un résumé d'une partie de nos travaux. Ils ont, pour la plupart, fait l'objet d'une publication dans une conférence avec un comité de sélection. Bien que ces travaux abordent des sujets assez différents les uns des autres, ils relèvent presque tous du domaine de la sécurité physique des systèmes embarqués. Nous les avons répartis par thèmes dans les trois prochaines parties de ce mémoire.

Partie II : Les attaques

Cette partie est consacrée à la description d'attaques nouvelles qui s'appliquent à des implémentations non protégées.

Correlation Power Analysis with a Leakage Model [BCO04] :

Nous commençons par décrire au Chapitre 3 une nouvelle technique d'analyse des canaux auxiliaires : l'*analyse du courant par corrélation* (CPA). C'est une variante de l'*analyse différentielle du courant* (DPA) [KJJ99] de Paul Kocher, Joshua Jaffe et Benjamin Jun, et de ses dérivés. Historiquement, notre démarche a été d'expliquer et de prédire les abérrations de la DPA, appelées "pics fantômes", afin de pouvoir exploiter toute l'information sur la clé qu'ils contiennent dans leur ensemble. De cette quête, il a résulté notre technique, décrite en détail en 2003 [BCO03] et publiée à Ches '04 [BCO04], avec Éric Brier et Francis Olivier. Nous utilisons un modèle de fuite et analysons toute l'information disponible sur la valeur de la donnée manipulée, permettant ainsi une discrimination plus efficace des différentes hypothèses de sous-clés considérées.

On the Implementation of a Fast Prime Generation Algorithm [CC07] :

Nous présentons ensuite au Chapitre 4 un exemple d'*analyse simple du courant* (SPA) qui offre l'originalité de s'appliquer à la génération de nombres premiers, et donc de clés RSA. Dans un article publié à Ches '07 [CC07], avec Jean-Sébastien Coron, nous montrons que, lorsqu'il est possible de déterminer par SPA l'issue d'un branchement conditionnel (ce qui nous donne la parité d'un résultat intermédiaire), alors une certaine variante de l'algorithme de génération de premiers proposé par Marc Joye et Pascal Paillier à Ches '06 [JP06] est vulnérable à une attaque permettant de retrouver la factorisation d'une fraction faible, mais non négligeable, des modules RSA qu'elle construit. Nous proposons plusieurs contre-mesures pour empêcher cette attaque.

Why One Should Also Secure RSA Public Key Elements [BCCC06] :

Le Chapitre 5 aborde le domaine des attaques par injection de fautes. Il y est décrit une variante améliorée d'une attaque de Jean-Pierre Seifert [Sei05] basée sur l'originale idée d'exploiter la modification d'un élément *public* d'une clé RSA, en l'occurence le module. Contrairement à ce que permet l'idée décrite dans [Sei05], notre attaque, publiée à Ches '06 [BCCC06], avec Éric Brier, Benoît Chevallier-Mames et Mathieu Ciet, offre la possibilité de réellement casser la clé en retrouvant l'exposant privé. C'est d'ailleurs à notre connaissance la seule attaque par fautes connue qui permet de retrouver l'exposant privé d'une clé RSA en ne corrompant que des éléments publics de la clé. C'est aussi, nous semble-t-il, la seule attaque par faute connue sur le RSA en mode standard qui ne repose sur aucun modèle de faute, ni sur aucune hypothèse d'implémentation. C'est enfin l'attaque par faute sur le RSA en mode standard qui nécessite le moins d'injections de fautes pour retrouver la clé.

Case Study of a Fault Attack on Asynchronous DES Crypto-processors [MRL+06b] :

Une étude de la résistance d'un crypto-processeur DES asynchrone aux attaques par injection de fautes est présentée au Chapitre 6. Bien qu'on attende d'un circuit asynchrone qu'il soit intrinsèquement plus résistant aux injections de fautes qu'un circuit synchrone, nous observons dans cette étude, publiée à Fdtc '06 [MRL+06b] avec Yannick Monnet, Marc Renaudin, Régis Leveugle et Pascal Moitrel, que le circuit asynchrone étudié produit et renvoie des chiffrés erronés que nous avons pu interpréter comme étant provoqués par une modification de la valeur du compteur de tours. Nous expliquons alors de manière détaillée comment exploiter cette faiblesse particulière. Il est intéressant de noter que cette faiblesse s'observe également, certes moins fréquemment, dans une version "durcie", c'est-à-dire conçue pour détecter et filtrer les injections de fautes, de ce type de crypto-processeur asynchrone.

Fault Analysis Study of IDEA (en cours de soumission) :

Enfin, nous concluons cette partie au Chapitre 7 par la description d'une *analyse différentielle de fautes* (DFA) sur l'algorithme IDEA. Par conception, cet algorithme de chiffrement symétrique n'utilise pas de boîtes de substitution, mais plutôt un mélange d'opérations dans trois groupes algébriques différents. Il est aujourd'hui réputé pour être un des algorithmes cryptographiques à clé secrète qui résiste le mieux à la cryptanalyse. L'attaque que nous présentons, proposée conjointement avec Benedikt Gierlichs, utilise un modèle de fautes très général dans lequel l'effet requis de la faute est simplement de modifier aléatoirement une donnée. Notre attaque permet de retrouver aisément 93 des 128 bits de la clé à l'aide d'un très petit nombre de fautes.

Partie III : Les contre-mesures

Dans cette partie, nous considérons les contre-mesures dont peuvent être dotées les implémentations pour se protéger des attaques par analyse de canaux auxiliaires, ou par exploitation de fautes. La conception de telles contre-mesures est évidemment primordiale pour garantir la sécurité des dispositifs à cryptographie embarquée.

Universal Exponentiation Algorithm [CJ01] :

Nous présentons ainsi au Chapitre 8 un article publié à Ches '01 [CJ01], avec Marc Joye, dans lequel nous décrivons un codage de l'exposant privé d'une clé RSA basé sur une décomposition de celui-ci en chaîne d'additions. Nous proposons alors un algorithme universel d'exponentiation qui utilise ce codage et transfère la sécurité d'une quelconque méthode d'exponentiation

vers celle de la multiplication modulaire considérée comme opération élémentaire. Cet algorithme, simple à implémenter, facilite grandement l'analyse de sécurité vis à vis de la SPA[11], et constitue un premier pas vers une certaine forme de sécurité prouvée des implémentations réelles.

Au delà de la conception proprement dite des contre-mesures, il nous semble important de pouvoir juger le plus précisément possible de leurs domaines d'application, ainsi que de leurs forces et de leurs limitations. Il est donc pour cela utile de les analyser, de les remettre en question, et d'essayer de trouver des attaques contre celles-ci. Les deux derniers chapitres de cette partie sont ainsi consacrés aux attaques de contre-mesures isolées ou d'implémentations protégées.

Differential Power Analysis in the Presence of Hardware Countermeasures [CCD00] :

Nous présentons au Chapitre 9 une étude de la contre-mesure au niveau matériel consistant à insérer aléatoirement des cycles d'horloge fictifs assurant un certain niveau de désynchronisation des exécutions. Dans un article publié à Ches '00 [CCD00], avec Jean-Sébastien Coron et Nora Dabbous, nous montrons comment il est possible de contourner en partie cette contre-mesure et rehausser ainsi le pic de DPA attendu.

Fault Analysis of DPA-Resistant Algorithms [ACT06] :

Le Chapitre 10 décrit quant à lui plusieurs attaques publiées à Fdtc '06 [ACT06], avec Frédéric Amiel et Michael Tunstall, contre des implémentations du DES ou de l'AES protégées contre les attaques différentielles (DPA,CPA). Pour certaines d'entre elles, nous proposons des solutions pour les éviter et redonner toute leur efficacité aux contre-mesures initiales.

Partie IV : Les attaques sur algorithmes inconnus

Depuis leur apparition, et pendant une longue période, les attaques physiques se sont essentiellement attachées à retrouver la clé cryptographique utilisée par un algorithme connu (DES, AES, RSA,...). Ce n'est qu'en 2003 qu'a été publiée par Roman Novak [Nov03] la première utilisation d'une analyse de canaux auxiliaires dans le cas particulier où les spécifications de l'algorithme sont inconnues de l'attaquant. Dans le domaine des analyses de fautes, Eli Biham et Adi Shamir [BS97] ont proposé dès l'apparition de ces techniques les seules attaques sur crypto-systèmes inconnus dont nous ayons connaissance[12]. C'est un contexte intéressant et exigeant mais qui n'a été que peu étudié jusqu'à présent. Il est intéressant car il existe encore de nombreux exemples, notamment dans les secteurs de la téléphonie mobile et de la télévision à péage (pour ne citer que des domaines civils), où la confidentialité de l'algorithme, contraire au principe de Kerckhoffs, participe à la sécurité espérée du système. Il est exigeant car la connaissance de l'algorithme facilite généralement beaucoup la conception des attaques. Nous présentons dans cette partie notre contribution à ce domaine nouveau de recherche à travers divers exemples d'exploitation des canaux auxiliaires et de fautes applicables dans le cas où l'algorithme est, au moins en grande partie, inconnu de l'attaquant. Ces attaques se répartissent en deux catégories en fonction de l'objectif de l'attaquant qui peut être, soit de réaliser une rétro-conception partielle ou totale de l'algorithme et ainsi révéler des détails fonctionnels de ses

[11] Concernant la sécurité du RSA vis-à-vis de l'analyse du courant, la SPA constitue une menace plus importante que la DPA dont on peut se protéger facilement par modification aléatoire de la base, de l'exposant ou du module.
[12] L'une de ces attaques sera améliorée par Pascal Paillier quelques années plus tard [Pai99].

spécifications, soit plus simplement de retrouver la clé cryptographique malgré l'ignorance de la fonction qui l'utilise.

An Improved SCARE Cryptanalysis Against a Secret A3/A8 GSM Algorithm [Cla07b] :

Dans la première catégorie, nous présentons au Chapitre 11 un exemple d'une attaque SCARE (en anglais, *Side Channel Analysis for Reverse Engineering*) appliquée à un algorithme secret de type A3/A8 utilisé pour l'authentification et l'échange de clés sur le réseau GSM. Reprenant et améliorant le travail de Roman Novak [Nov03] évoqué ci-dessus, notre attaque, décrite en 2004 [Cla04] et publiée à Iciss '07 [Cla07b], permet de retrouver les valeurs de deux tables de substitution (S-Box) inconnues de l'attaquant, à partir de la connaissance d'une partie minime des spécifications. Incidemment, cette attaque a aussi la propriété de révéler la clé en sus des valeurs des tables.

Les deux derniers chapitres de cette partie sont dédiés aux attaques dans lesquelles l'objectif est "simplement" de retrouver la clé. Nous y présentons deux attaques ayant chacune l'avantage de s'appliquer à une grande classe d'algorithmes. La première, présentée au Chapitre 12 mais qui n'a jamais été publiée, est une généralisation triviale d'une *analyse de fautes par collisions* (CFA) sur l'AES, à l'ensemble des algorithmes cryptographiques débutant par un XOR entre le message et la clé. La deuxième fait l'objet du Chapitre 13 :

Secret External Encodings Do not Prevent Transient Fault Analysis [Cla07a] :

Publiée à Ches '07 [Cla07a], cette attaque concerne la classe des fonctions de chiffrement par bloc dans lesquelles deux bijections inconnues P_1 et P_2, opérant respectivement sur les espaces des messages et des chiffrés, obfusquent l'entrée et la sortie de l'algorithme DES ou Triple-DES. Nous montrons que l'attaquant peut mener une *analyse de fautes sans effet* (IFA) pour retrouver une grande quantité d'information sur la clé utilisée sans avoir à faire de supposition concernant lesdites couches d'obfuscation P_1 et P_2.

Annexe

Enfin, nous présentons séparément nos travaux n'ayant pas trait à la sécurité physiques des systèmes embarqués.

Cryptanalysis of RSA Signatures with Fixed-Pattern Padding [BCCN01] :

Un article publié à Crypto '01 [BCCN01], avec Éric Brier, Jean-Sébastien Coron et David Naccache, est proposé en Annexe A. Nous y décrivons une méthode permettant de créer des falsifications de signatures RSA lorsque celui-ci est utilisé avec un padding fixe. Nous montrons que cette méthode est applicable dès lors que la taille du padding relativement à la taille du module est inférieure à $\frac{2}{3}$, améliorant par là la précédente borne qui était de $\frac{1}{2}$.

Deuxième partie

Les attaques

Chapitre 3

Analyse du courant par corrélation avec un modèle de fuite

Sommaire

L’analyse différentielle du courant, proposée par Paul Kocher en 1998, a constitué un important progrès dans le domaine des attaques physiques. L’apparition de cette technique a bouleversé bien des idées et des pratiques, tant (certes lentement) dans le monde académique de la cryptologie, que dans l’industrie de la sécurité des systèmes embarqués. Néanmoins, il est apparu progressivement que la puissance de cet outil était parfois limitée par le fait que certains des pics qu’elle permettait d’obtenir étaient difficiles à interpréter et contradictoires, en apparence, avec la donnée secrète utilisée. Nous avons souhaité comprendre pourquoi et comment ces pics “fantômes” survenaient, afin de pouvoir les “déchiffrer” et les exploiter, plutôt que de les subir. C’est dans le cadre de cette démarche que nous avons compris que la DPA classique reposait sur des hypothèses hasardeuses, et était loin d’utiliser toute l’information disponible pour l’attaquant. Nous présentons dans ce chapitre les résultats issus de cette quête de compréhension, qui ont été publiés à Ches ’04 conjointement avec Éric Brier et Francis Olivier [BCO04].

Nous utilisons dans ce chapitre un modèle classique pour la consommation de courant des dispositifs cryptographiques. Celui-ci est basé sur la distance de Hamming entre la donnée manipulée et un état de référence inconnu mais constant. Une fois validé expérimentalement, nous en dérivons une technique d’attaque appelée *analyse du courant par corrélation*. Ce modèle nous permet également d’expliquer les défauts de l’analyse différentielle.

3.1 Introduction

Dans le domaine des analyses statistiques du courant contre les dispositifs cryptographiques, deux tendances historiques peuvent être observées. La première est la très connue *analyse différentielle du courant* (DPA) introduite par Paul Kocher, Joshua Jaffe et Benjamin Jun [KJJ98, KJJ99] et formalisée par Thomas Messerges, Ezzy Dabbish et Robert Sloan [MDS99]. La seconde a été suggérée dans des articles divers [CKN01, May00, Osw03] et propose d'utiliser le facteur de corrélation entre les échantillons de courant et le poids de Hamming de la donnée manipulée. Les deux approches montrent certaines limitations dues à des hypothèses non réalistes et à des imperfections de modèles qui vont être étudiées plus en détails dans ce chapitre. Ce travail poursuit celui d'études précédentes qui avaient pour objectif ou bien d'améliorer le modèle du poids de Hamming [ABDM00], ou bien de renforcer la DPA elle-même par différents moyens [CCD00, BK03].

L'approche proposée est basée sur le modèle de la distance de Hamming qui peut être vu comme une généralisation du modèle du poids de Hamming. Toutes ses hypothèses de bases ont déjà été mentionnées dans différents articles [MDS99, CKN01, CCD00, ABDM00]. Mais elles restaient de vagues explications possibles aux défauts de la DPA et ne leur ont jamais apporté une explication complète et satisfaisante. Notre travail expérimental est une synthèse de ces approches précédentes visant à donner une interprétation correcte de la fuite des données. Suivant [CKN01, May00, Osw03] nous proposons d'utiliser l'*analyse du courant par corrélation* (CPA) pour identifier les paramètres du modèle de fuite. Nous montrons alors que des attaques solides et efficaces peuvent être conduites contre des implémentations non protégées de divers algorithmes comme le DES ou l'AES. Cette étude est délibérément restreinte au domaine de la cryptographie à clé secrète bien qu'elle puisse être étendue au-delà.

Ce chapitre est organisé de la manière suivante : la Section 3.2 introduit le modèle de la distance de Hamming et la Section 3.3 prouve la pertinence du facteur de corrélation. L'attaque par corrélation est décrite à la Section 3.4 où nous discutons également de l'influence de certaines erreurs sur le modèle. La Section 3.5 traite du problème de l'estimation, et les résultats expérimentaux qui valident le modèle sont exposés à la Section 3.6. La Section 3.7 comprend l'étude comparative avec la DPA et traite plus spécifiquement de ce que l'on nomme le problème des "pics fantômes", rencontré par ceux qui souffrent de conclusions erronées lorsqu'ils implémentent la DPA classique sur les boîtes de substitution du premier tour du DES : il est montré comment le modèle proposé explique plusieurs défauts de la DPA et comment l'*analyse du courant par corrélation* peut aider à mener des attaques robustes dans des conditions optimales. Notre conclusion résume les avantages et les inconvénients de la CPA par rapport à la DPA, et rappelle que les contre-mesures fonctionnent tout aussi bien pour les deux méthodes.

3.2 Le modèle de consommation de la distance de Hamming

Classiquement, la plupart des analyses de courant trouvées dans la littérature reposent sur le modèle du poids de Hamming [KJJ99, MDS99], c'est-à-dire sur le nombre de bits à 1 dans un mot de donnée. Dans un micro-processeur à m bits, une donnée binaire est codée $D = \sum_{j=0}^{m-1} d_j 2^j$, avec les valeurs des bits $d_j = 0$ ou 1. Son poids de Hamming est simplement le nombre de bits égaux à 1, $H(D) = \sum_{j=0}^{m-1} d_j$. Ses valeurs entières sont comprises entre 0 et m. Si D contient m bits indépendants et uniformément distribués, le mot complet a un poids de Hamming moyen $\mu_H = m/2$ et une variance $\sigma_H^2 = m/4$.

Un micro-processeur est modélisé comme une machine d'état où les transitions d'état à état

sont déclenchées par des évènements comme les fronts d'un signal d'horloge. Il est communément supposé que la fuite sur la donnée à travers le canal auxiliaire de courant dépend du nombre de bits qui basculent d'un état à l'autre [CCD00, CKN01] à un instant donné. Cela semble pertinent lorsque nous considérons une porte logique élémentaire implémentée en technologie CMOS. Le courant consommé est lié à l'énergie requise pour faire basculer les bits d'un état à l'état suivant. Il est formé de deux contributions principales : la charge des condensateurs et le court-circuit induit par la transition de la porte. Curieusement, ce comportement élémentaire est communément admis mais n'a mené à aucun modèle satisfaisant qui soit largement applicable. Seuls les concepteurs de circuits sont familiers avec les outils de simulation pour prévoir la consommation des dispositifs micro-électroniques.

Si le modèle de transitions est adopté, une question basique se pose : quel est l'état de référence à partir duquel les bits sont basculés ? Nous supposons ici que cet état de référence est un mot machine constant, R, qui est inconnu mais pas nécessairement nul. Il sera toujours le même si la manipulation de la donnée a toujours lieu au même instant, bien que cela suppose l'absence de tout effet de désynchronisation. De plus, on suppose que faire basculer un bit de 0 à 1 ou de 1 à 0 requiert la même quantité d'énergie et que tous les bits machines manipulés à un instant donné sont parfaitement équilibrés et consomment autant.

Ces hypothèses restrictives sont tout à fait réalistes et abordables sans aucune connaissance profonde des composants micro-électroniques. Elles mènent à une expression simple pour le modèle de fuite. En effet le nombre de bits qui basculent de R à D est donné par $H(D \oplus R)$ aussi appelé distance de Hamming entre D et R. Cette expression englobe le modèle du poids de Hamming qui suppose que $R = 0$. Si D est une variable aléatoire uniforme, il en est alors de même pour $D \oplus R$, et $H(D \oplus R)$ a les mêmes moyenne $m/2$ et variance $m/4$ que $H(D)$.

Nous supposons également une relation linéaire entre la consommation de courant et $H(D \oplus R)$. Cela peut être perçu comme une limitation, mais considérant un composant comme un grand ensemble de composants électriques élémentaires, ce modèle linéaire s'accorde assez bien à l'expérience. Cela ne représente pas la consommation entière d'un composant mais seulement la partie dépendant de la donnée. Cela ne semble pas irréaliste car les lignes de bus sont généralement considérées comme étant les éléments les plus consommateurs dans un micro-contrôleur. Toutes les autres contributions dans la consommation de courant d'un composant sont affectées à un terme noté b qui est supposé indépendant des autres variables : b comprend les décalages de calibration instrumentale, les composantes dépendant du temps et le bruit. Le modèle basique de dépendance vis-à-vis de la donnée peut donc s'écrire :

$$W = a \times H(D \oplus R) + b \,,$$

où a est un facteur de gain entre la distance de Hamming et W le courant consommé.

3.3 Le facteur de corrélation linéaire

Un modèle linéaire implique certaines relations entre les variances des différents termes considérés comme des variables aléatoires : $\sigma_W^2 = a^2\sigma_H^2 + \sigma_b^2$. Les statistiques classiques introduisent le facteur de corrélation ρ_{WH} entre la distance de Hamming et le courant mesuré pour estimer le taux d'accord avec le modèle linéaire. C'est la covariance entre les deux variables normalisée par le produit de leurs écarts-types. Sous l'hypothèse d'un bruit indépendant, cette définition conduit à :

$$\rho_{WH} = \frac{\mathrm{cov}(W,H)}{\sigma_W \sigma_H} = \frac{a\sigma_H}{\sigma_W} = \frac{a\sigma_H}{\sqrt{a^2\sigma_H^2 + \sigma_b^2}} = \frac{a\sqrt{m}}{\sqrt{ma^2 + 4\sigma_b^2}} \,.$$

Cette équation satisfait la propriété bien connue : $-1 \leqslant \rho_{WH} \leqslant +1$: pour un modèle parfait le facteur de corrélation tend vers ± 1 si la variance du bruit tend vers 0, le signe dépendant du signe du gain linéaire a. Si le modèle s'applique seulement à l bits indépendants parmi m, une corrélation partielle existe tout de même :

$$\rho_{WH_{l/m}} = \frac{a\sqrt{l}}{\sqrt{ma^2 + 4\sigma_b^2}} = \rho_{WH}\sqrt{\frac{l}{m}} \,.$$

3.4 Inférence de secret par *analyse du courant par corrélation*

Les relations données ci-dessus montrent que si le modèle est valide alors la corrélation est maximisée lorsque la variance du bruit est minimale. Cela signifie que ρ_{WH} peut aider à déterminer l'état de référence R. Supposons, tout comme dans la DPA, que l'on dispose d'un ensemble de données D connues et variables, et de l'ensemble des consommations de courant W qui leur correspondent. Si les 2^m valeurs possibles pour R sont parcourues exhaustivement, elles peuvent être ordonnées suivant le facteur de corrélation qu'elles produisent quand elles sont combinées avec les observations W. Ce n'est pas très coûteux lorsqu'on considère un microcontrôleur à 8 bits, ce qui est le cas pour de nombreuses cartes à puce actuelles, puisque seulement 256 valeurs doivent être testées. Sur des architectures 32 bits, cette recherche exhaustive ne peut pas être menée telle quelle facilement. Mais il est toujours possible de travailler avec des corrélations partielles ou d'introduire de la connaissance *a priori*.

Soit R la vraie référence et $H = H(D \oplus R)$ la prédiction correcte de la distance de Hamming. Représentons par R' une valeur candidate et par H' le modèle correspondant $H' = H(D \oplus R')$. Supposons une valeur de R' avec k bits différents de ceux de R, de sorte que : $H(R \oplus R') = k$. Puisque b est indépendant des autres variables, le test de corrélation conduit à (voir [BCO03]) :

$$\rho_{WH'} = \frac{\mathrm{cov}(aH + b, H')}{\sigma_W \sigma'_H} = \frac{a}{\sigma_W}\frac{\mathrm{cov}(H, H')}{\sigma'_H} = \rho_{WH}\rho_{HH'} = \rho_{WH}\frac{m - 2k}{m} \,.$$

Cette formule montre comment le facteur de corrélation est capable de rejeter de mauvais candidats pour R. Par exemple, si un seul bit est incorrect sur un mot de 8 bits, alors la corrélation est réduite d'un quart. Si tous les bits sont incorrects, c'est-à-dire si $R' = \neg R$, alors une anti-corrélation doit être observée avec $\rho_{WH'} = -\rho_{WH}$. En valeur absolue, ou si le gain linéaire est supposé positif ($a > 0$), il ne peut exister aucun R' conduisant à un taux de corrélation plus élevé que pour R. Cela prouve l'unicité de la solution et ce faisant comment l'état de référence peut être déterminé.

Cette analyse peut être menée sur la trace de courant attribuée à une portion de code pendant qu'il manipule des données variables et connues. Si nous supposons que la donnée manipulée est le résultat d'une opération XOR entre un mot de clé secrète K et un mot de message connu M, $D = K \oplus M$, la procédure décrite ci-dessus, c'est-à-dire la recherche exhaustive sur R et le test de corrélation, devrait aboutir à $K \oplus R$ associé à $\max(\rho_{WH})$. En effet, si une corrélation existe quand M est manipulé par rapport à R_1, une autre doit survenir plus tard lorsque $M \oplus K$ est manipulé à son tour, avec un état de référence possiblement différent R_2 (en réalité avec $K \oplus R_2$ puisque seul M est connu).

Par exemple, considérant la première fonction `AddRoundKey` au début de l'algorithme AES embarqué sur un processeur à 8 bits, il est évident qu'une telle méthode conduit à la clé complète masquée par l'octet de référence constant R_2. Si R_2 est le même pour tous les octets de clé, ce qui est très plausible, il reste seulement 2^8 possibilités à tester par recherche exhaustive pour en

déduire la clé complète. Cette recherche exhaustive complémentaire peut être évitée si R_2 est déterminé par d'autres moyens ou connu pour être toujours égal à zéro (sur certains composants).

Cette attaque n'est pas restrictive à l'opération $\oplus$. Elle s'applique aussi à plusieurs autres opérateurs souvent rencontrés en cryptographie à clé secrète. Par exemple d'autres opérations arithmétiques ou logiques, ou des substitutions par lecture de table (LUT), peuvent être traitées de la même manière en utilisant H(LUT($M \star K$) $\oplus$ R), où $\star$ représente la fonction impliquée, par exemple $\oplus$, +, -, OR, AND, ou toute autre opération. Remarquons que l'ambiguïté entre K et $K \oplus R$ est complètement supprimée par les boîtes de substitution dans les algorithmes à clé secrète grâce à la non-linéarité de la lecture de table correspondante : cela peut impliquer la nécessité de rechercher exhaustivement à la fois K et R, mais une fois seulement pour R dans la plupart des cas. Pour conduire une analyse dans les meilleures conditions, nous attirons l'attention sur le bénéfice de correctement modéliser le mot machine entier qui est réellement manipulé et sa transition par rapport à l'état de référence R, lequel doit être déterminé comme une inconnue du problème.

3.5 Estimation

Dans un cas réel, avec un ensemble de N courbes de courant W_i et N mots de données aléatoires correspondants M_i, pour un état de référence donné R, les mots de données connus produisent un ensemble de N distances de Hamming prédites $H_{i,R} = H(M_i \oplus R)$. Une estimation $\hat{\rho}_{WH}$ du facteur de corrélation ρ_{WH} est donnée par l'expression suivante :

$$\hat{\rho}_{WH}(R) = \frac{N \sum W_i H_{i,R} - \sum W_i \sum H_{i,R}}{\sqrt{N \sum W_i^2 - (\sum W_i)^2}\sqrt{N \sum H_{i,R}^2 - (\sum H_{i,R})^2}} ,$$

où les sommations s'appliquent sur les N échantillons ($i = 1, \ldots, N$) à chaque instant élémentaire t des traces de courant $W_i(t)$.

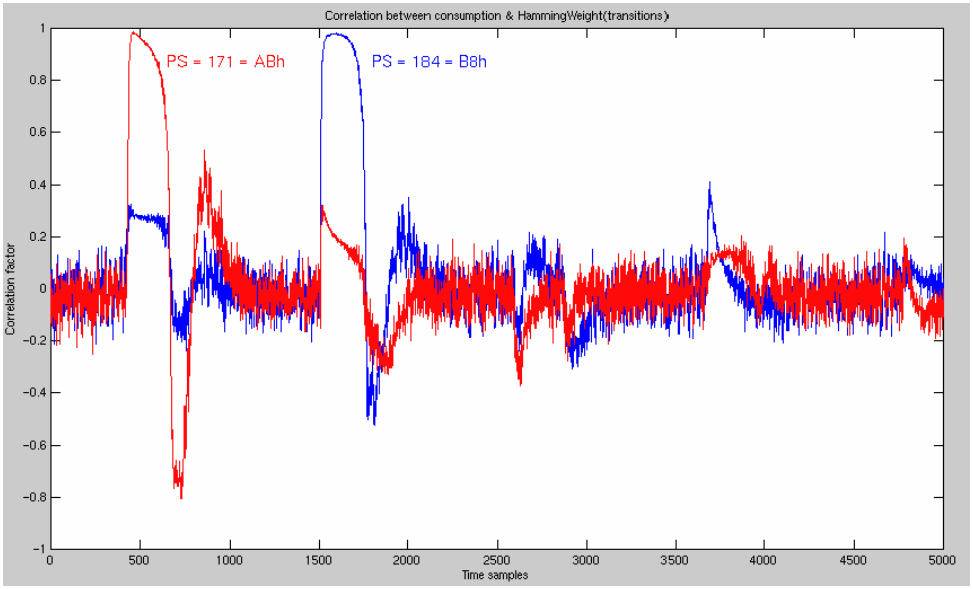
Il est théoriquement difficile de calculer la variance de l'estimateur $\hat{\rho}_{WH}$ en fonction du nombre N d'échantillons disponibles. Dans la pratique quelques centaines d'expériences (courbes) suffisent pour fournir une estimation exploitable du facteur de corrélation. N doit être augmenté avec la variance du modèle $m/4$ (plus élevée sur les architectures à 32 bits) ou bien sûr en présence d'un niveau de bruit élevé. Nous renvoyons le lecteur à [BCO03] pour une discussion plus détaillée sur l'estimation à partir de données expérimentales et sur le problème de l'optimalité. Il y est montré que cette approche peut être vue comme une procédure d'ajustement de modèle au maximum de vraisemblance, lorsque R est recherché exhaustivement pour maximiser $\hat{\rho}_{WH}$.

3.6 Résultats expérimentaux

Cette section vise à confronter le modèle de fuite avec des expériences réelles. Des règles générales de comportement sont déduites de l'analyse de plusieurs composants pour des dispositifs sécurisés.

Notre première expérience a été réalisée sur un algorithme basique XOR implémenté sur un composant à 8 bits connu pour sa fuite d'information (plutôt destiné à un propos didactique). La séquence d'instructions était simplement la suivante :

- chargement de l'octet D_1 dans l'accumulateur
- XOR de D_1 avec une constante D_2 (connue de l'attaquant)
- stockage du résultat depuis l'accumulateur vers une cellule mémoire



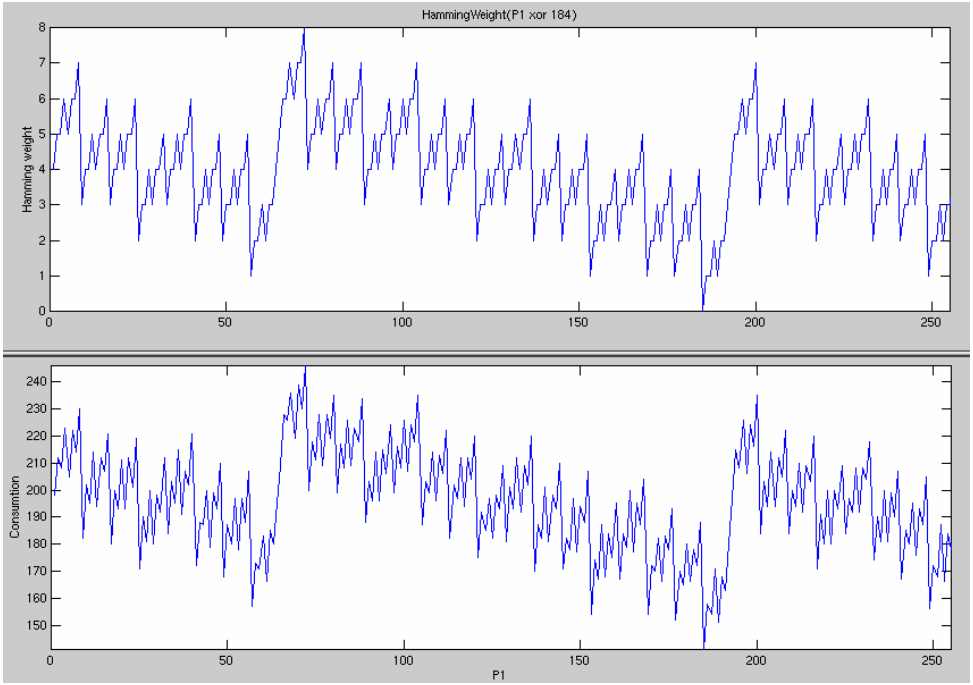


FIG. 3.1 – En haut : pics de corrélation successifs pour deux états de référence différents. En bas : Pour une donnée variable (0-255), table de modèle et table d'observation obtenues à l'instant du second pic.

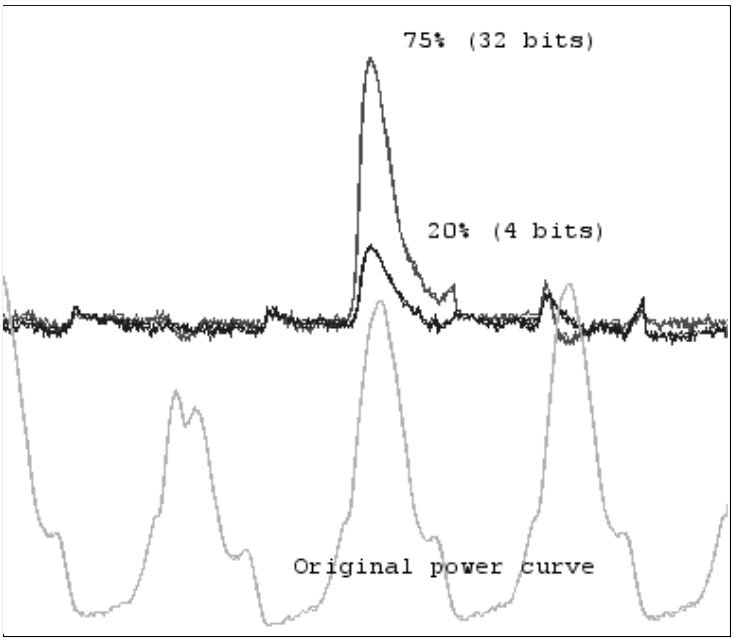


FIG. 3.2 – Deux pics de corrélation pour des prédictions sur mot entier (32 bits) et partielle (4 bits). La théorie voudrait que le pic à 20% soit plutôt autour de 26%.

Le programme a été exécuté 256 fois avec D_1 variant de 0 à 255. Comme illustré sur la Figure 3.1, deux pics de corrélation significatifs ont été obtenus avec deux états de référence différents : le premier étant l'adresse de D_1, le second le code d'opération de l'instruction XOR. Ces courbes apportent le témoignage expérimental des principes de fuite que les travaux précédents ont suggérés sans entrer dans les détails [MDS99, CKN01, CCD00, MDS02]. Elles illustrent le cas le plus général d'une séquence de transfert sur un bus commun (architecture dite de VON NEUMAN). L'adresse d'un mot de donnée est transmise juste avant sa valeur qui est elle-même immédiatement suivie par le code d'opération de l'instruction suivante qui est rapportée. Un tel comportement peut être observé sur une grande variété de composants, même ceux implémentant des architectures à 16 ou 32 bits. Des taux de corrélation allant de 60% à plus de 90% sont souvent obtenus. La Figure 3.2 montre un exemple de corrélation partielle sur une architecture à 32 bits : lorsque seulement 4 bits sur 32 sont prédits, la perte de corrélation est dans un rapport de l'ordre de $\sqrt{8}$ ce qui est cohérent avec les corrélations affichées.

Ce type de résultats peut être observé sur différentes technologies et implémentations. Néanmoins, nous devons exprimer les remarques suivantes :

- Dans certains cas, l'état de référence est systématiquement égal à zéro. Cela peut être attribué à ce que l'on nomme la logique pré-chargée où le bus est effacé entre chaque valeur transférée. Une autre raison possible est que des architectures complexes implémentent des bus de donnée et d'adresse séparés (architecture dite de Harvard), ce qui peut interdire certaines transitions. Dans tous ces cas, le modèle de poids du Hamming est retrouvé comme un cas particulier du modèle plus général de la distance de Hamming.
- La séquence des pics de corrélation peut suinter ou être étalée dans le temps en présence d'un pipeline.

Il existe différents types de contre-mesures qui sont tout à fait similaires à celles conçues contre la DPA :

– Certaines consistent à introduire des désynchronisations dans l'exécution du processus de sorte que les courbes d'un ensemble d'acquisitions ne sont plus alignées. Il existe pour cela différentes techniques telles que l'insertion de cycles fictifs, l'horloge instable ou les délais aléatoires [CCD00, Osw03]. Dans certains cas, leur effet peut être corrigé par application d'un traitement de signal approprié.
– D'autres contre-mesures consistent à rendre floues et imprécises les traces de courant par addition de bruit ou d'un circuit filtrant [Sha00]. Elles peuvent parfois être contournées par sélection et/ou moyennage des courbes ou en utilisant un autre canal auxiliaire tel que le rayonnement électromagnétique [GMO01, AARR03].
– La donnée peut aussi être déchiffrée dynamiquement pendant un processus par des mécanismes hardware (tel que le chiffrement de bus) ou logiciel (masquage de donnée avec un aléa [GP99, CG00, TSG03, GT03]), de sorte que les variables manipulées deviennent imprédictibles : on ne doit alors plus attendre de corrélation. En théorie, des attaques sophistiquées comme l'analyse d'ordre supérieur [Mes00] peuvent venir à bout de la méthode de masquage de donnée, mais on les contourne en pratique en utilisant par exemple la désynchronisation.

Dans les faits, implémentée seule, aucune de ces contre-mesures ne peut être considérée comme absolument sûre contre les analyses statistiques. Elles ne font qu'accroître l'effort et le niveau d'expertise requis pour mener l'attaque. Néanmoins, des défenses combinées, implémentant au moins deux de ces contre-mesures, se révèlent très efficaces et dissuasives en pratique. L'état de l'art des contre-mesures dans la conception de dispositifs sûrs a réalisé de grands progrès durant les dernières années. Il est maintenant admis que les exigences de sécurité incluent des implémentations solides tout autant que des schémas cryptographiques robustes.

3.7 Comparaison avec la DPA

Cette section traite de la comparaison de la méthode CPA proposée avec l'analyse différentielle du courant. Elle se réfère aux précédents travaux de Thomas Messerges, Ezzy Dabbish et Robert Sloan [MDS99, MDS02] qui ont formalisé les idées précédemment suggérées par Paul Kocher, Joshua Jaffe et Benjamin Jun [KJJ98, KJJ99]. Une étude critique est proposée dans [BCO03].

3.7.1 Problème pratique de la DPA : les "pics fantômes"

Nous considérons ici l'implémentation classique de la DPA contre les substitutions du DES au premier tour. Dans les faits cette attaque bien connue ne fonctionne correctement que si les hypothèses suivantes sont vérifiées :

1. Hypothèse relative au mot : dans le mot contenant le bit prédit, les contributions des bits non prédits sont indépendantes de la valeur du bit prédit. Leur influence moyenne dans le paquet de courbes "0" est la même que celle dans le paquet de courbes "1". Ainsi l'attaquant n'a pas à se soucier des valeurs de ces bits.
2. Hypothèse relative aux suppositions : la valeur prédite du bit cible pour toute supposition incorrecte de la sous-clé ne dépend pas de la valeur associée à la supposition correcte.
3. Hypothèse relative au temps : la consommation de courant W ne dépend pas de la valeur du bit cible sauf lorsque celui-ci est explicitement manipulé.

Mais lorsque l'on est confronté à l'expérience, l'attaque vient se heurter aux faits suivants :

TAB. 3.1 – Similarité des prédictions du bit 1 de la S-Box 6 pour les suppositions 0 et 36.

	$D = 0$ … $D = 63$
MSB(S6($D \oplus 0$))	1**1**0110101**0**0101100010010110**0**10011**1**10101**0**0101101**1**0101010010001**0**1**1**01
MSB(S6($D \oplus 36$))	1**0**0110101**1**0101100010010111**1**0100**1**0101011**0**1011**0**1**0**0101010010001**1**1**0**01
XOR	0**1**0000000**1**0000000000000001**0**0000**1**0000**1**0000000**1**0000000000000**1**0**1**00

– *Fait A.* Pour la supposition de sous-clé correcte, des pics de DPA apparaissent également lorsque le bit cible n'est pas explicitement manipulé. Il est utile de le mentionner même si ce n'est pas vraiment gênant. Cela vient cependant en contradiction avec la troisième hypothèse.
– *Fait B.* Des pics de DPA apparaissent également pour des suppositions de sous-clé incorrectes : ils sont appelés "pics fantômes". Ce fait est plus problématique pour prendre une décision fiable, et vient contredire la deuxième hypothèse.
– *Fait C.* Le vrai pic de DPA donné par la supposition de sous-clé correcte peut être plus petit que certains pics fantômes. Il peut même être nul voire négatif. Cela semble assez bizarre et tout à fait déroutant pour un attaquant. Les raisons doivent être recherchées dans la nature grossière de la première hypothèse par trop optimiste et simplificatrice.

3.7.2 L'explication des "pics fantômes"

Grâce à une étude approfondie des boîtes de substitution et du modèle de la distance de Hamming, il est maintenant possible d'expliquer les faits observés et de montrer en quoi les hypothèses de bases de la DPA sont erronées.

Fait A. C'est un fait que certaines données manipulées durant l'algorithme peuvent être partiellement corrélées avec le bit cible. Ce n'est pas si surprenant quand on regarde à la structure du DES. Un bit du quartet de sortie d'une S-Box a une "durée de vie" au moins jusqu'à la fin du tour (et au-delà si la partie gauche de la sortie de la permutation initiale ne varie pas trop). Un pic de DPA surgit chaque fois que ce bit et ses 3 bits compagnons sont traités par la permutation P puisqu'ils appartiennent au même mot machine.

Fait B. La raison pour laquelle des suppositions de sous-clé incorrectes peuvent générer des pics de DPA est que les distributions d'un bit de sortie de S-Box pour deux hypothèses de sous-clé différentes sont déterministes et peuvent donc très bien être partiellement corrélées. L'exemple suivant est très convaincant sur ce point. Considérons le bit le plus à gauche de la sortie de la $6^{\text{ème}}$ S-Box du DES, lorsque la donnée d'entrée D varie de 0 à 63, vis-à-vis de deux suppositions de sous-clé différentes : MSB(S6($D \oplus 0$)) et MSB(S6($D \oplus 36$))[13]. Les deux séries de bits sont listées à la Table 3.1, avec leur XOR bit à bit sur la dernière ligne :

La troisième ligne contient huit bits à 1, révélant seulement huit erreurs de prédiction parmi les 64. Cet exemple montre qu'une mauvaise supposition de sous-clé, disons 0, peut fournir une bonne prédiction 56 fois sur 64, ce qui n'est pas si éloigné de la bonne supposition de sous-clé 36. Le résultat serait équivalent pour toute autre paire de sous-clés K et $K \oplus 36$. Il en résulte qu'un pic de DPA concurrent apparaîtra au même instant que le pic correct. La faiblesse du contraste perturbera l'ordonnancement des suppositions de sous-clé, spécialement en présence d'un faible rapport signal sur bruit.

[13]Nous remercions ici CÉCILE CANOVAS et JESSY CLÉDIÈRE de nous avoir fait remarquer qu'une erreur s'était glissée dans l'article original au sujet du numéro de la S-Box et de la valeur de la différentielle de sous-clé.

Fait C. La DPA considère implicitement les bits du mot transportés avec le bit cible comme étant uniformément distribués et indépendants du bit cible. Ceci est erroné car l'implémentation introduit un lien déterministe entre leurs valeurs. Leurs contributions asymétriques peut affecter la hauteur et le signe d'un pic de DPA. Cela peut influencer l'analyse, d'une part en rapetissant les pics pertinents, d'autre part en rehaussant ceux qui ne sont pas significatifs. Il existe une astuce bien connue pour contourner cette difficulté, mentionnée par Régis Bevan et Erik Knudsen dans [BK03]. Elle consiste à décaler l'attaque DPA un peu plus loin dans le processus et à effectuer la prédiction juste après la fin du premier tour lorsque la partie droite de la donnée (32 bits) est XOR-ée avec la partie gauche de la sortie de la permutation initiale. Le message étant choisi librement, cela procure une opportunité de ré-équilibrer la perte d'aléa en rafraîchissant la donnée aléatoire. Notons toutefois que l'applicabilité de cette astuce est tout à fait dépendante de la structure de l'algorithme attaqué. Elle peut être menée dans le cas du DES qui, comme réseau de Feistel, donne à l'attaquant l'opportunité de blanchir les sorties des S-Box avec une donnée connue, mais elle pourrait tout aussi bien ne pas être applicable dans le cas d'un autre algorithme comme l'AES. Qui plus est, cela ne résout pas les problèmes soulevés par le *Fait B* dans le cas général.

Pour lever ces ambiguïtés, l'approche basée sur un modèle vise à prendre en compte l'intégralité de l'information. Cela nécessite d'introduire la notion d'implémentation algorithmique que les hypothèses de la DPA permettent d'occulter complètement.

Quand on considère les boîtes de substitution du DES, on ne doit pas perdre de vue que les valeurs de sortie sont sur 4 bits. Bien que ces 4 bits sont en principe équivalents comme bits de sélection de DPA, ils sont accompagnés de 4 autres bits dans le contexte d'un micro-processeur à 8 bits. Des implémentations efficaces exploitent ces 4 bits non utilisés pour économiser de l'espace de stockage dans les environnements contraints comme les composants pour cartes à puce. Une astuce appelée "compression de S-Box" consiste à stocker 2 valeurs de S-Box sur un même octet. L'espace requis est ainsi divisé par deux. Il existe différentes façons d'implémenter cette technique. Considérons par exemple les deux premières S-Box : plutôt que d'allouer deux tables différentes, il est plus efficace de construire la table de lecture suivante : $\mathrm{LUT}_{12}(k) = \mathrm{S\text{-}Box}_1(k) \parallel \mathrm{S\text{-}Box}_2(k)$. Pour un indice d'entrée donné k, l'octet de la table contient les valeurs de deux S-Box adjacentes. D'après le modèle de consommation de la distance de Hamming, la trace de courant devrait varier comme :

- $\mathrm{H}(\mathrm{LUT}_{12}(D_1 \oplus K_1) \oplus R_1)$ lorsque l'on calcule $\mathrm{S\text{-}Box}_1$,
- $\mathrm{H}(\mathrm{LUT}_{12}(D_2 \oplus K_2) \oplus R_2)$ lorsque l'on calcule $\mathrm{S\text{-}Box}_2$.

Lorsque les valeurs sont liées de cette manière, leurs bits respectifs ne peuvent pas être considérés comme indépendants. Pour prouver cette assertion nous avons mené une expérience sur une implémentation à 8 bits qui n'était protégée par aucune contre-mesure anti-DPA. Travaillant en mode "boîte blanche", les paramètres du modèle ont été préalablement déterminés en fonction des traces de consommation mesurées. L'état de référence $R = 0\mathrm{x}B7$ a été identifié comme le code d'opération d'une instruction transférant le contenu de l'accumulateur en RAM par adressage direct. Le modèle s'accordait très bien aux données expérimentales ; leur facteur de corrélation à même atteint 97%. Nous avons donc été capables de simuler la consommation réelle de la sortie de S-Box avec une grande précision. L'étude a alors consisté à appliquer une DPA classique "mono-bit" à la sortie de la S-Box 1 en parallèle sur les deux ensembles de 200 échantillons de données : les consommations de courant mesurées d'une part, et celles simulées d'autre part, c'est-à-dire pour chacune desquelles on a appliqué le modèle identifié au poids de Hamming réel de la donnée.

Comme le montre la Figure 3.3, les biais de DPA simulée et expérimentale s'accordent particulièrement bien. On peut remarquer les points suivants :

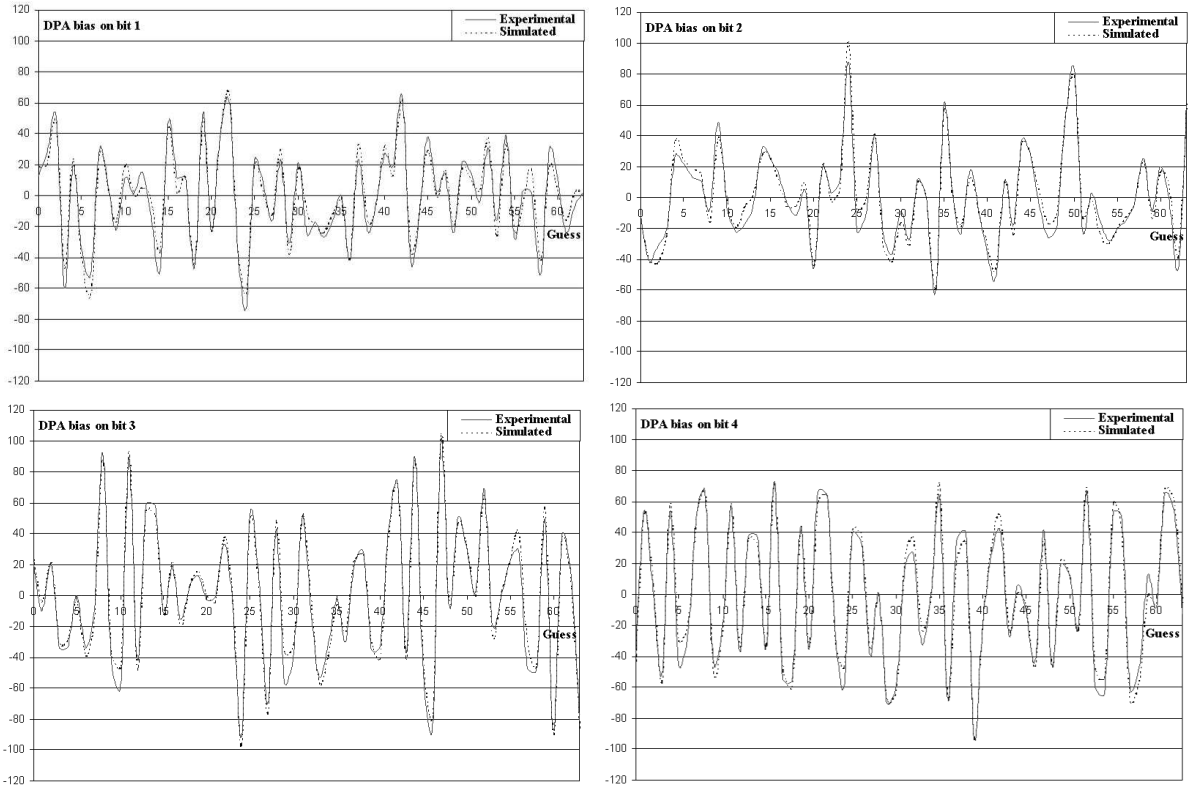


FIG. 3.3 – Biais de DPA sur la S-Box 1 en fonction de la prédiction des bits de sélection 1, 2, 3 et 4, sur des données modélisées et expérimentales. La supposition de sous-clé correcte est 24.

TAB. 3.2 – Taux de corrélation obtenus avec 40 courbes sur les sorties de S-Box d'un DES sur une architecture à 8 bits. Les valeurs des sous-clés sont 24, 19, 8, 8, 5, 50, 43 et 2.

$S\text{-Box}_1$		$S\text{-Box}_2$		$S\text{-Box}_3$		$S\text{-Box}_4$		$S\text{-Box}_5$		$S\text{-Box}_6$		$S\text{-Box}_7$		$S\text{-Box}_8$	
K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}
24	92%	**19**	90%	**8**	87%	**8**	88%	**5**	91%	**50**	92%	**43**	89%	**2**	89%
48	74%	18	77%	18	69%	44	67%	32	71%	25	71%	42	76%	28	77%
01	74%	57	70%	05	68%	49	67%	25	70%	05	70%	52	70%	61	76%
33	74%	02	70%	22	66%	02	66%	34	69%	54	70%	38	69%	41	72%
15	74%	12	68%	58	66%	29	66%	61	67%	29	69%	0	69%	37	70%
06	74%	13	67%	43	65%	37	65%	37	67%	53	67%	30	68%	15	69%

– Les 4 bits de sortie sont loin d'être équivalents.
– La polarité du pic associé à la supposition correcte 24 dépend de la valeur du bit correspondant de l'état de référence. Comme $R = 0\text{x}B7$, son quartet de gauche aligné avec la S-Box 1 est $0\text{x}B = 1011_2$ et seul le bit de sélection n°2 (compté à partir de la gauche) provoque un pic positif alors que les 3 autres subissent une transition de 1 vers 0 menant à un pic négatif.
– De plus ce bit est plutôt chanceux car lorsqu'il est utilisé comme bit de sélection, seule l'hypothèse 50 rivalise avec la sous-clé correcte. Ceci est un cas particulièrement favorable arrivant ici sur la S-Box 1, en partie dû à l'ensemble des 200 messages utilisés. Il ne peut pas être extrapolé aux autres S-Box.
– La dispersion du biais de DPA lorsque la supposition de sous-clé varie peut rendre la situation assez confuse (voir le bit 4).

La qualité de la modélisation prouve que ces faits ne peuvent être attribués au nombre d'acquisitions. Accroître celui-ci bien au delà de 200 n'est d'aucun secours : le niveau des pics en fonction de l'hypothèse n'évolue pas et converge vers le même classement. Ce contre-exemple particulier prouve que l'ambiguïté de la DPA ne réside pas dans une estimation imparfaite, mais plutôt dans des hypothèses de base fausses.

3.7.3 Résultats de la CPA basée sur un modèle

La Table 3.2 fournit le classement des 6 meilleures suppositions classées par taux de corrélation décroissant. Ce résultat est obtenu avec seulement 40 courbes. La clé complète est 11 22 33 44 55 66 77 88 en hexadécimal et les sous-clés correspondantes au premier tour valent respectivement 24, 19, 8, 8, 5, 50, 43 et 2 en représentation décimale.

Cette table montre que la supposition correcte sort toujours du lot avec un fort contraste. Une décision peut donc être prise sans aucune ambiguïté malgré une estimation grossière de ρ_{max}.

Une attaque similaire a aussi été menée sur une implémentation 32 bits, dans un mode "boîte blanche" avec une connaissance parfaite de l'implémentation des boîtes de substitution, et de l'état de référence qui était 0. La clé était 7C A1 10 45 4A 1A 6E 57 en hexadécimal et les sous-clés correspondantes au premier tour valaient 28, 12, 43, 0, 15, 60, 5 et 38 en décimal. Le nombre de courbes est 100. Comme le montre la Table 3.3, le contraste est bon entre la supposition correcte et sa meilleure rivale (environ 40% sur les boîtes 1 à 4). Le taux de corrélation n'est pas si élevé pour les S-Box 5 à 8, probablement à cause d'une modélisation partielle et imparfaite,

TAB. 3.3 – Taux de corrélation obtenus avec 100 courbes sur les sorties de S-Box d'un DES sur une architecture à 32 bits. Les valeurs des sous-clés sont 28, 12, 43, 0, 15, 60, 5 et 38.

S-Box$_1$		S-Box$_2$		S-Box$_3$		S-Box$_4$		S-Box$_5$		S-Box$_6$		S-Box$_7$		S-Box$_8$	
K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}	K	ρ_{max}
28	77%	**12**	69%	**43**	73%	**0**	82%	**15**	52%	**60**	51%	**5**	51%	**38**	47%
19	36%	27	29%	40	43%	29	43%	03	33%	10	34%	15	40%	05	29%
42	35%	24	27%	36	35%	20	35%	58	30%	58	33%	6	29%	55	26%
61	31%	58	27%	06	33%	60	32%	10	30%	18	31%	12	29%	39	25%

mais il s'avère rester exploitable et constituer un indicateur robuste. Quand le nombre de bits par mot machine est plus grand, le contraste entre les suppositions est relativement réhaussé, mais trouver le bon modèle pourrait être plus difficile en mode "boîte noire".

3.8 Conclusion

Notre expérience sur un large éventail de composants pour cartes à puce nous a convaincu de la validité du modèle de la distance de Hamming et des avantages de la méthode CPA sur la DPA, en termes d'efficacité, de robustesse et de nombre d'acquisitions. Une conclusion importante et rassurante est que toutes les contre-mesures conçues contre la DPA offrent la même efficacité défensive contre l'attaque CPA basée sur un modèle. Ce n'est pas si surprenant puisque ces contre-mesures visent à saper les conditions communes sur lesquelles les deux approches sont basées : observabilité du canal auxiliaire, et prédictibilité de variable intermédiaire.

L'inconvénient majeur de la CPA est lié à la caractérisation des paramètres du modèle de fuite. Comme elle est plus exigeante que la DPA, la méthode peut apparaître plus difficile à implémenter. On peut néanmoins objecter que :

- Une analyse statistique du courant, de quelque type qu'elle soit, n'est jamais menée à l'aveugle sans une rétro-conception préliminaire (identification de processus, traçage de bits) : c'est alors l'opportunité pour l'attaquant de quantifier le taux de fuite par CPA sur des données connues.
- La DPA requiert plus de courbes en toutes circonstances puisque les bits de donnée non prédits pénalisent le rapport signal sur bruit (voir [BCO03]).
- Si la DPA échoue par manque de connaissance sur l'implémentation (accroître le nombre de courbes n'aide pas nécessairement), nous avons montré comment inférer une partie de cette information sans effort excessif : par exemple l'état de référence doit être trouvé par recherche exhaustive une seule fois en général.
- Il existe de nombreuses situations où les variantes d'implémentation (comme le plan mémoire des S-Box du DES) ne sont pas si nombreuses à cause de contraintes opérationnelles.
- Si une partie du modèle ne peut pas être inférée (implémentation des S-Box du DES, crypto-processeur), alors la corrélation partielle avec la partie connue du modèle peut tout de même fournir des informations exploitables.

Finalement la DPA reste pertinente dans les cas d'architectures très spéciales pour lesquelles le modèle peut être complètement hors d'atteinte, comme dans certains crypto-processeurs câblés.

Chapitre 4

De l'implémentation d'un algorithme rapide de génération de premiers

Sommaire

Nous présentons dans ce chapitre une analyse simple du courant sur un algorithme de génération de premiers à bord d'une carte à puce. Ce travail a été publié à Ches '07 avec Jean-Sébastien Coron [CC07].

Quoique rarement utilisée dans ce but, l'analyse des canaux auxiliaires peut également s'appliquer aux algorithmes de génération de clés. Dans ce chapitre, nous montrons que, lorsqu'il n'est pas correctement implémenté, l'algorithme rapide de génération de premiers proposé par Marc Joye et Pascal Paillier à Ches '06 [JP06] est vulnérable à une analyse des canaux auxiliaires. Son application principale est la génération de paires de clés RSA pour des plateformes embarquées comme les cartes à puce. Notre attaque suppose qu'un bit de parité peut être retrouvé par SPA quand il apparaît dans un branchement conditionnel. Elle peut être combinée au théorème de Coppersmith pour améliorer son efficacité, et nous montrons que pour une taille de module de 1024 bits, on peut retrouver la factorisation d'environ $\frac{1}{1000}$ des modules RSA construits. Nous proposons des contre-mesures empêchant notre attaque en rendant l'observation du bit de parité impossible ou inexploitable.

4.1 Introduction

Les analyses des canaux auxiliaires, telles que l'*analyse simple du courant* (SPA) ou l'*analyse différentielle du courant* (DPA) [KJJ99], se concentrent généralement sur les phases de chiffrement ou de déchiffrement, rarement sur la phase de génération de la clé. De fait, le chiffrement ou le déchiffrement offrent plus de souplesse à l'attaquant qui peut fournir en entrée des messages variés et enregistrer à chaque fois une fuite de canal auxiliaire. Par contraste, un algorithme de génération de clés n'admet aucune entrée variable connue de l'attaquant, et généralement

l'exécuter plusieurs fois n'apporte rien à l'attaquant puisqu'une clé différente est calculée à chaque nouvelle exécution.

Dans ce chapitre, nous montrons que, s'il n'est pas correctement implémenté, un des algorithmes rapides de génération de premiers proposés par Marc Joye et Pascal Paillier à Ches '06 [JP06] est vulnérable à une analyse des canaux auxiliaires. L'application principale de l'algorithme de Joye-Paillier est de générer des clés RSA sur des plate-formes embarquées comme les cartes à puce, où l'efficacité est d'une importance cruciale. La génération d'un premier se fait généralement en appliquant un test de primalité sur des entiers générés aléatoirement, jusqu'à ce qu'un premier soit trouvé. La technique décrite dans [JPV00, JP06] consiste à générer aléatoirement des entiers qui sont non divisibles par les petits premiers p_i ; un premier apparaîtra alors avec une plus forte probabilité, et moins de tests de primalité devront être effectués en moyenne, améliorant ainsi l'efficacité.

Dans une variante plus rapide, décrite dans [JP06], une séquence de candidats est générée à partir d'une graine aléatoire et la parité de chaque candidat est testée avant d'appliquer un test de primalité, jusqu'à ce qu'un premier soit trouvé. Dans ce chapitre, nous nous concentrons sur une implémentation de cette variante, et montrons que si n tests de primalité ont été effectués et si les bits de parité peuvent être obtenus par SPA, il est alors possible de retrouver les $(n-1)$ bits les moins significatifs du premier généré. Le théorème de Coppersmith [Cop97] montre qu'un module RSA $N = pq$ peut être factorisé en temps polynomial étant donné la moitié des bits les moins significatifs (ou les plus significatifs) de p. En conséquence, si le nombre n de tests de primalité pour p ou pour q est plus grand que la moitié de la taille en bits du premier, on peut facilement retrouver la factorisation de N. Nous présentons une analyse qui montre que pour certains paramètres ($b_{min} = b_{max} = 0$) et pour des modules de 1024 bits, cela arrive pour 10^{-3} des modules RSA générés.

4.2 L'algorithme de génération de premiers de Joye-Paillier

L'algorithme de Joye-Paillier consiste à générer une séquence de candidats q qui sont premiers avec les petits premiers p_i ; un test de primalité $T(q)$ est alors effectué sur chaque candidat jusqu'à ce qu'un premier soit trouvé. Nous nous concentrons ici sur la variante la plus rapide (présentée à la Figure 3 dans [JP06]). On définit Π comme étant le produit des r premiers premiers, $p_1 = 2$ exclu, si bien que Π est impair :

$$\Pi = \prod_{i=2}^{r} p_i \ .$$

Soit $[q_{min}, q_{max}]$ l'intervalle dans lequel les entiers premiers doivent être générés. On définit des entiers b_{min}, b_{max} et v tels que :

$$q_{min} \leqslant (v + b_{min})\Pi, \text{ et } \ (v + b_{max})\Pi + \Pi - 1 \leqslant q_{max} \ .$$

Les entiers premiers seront en réalité générés dans le sous-intervalle :

$$[(v + b_{min})\Pi, (v + b_{max} + 1)\Pi - 1] \ .$$

Le lecteur pourra se référer à [JP06] pour plus de détails sur la sélection des paramètres b_{min}, b_{max} et v. Nous notons $T(q)$ un test de primalité, par exemple le test probabiliste de Miller-Rabin [Mil76].

Algorithme 4.1 Algorithme rapide de génération de premiers [JP06]

Entrée : Π impair, b_{min}, b_{max}, v

Sortie : un premier aléatoire $q \in [q_{min}, q_{max}]$

1. $\ell \leftarrow v \cdot \Pi$
2. choisir aléatoirement $k \in (\mathbb{Z}/\Pi\mathbb{Z})^*$
3. choisir aléatoirement $b \in \{b_{min}, \ldots, b_{max}\}$ et poser $t \leftarrow b \cdot \Pi$
4. $q \leftarrow k + t + \ell$
5. **si** (q est pair) **alors**
6. $\quad q \leftarrow \Pi - k + t + \ell$
7. **fin si**
8. **si** ($T(q) =$ `faux`) **alors**
9. $\quad k \leftarrow 2k \mod \Pi$
10. $\quad$ **aller à l'étape** 4
11. **fin si**
12. **retourne** q

FIG. 4.1 – Algorithme rapide de génération de premiers [JP06].

La Figure 4.1 présente l'algorithme que nous étudions. On remarque facilement que le candidat q à l'étape 8 est impair et premier avec tous les premiers dans Π. Précisément, nous avons $q = \pm k \mod \Pi$ et k reste toujours dans $(\mathbb{Z}/\Pi\mathbb{Z})^*$, car $2 \in (\mathbb{Z}/\Pi\mathbb{Z})^*$. Cela implique que q est premier avec tous les premiers dans Π. De plus, si $q = k + t + \ell$ est pair à l'étape 4, alors $q' = \Pi - k + t + \ell = \Pi - 2k + q$ doit être impair puisque Π est impair, et donc q est impair à l'étape 8. Puisque nous assurons que chaque candidat q est non divisible par les petits premiers, q possède une plus grande probabilité d'être premier, et l'on obtient donc un algorithme de génération plus rapide (voir [JP06] pour une analyse complète).

4.3 Notre attaque par canaux auxiliaires

Notre attaque se fonde sur l'hypothèse que nous pouvons retrouver la parité de k à l'étape 4 grâce au test de parité effectué sur q à l'étape 5. Précisément, sur une implémentation concrète, un branchement conditionnel produit généralement une fuite physique différente en fonction du résultat du test. En mesurant la consommation de courant, l'attaquant peut donc être capable de déterminer si l'opération $q \leftarrow \Pi - k + t + \ell$ a été effectuée ou non, ce qui lui donne le bit de parité sur q. Notre attaque est donc une *analyse simple du courant* (SPA) sur le bit de parité de q. Dans la pratique, cette hypothèse peut être plus ou moins réaliste en fonction du microprocesseur utilisé, de la présence éventuelle de contre-mesures au niveau matériel, et de la façon dont le test est implémenté. Notons que des attaques SPA basées sur des hypothèses analogues ont été décrites dans [DK05, FKM+06].

Nous montrons maintenant que la séquence de bits de parité sur q nous permet de retrouver les bits les moins significatifs du premier q retourné en sortie. Notre attaque est basée sur le lemme simple suivant :

Lemme 1. *Soit* Π *un entier impair, et soit* $k_0 \in \mathbb{Z}_\Pi$. *Définissons les séquences* $B_i = \lfloor 2^i k_0/\Pi \rfloor$

et $k_i = 2^i\, k_0 \bmod \Pi$. *Alors, pour* $i \geqslant 1$, $B_i = \sum_{j=1}^{i} (k_j \bmod 2) 2^{i-j}$.

Démonstration. Par définition, nous avons $2^{i-1}\, k_0 = B_{i-1}\,\Pi + k_{i-1}$. D'où $2^i\, k_0 = 2B_{i-1}\,\Pi + 2k_{i-1} = (2B_{i-1} + \lfloor 2k_{i-1}/\Pi \rfloor)\Pi + k_i$, qui donne $B_i = 2B_{i-1} + \lfloor 2k_{i-1}/\Pi \rfloor$. Il s'en suit que $B_i = \sum_{j=1}^{i} \lfloor 2k_{j-1}/\Pi \rfloor 2^{i-j}$ (notons que $B_0 = 0$).

De plus, nous avons $2k_{j-1} = \lfloor 2k_{j-1}/\Pi \rfloor \Pi + k_j$. Considérant la relation modulo 2, nous obtenons $\lfloor 2k_{j-1}/\Pi \rfloor \equiv k_j \pmod 2$ puisque Π est impair, et puisque $2k_{j-1} < 2\Pi$, nous avons $\lfloor 2k_{j-1}/\Pi \rfloor = k_j \bmod 2$, ce qui conclut la preuve. □

Proposition 1. *Avec les notations précédentes, et posant* $b_j = k_j \bmod 2$, *nous obtenons que* $k_i \equiv -(\sum_{j=1}^{i} b_j\, 2^{i-j})\, \Pi \pmod{2^i}$.

Démonstration. Cela se déduit immédiatement du lemme précédent en observant que $B_i\,\Pi = 2^i\, k_0 - k_i$. □

Nous supposons que les paramètres Π, b_{min}, b_{max} et v sont publics et connus de l'attaquant. Soit $k_i = k_0 \cdot 2^i \mod \Pi$ pour $i \geqslant 0$ la séquence des entiers k qui apparaissent à l'étape 4, où k_0 est l'entier initialement généré à l'étape 2. Soit $q_i = k_i + t + \ell$ l'entier correspondant calculé à l'étape 4. À partir de la parité du q_i testé à l'étape 4, on peut obtenir la parité de k_i ; pour cela on fait une supposition sur la parité de $t + \ell$, qui est constant après l'étape 3. Alors en utilisant le lemme précédent, après $(n+1)$ tests de primalité[14], on obtient la valeur de $k_n \bmod 2^n$. Nous pouvons alors écrire :

$$k_n = 2^n \cdot x + x_0 \ ,$$

où $0 \leqslant x_0 < 2^n$ est connu et x est inconnu. Le premier généré est donc égal à :

$$q \ = 2^n \cdot y + y_0 + b \cdot \Pi + \ell$$

$$\text{avec} \quad (y, y_0) = \begin{cases} (x, x_0) & \text{si } q = k_n + t + \ell \\ (\lfloor \frac{\Pi}{2^n} \rfloor - x, (\Pi \bmod 2^n) - x_0) & \text{si } q = \Pi - k_n + t + \ell \end{cases} ,$$

où y et $b \in [b_{min}, b_{max}]$ sont des entiers inconnus de l'attaquant, et les entiers n, y_0, Π et ℓ sont connus.

Nous faisons l'hypothèse suivante : nous supposons que $b_{min} = b_{max} = 0$, de sorte que $b = 0$. Notons que poser $b_{min} = b_{max} = 0$ est un choix de paramètres valide dans l'algorithme de Joye-Paillier. Dans ce cas, l'entier q peut être écrit :

$$q \ = \ 2^n \cdot y + C \ ,$$

où C est une constante connue et y est un entier inconnu.

Théorème 1 (Coppersmith [Cop97]). *Étant donnés* $N = pq$ *et les* $\frac{1}{4} \log_2 N$ *bits de poids faible ou de poids fort de* p, *on peut retrouver la factorisation de* N *en temps polynomial en* $\log N$.

En utilisant le théorème de Coppersmith, on peut donc retrouver la factorisation de N en temps polynomial si le nombre n de tests de primalité est au moins la moitié de la taille en bits de p. Soit α la probabilité qu'un entier aléatoire impair q de n_0 bits, premier avec Π, soit premier. Nous faisons l'approximation heuristique que les candidats q_i se comportent comme

[14] On a besoin de $(n+1)$ bits de parité car on n'utilise pas le premier.

TAB. 4.1 – Taille du module RSA, taille en bits n_0 des premiers p et q, nombre r de premiers p_i dans Π, nombre moyen $E[X]$ de tests de primalité, et fraction δ de modules RSA faibles.

RSA	n_0	r	E[X]	δ
512 bits	256	43	18.7	1.9×10^{-3}
768 bits	384	60	26.1	1.2×10^{-3}
1024 bits	512	75	33.3	8.4×10^{-4}
2048 bits	1024	131	60.0	3.7×10^{-4}

s'ils étaient indépendants et uniformément distribués. De l'analyse de [JPV00], nous obtenons qu'un candidat q_i est premier avec probabilité :

$$\alpha = \frac{1}{n_0 \cdot \ln 2} \cdot \frac{\Pi'}{\varphi(\Pi')} \quad ,$$

où $\Pi' = 2\Pi$, et φ est la fonction indicatrice d'EULER. Posant X la variable aléatoire qui donne le nombre de tests de primalité, nous avons donc :

$$\Pr[X = i] = (1 - \alpha)^{i-1} \cdot \alpha \quad ,$$

ce qui donne $E[X] = 1/\alpha$ et :

$$\Pr[X \geqslant i] = (1 - \alpha)^{i-1} \quad .$$

Pour résumer, si l'attaque est menée à la fois sur p et q, la factorisation d'un module RSA de $2n_0$ bits peut être retrouvée en temps polynomial pour une fraction :

$$\delta \simeq 2 \cdot \left(1 - \frac{1}{n_0 \cdot \ln 2} \cdot \frac{\Pi'}{\varphi(\Pi')}\right)^{n_0/2-1}$$

des modules générés. Nous fournissons à la Table 4.1 la fraction correspondante pour différentes tailles de modules. Pour une taille de premier de n_0 bits, nous considérons le plus grand r tel que :

$$\Pi = \prod_{i=2}^{r} p_i < 2^{n_0-1} \; .$$

Nous prenons alors $v = \lfloor 2^{n_0}/\Pi \rfloor - 1$, puis $q_{min} = v \cdot \Pi$ et $q_{max} = (v+1) \cdot \Pi - 1$, avec $b_{min} = b_{max} = 0$; c'est un choix de paramètres valide pour l'algorithme de JOYE-PAILLIER. La Table 4.1 montre que pour une taille de module de 1024 bits, notre attaque par canaux auxiliaires permet de factoriser une fraction $\delta \simeq 8.4 \cdot 10^{-4}$ des modules RSA générés. Si l'algorithme n'est exécuté qu'une seule fois à l'intérieur d'une carte à puce, cela signifie concrètement qu'une fraction $8.4 \cdot 10^{-4}$ des cartes peuvent être "cassées".

Notons que l'hypothèse $b_{min} = b_{max} = 0$ peut être allégée à des valeurs petites (connues) de b_{min} et de b_{max} ; dans ce cas, la valeur $b \in [b_{min}, b_{max}]$ doit être recherchée exhaustivement[15].

[15] Alternativement, on pourrait essayer de dériver une variante du théorème de COPPERSMITH pour des premiers de la forme $q = 2^n \cdot x + b \cdot \Pi + C$, avec x et b inconnus, et des constantes Π et C connues.

4.4 Contre-mesures

Bien que notre attaque ne permet de casser qu'une faible proportion des clés générées, nous pensons qu'il est nécessaire de s'en protéger. Nous discutons dans cette section de trois contre-mesures possibles.

La première contre-mesure est de re-générer périodiquement une nouvelle graine k de sorte que l'attaquant n'obtienne pas assez d'information sur le premier q. Soit $s > 0$ le nombre maximum de tests de primalité effectués pour une graine donnée k. L'entier s doit être suffisamment petit pour empêcher l'attaque précédente, mais pas trop petit pour garder le même niveau d'efficacité ; nous proposons de prendre $s = \frac{n_0}{4}$ où n_0 est la taille en bits du premier à générer. On obtient alors l'algorithme modifié de la Figure 4.2.

Algorithme 4.2 Algorithme modifié de génération de premiers

Entrée : s, Π impair, b_{min}, b_{max}, v

Sortie : un premier aléatoire $q \in [q_{min}, q_{max}]$

1. $\ell \leftarrow v \cdot \Pi$
2. $i \leftarrow 0$
3. choisir aléatoirement $k \in (\mathbb{Z}/\Pi\mathbb{Z})^*$
4. choisir aléatoirement $b \in \{b_{min}, \ldots, b_{max}\}$ et poser $t \leftarrow b \cdot \Pi$
5. $q \leftarrow k + t + \ell$
6. **si** (q est pair) **alors**
7. $\quad q \leftarrow \Pi - k + t + \ell$
8. **fin si**
9. **si** ($T(q) =$ `faux`) **alors**
10. $\quad k \leftarrow 2k \mod \Pi$
11. $\quad i \leftarrow i + 1$
12. $\quad$ **si** ($i < s$) **alors**
13. $\quad\quad$ **aller à l'étape** 5
14. $\quad$ **sinon**
15. $\quad\quad$ **aller à l'étape** 2
16. $\quad$ **fin si**
17. **fin si**
18. **retourne** q

FIG. 4.2 – Algorithme modifié de génération de premiers.

La deuxième contre-mesure consiste à remplacer l'instruction $k \leftarrow 2 \cdot k \mod \Pi$ par $k \leftarrow 2^t \cdot k \mod \Pi$ où t est un petit entier aléatoire. Si $k \leftarrow 2^t \cdot k \mod \Pi$ est implémenté en temps constant, alors l'attaquant ne sait pas pour quels entiers $k_i = 2^i \cdot a \mod \Pi$ il obtient les bits de parité, ce qui empêche l'attaque précédente. Cette seconde contre-mesure est efficace car le temps d'exécution additionnel est probablement négligeable comparé au temps d'exécution du test de primalité.

La troisième contre-mesure consiste à implémenter le test de manière à ce qu'il ne "fuie" pas. La façon classique est de calculer les deux issues du branchement et de sélectionner le résultat correct. Cette contre-mesure est analogue aux contre-mesures *élever au carré et multiplier systématiquement* (en anglais, *square and multiply always*) de l'exponentiation RSA et *doubler et*

additionner systématiquement (en anglais, *double and add always*) de la multiplication scalaire ECC (voir [Cor99]). Les instructions :

6. **si** (q est pair) **alors**
7. $\quad q \leftarrow \Pi - k + t + \ell$
8. **fin si**

sont alors remplacées par :

6. $u \leftarrow q \mod 2$
7. $A[0] \leftarrow \Pi - k + t + \ell$
8. $A[1] \leftarrow q$
9. $q \leftarrow A[u]$

Dans ce cas le calcul de $\Pi - k + t + \ell$ est toujours effectué et il est donc probablement plus difficile pour un attaquant de retrouver la parité de q.

4.5 Conclusion

Nous avons montré qu'une implémentation impropre de l'algorithme de génération de premiers de Joye-Paillier peut succomber à une analyse de canaux auxiliaires. Notre attaque est basée sur l'hypothèse qu'une analyse simple du courant peut révéler les valeurs successives d'un bit de parité qui apparaît dans un branchement conditionnel. Nous avons montré que pour certains paramètres ($b_{min} = b_{max} = 0$) et pour des modules RSA de 1024 bits, un fraction 10^{-3} de ces modules peuvent être factorisés efficacement. Cependant, en pratique, certains des bits de parité obtenus par SPA peuvent être erronés. Un problème ouvert intéressant est donc de trouver une attaque qui fonctionne en présence de telles erreurs ; formellement :

Problème ouvert : Soit $\beta > 0$, soit Π un entier impair, et soit $k_0 \in \mathbb{Z}_\Pi$. Définissons la séquence $k_i = 2^i\, k_0 \bmod \Pi$ et $b_i = k_i \mod 2$ pour $i \geqslant 1$. Soit $b'_i = b_i$ avec probabilité $1 - \beta$ et $b'_i = 1 - b_i$ sinon, indépendamment pour chaque $i \geqslant 1$. Étant donnée la séquence b'_i pour $i \geqslant 1$, trouver k_0 en temps polynomial en $\log \Pi$.

Chapitre 5

Pourquoi doit-on sécuriser également les éléments publics d'une clé RSA

Sommaire

Ce chapitre décrit plusieurs idées d'attaques par fautes sur le RSA, basées sur une exponentiation modulaire calculée avec une valeur erronée du module. Ce travail, réalisé conjointement avec Éric Brier, Benoît Chevallier-Mames et Mathieu Ciet, a été publié à Ches '06 [BCCC06].

Il est bien connu qu'un adversaire malicieux peut essayer de retrouver de l'information secrète en provoquant une faute durant des opérations cryptographiques. À la suite du travail de Jean-Pierre Seifert [Sei05] sur les injections de fautes durant la vérification de signature RSA, nous considérons ici son équivalent pour la signature.

Nous présentons la première attaque par fautes qui s'applique au RSA en mode standard et qui permet de retrouver l'exposant privé en ne corrompant qu'un élément public de la clé. En effet, de manière similaire à l'attaque de Jean-Pierre Seifert, notre attaque est réalisée en ne modifiant que le module.

Un des points forts de notre attaque est que les hypothèses faites sur les effets de la faute provoquée sont assouplies. Dans un certain mode, *absolument aucune connaissance* sur l'effet de la faute n'est requise pour parvenir à une révélation complète de l'exposant privé. Dans un autre mode, basé sur un modèle de faute définissant ce que nous appelons un *dictionnaire*, l'efficacité de l'attaque est accrue et le nombre de fautes est considérablement réduit.

Notons que ces attaques fonctionnent même contre des implémentations avec des paddings déterministes (*i.e.* RSA-FDH) ou aléatoires (*i.e.* RSA-PFDH), à l'exception des cas où les signatures prévoient une récupération de l'aléa (comme dans RSA-PSS).

En fin de compte, les résultats présentés ici nous amènent à conclure qu'il est tout aussi impératif de protéger les paramètres RSA publics que les paramètres privés contre les attaques par fautes. C'est la raison pour laquelle nous mentionnons, après la description de nos attaques, quelques idées de contre-mesures permettant de s'en protéger.

5.1 Introduction

5.1.1 Rudiments

RSA [RSA78] est aujourd'hui le crypto-système à clé publique le plus largement utilisé. Soit $n = pq$ le produit de deux grands entiers premiers typiquement de 512 ou 1024 bits. Soit e l'exposant public, premier avec $\varphi(n) = (p-1)(q-1)$, où $\varphi(\cdot)$ est la fonction indicatrice d'Euler. L'exposant public est lié à ce qui est appelé l'exposant privé d par l'équation $ed \equiv 1 \pmod{\varphi(n)}$.

À la base, dans le crypto-système RSA [BR93, BR96, PKCS-1], les opérations publiques (c'est-à-dire, la vérification de signature ou le chiffrement) sont réalisées en calculant une puissance $e^{\text{ème}}$, alors que les opérations privées (c'est-à-dire, la génération de signature ou le déchiffrement) sont réalisées en calculant une puissance $d^{\text{ème}}$. Pour accélérer les opérations privées, une technique efficace basée sur le *théorème chinois des restes* a été proposée [QC82] : cela est appelé le mode CRT, par opposition au mode standard.

RSA et les attaques physiques.

La sécurité du crypto-système à clé publique RSA est liée à la difficulté de la factorisation. De plus, quand on implémente les crypto-systèmes, on doit être très attentif aux fuites d'information qui prêtent le flanc aux *analyses de canaux auxiliaires* [KJJ99].

En 1996, Dan Boneh, Richard DeMillo et Richard Lipton [BDL97] ont introduit un nouveau type d'attaques, les *analyses de fautes* (en anglais, *Fault Analysis*, FA). Leur idée, connue sous le nom d'*attaque Bellcore*, s'applique à l'implémentation CRT du RSA. La provocation d'une seule faute sur une moitié du calcul suffit pour retrouver la factorisation du module à partir d'une signature correcte et d'une signature fautée, par un simple calcul de plus grand commun diviseur. Notons cependant que l'attaque Bellcore ne s'applique pas dans le cas d'utilisation d'un padding aléatoire.

Aujourd'hui, dans le cas du RSA standard, on ne connaît qu'une seule attaque par injection de faute qui permette de retrouver l'exposant privé. Cette attaque repose sur le basculement individuel des bits[16] de l'exposant privé.

[16] Cette attaque peut être généralisée pour modifier de petits ensembles de bits, typiquement des octets.

Les types de paramètres fautés.

Toutes les méthodes précédentes sont basées sur l'injection de fautes contre des paramètres privés.[17] Une exception est présentée dans un article récemment publié par Jean-Pierre Seifert [Sei05], dans lequel il propose pour la première fois d'attaquer la vérification de signature RSA. Le schéma RSA lui-même n'est pas menacé, car l'attaquant n'est pas capable de falsifier de nouvelles signatures, mais l'attaque de Seifert lui permet de passer — avec une certaine probabilité — l'étape de vérification de signature, pour un message de son choix, en corrompant le module public. L'objectif de l'attaquant est atteint, mais l'attaque nécessite d'être reproduite à chaque fois pour obtenir une nouvelle acceptation indue.

5.1.2 Notre contribution

Dans ce chapitre nous proposons la première attaque par fautes qui peut être utilisée contre la signature RSA en mode standard, permettant de retrouver l'exposant privé en ne corrompant que des éléments *publics* de la clé. Ce point est particulièrement important, car les autres attaques connues visent l'exposant privé, qui devrait donc, par essence, être protégé contre les fautes. La nécessité de protéger les éléments publics demeurait une question ouverte jusqu'à présent. Notre contribution aboutit donc clairement à la conclusion que *les éléments de clé publique* RSA *doivent aussi être protégés contre les attaques par fautes.*

Notre attaque a le même point de départ que celle de Seifert : elle consiste à corrompre le module public. Cependant, l'attaque de Seifert permet à l'attaquant de passer une vérification de signature (avec une certaine probabilité), alors que la nôtre permet une révélation complète de la clé. Une fois la clé connue, l'adversaire obtient les pleins pouvoirs, alors que l'attaque de Seifert ne permet qu'une unique acceptation indue.

Une autre propriété essentielle de notre attaque est que, dans un de ses modes, l'attaquant n'a besoin d'*absolument aucune connaissance* sur l'effet de la faute. Cela est une amélioration certaine de l'attaque de Seifert (dans laquelle l'attaquant doit *supposer* la valeur fautée du module), ou sur les attaques par basculement de bit (qui exigent une précision irréaliste en pratique).

Dans un autre mode, notre attaque peut être améliorée. Grâce à un modèle de faute, nous sommes capables de réduire considérablement le nombre de fautes nécessaires pour retrouver complètement la clé privée. Comme expliqué plus loin, la connaissance de la faute que produit l'attaquant peut être probabiliste ou imprécise : certaines des phases de l'attaque permettent d'accepter une certaine incertitude quant à l'effet de la faute.

Les nouvelles attaques par fautes que nous présentons ici s'appliquent au RSA standard pour lequel il n'était pas connu d'attaque aussi efficace que pour le mode CRT. De plus, des paddings fixes (*i.e.* RSA-FDH [BR93]) ou des paddings aléatoires avec aléa joint (*i.e.* RSA-PFDH [Cor02]) n'influencent pas nos attaques. La seule limitation concerne le cas d'une signature avec récupération d'aléa (*i.e.* RSA-PSS [BR96]) pour lequel le problème reste ouvert.

5.1.3 Plan de l'exposé

La description de notre travail est organisée de la manière suivante. Nous rappelons dans la Section 5.2 le contexte des attaques par fautes, et la nouveauté introduite par l'attaque de Seifert. La partie principale de notre contribution commence à la Section 5.3 où nous définissons

[17] Provoquer des fautes contre une méthode publique a néanmoins été déjà considéré dans le cas des courbes elliptiques [BMM00, CJ05].

le cadre général de notre attaque. Nous introduisons alors à la Section 5.4 le premier mode de notre attaque, dans lequel l'adversaire n'a besoin d'aucune connaissance particulière sur l'effet de la faute. La Section 5.5 décrit une adaptation de notre attaque au cas où un modèle de faute est accessible à l'adversaire. Enfin, nous concluons à la Section 5.6.

5.2 Préliminaires

Dans ce chapitre, la notation $DL(\mu, s, n)$ est utilisée pour exprimer le logarithme discret de s en base μ modulo n qui, soit est un entier défini modulo l'ordre multiplicatif de μ modulo n, soit n'existe pas dans le cas où s n'est pas une puissance de μ modulo n. Clairement, cette notation peut être généralisée à toute puissance de premier p^a divisant n, et à tout entier r divisant l'ordre multiplicatif de μ modulo p^a, par $DL(\mu, s, p^a) \bmod r$ (noté $DL(\mu, s, p^a, r)$ par la suite), qui est un entier défini modulo r ou n'existe pas.

Nous rappelons que pour des valeurs relativement petites de r — disons de 15 à 20 chiffres —, le logarithme discret $DL(\mu, s, p^a, r)$ peut être calculé efficacement par des méthodes de complexité en racine carrée telles que les *pas-de-bébé/pas-de-géant* ou la méthode *rho* de Pollard [MOV97].

5.2.1 Modèles de fautes

Les attaques basées sur les fautes peuvent être réalisées en pratique de différentes façons. Par le passé, il était possible sur certains composants de produire des fautes par le biais de défaillances subites de l'alimentation [ABF+02]. De nos jours les composants sont conçus pour résister à de tels moyens de provocation de fautes.

Le meilleur outil aujourd'hui pour injecter une faute est certainement le laser [BCN+06]. Les effets de la faute peuvent varier en fonction du composant, du type de laser utilisé, ou des différents mécanismes de protection implémentés par les concepteurs du composant. Divers modèles de faute sont communément considérés en fonction des capacités supposées de l'attaquant en termes de positionnement et de précision temporelle de ses fautes.

D'un point de vue pratique, l'effet de la faute est très dépendant du composant. La faute la plus simple a pour effet de changer la valeur d'un mot (dont la taille dépend de l'architecture) d'une manière indéterminée. Cela peut être obtenu par exemple en provoquant une faute sur les décodeurs d'adresse, lorsque des paramètres stockés en mémoire non volatile (en anglais, *non-volatile memory*, NVM) sont lus et transférés en RAM. Si ce transfert se fait en ordre aléatoire, alors le positionnement en terme d'indice du mot est alors inconnu également.

Pour certains composants, l'effet de la faute peut être connu, éventuellement avec une certaine probabilité. Dans la littérature, des modèles à basculement individuel de bit sont parfois considérés. Ce n'est cependant pas si facile à réaliser dans la pratique, alors que les modèles de mots corrompus sont très réalistes. Également, une distinction est faite entre les fautes permanentes (valeurs de bits "gravées" en NVM) et les fautes transitoires. Dans la suite nous ne considérons que le cas des fautes transitoires. Nous ne faisons que peu d'hypothèses sur les capacités d'injection de l'attaquant, de sorte que nos attaques sont compatibles avec les modèles de fautes les plus réalistes.

5.2.2 L'attaque de Seifert et Muir

Avant d'aller plus loin, donnons d'abord une brève description de l'article de Jean-Pierre Seifert [Sei05] qui a motivé notre travail, et de sa généralisation par James Muir [Mui06].

Dans un souci de simplicité, cette attaque sera appelée "attaque de SEIFERT" dans le reste de ce chapitre. Nous renvoyons le lecteur intéressé aux articles originaux pour plus de détails.

Le principe de base de l'attaque de SEIFERT est le suivant : dans un premier temps, et sans que cela ne nécessite la possession du dispositif attaqué, l'attaquant essaie de trouver un module fauté n' tel que l'exposant public e et $\varphi(n')$ sont premiers entre eux, et tel que n' est une valeur fautée *possible* ou *plausible* du module n. L'attaquant doit pour cela faire appel à un modèle de faute. La valeur de n' doit être facilement factorisable afin de pouvoir calculer efficacement d', l'inverse de e mod $\varphi(n')$. Dès que d' est calculé, l'attaquant construit une signature $s' = \mu^{d'} \bmod n'$.

Cette première opération, qui consiste à essayer de trouver un n' qui satisfasse une propriété utile et à construire une signature "fautée" correspondante, est effectuée avant l'attaque. Dès lors, une phase "on-line" peut être menée : l'attaquant exécute l'algorithme de vérification de signature avec (s', μ) en entrée, et tente d'injecter une faute durant le chargement du module de manière à effectuer les calculs modulo le n' visé plutôt que modulo n. Clairement, la probabilité de succès, et donc le nombre moyen de fautes nécessaires, dépend de la précision du modèle de faute et de la capacité de l'attaquant à produire une faute assez précise pour obtenir le module fauté n' avec une probabilité non négligeable.

5.3 Cadre de nos extensions à l'attaque de SEIFERT

L'attaque de SEIFERT réussit à falsifier une signature qui est acceptée comme valide, mais ne révèle aucune information sur les éléments privés de la clé. Un certain accès indu peut être obtenu, mais la clé RSA elle-même n'est pas cassée.

Nous présentons maintenant des extensions à l'attaque de SEIFERT. Elles permettent à un attaquant de retrouver l'exposant privé d à partir de plusieurs calculs fautés alors que le module est corrompu avant une exponentiation RSA standard.

5.3.1 Description générale et contraintes de notre attaque

Méthodologie générale

De manière similaire à [Sei05, Mui06], notre attaque par fautes consiste à modifier le module avant une exponentiation RSA. L'opération $s = \mu^d \bmod n$ est visée, et différentes fautes sont provoquées afin de collecter des signatures corrompues à partir desquelles l'attaquant apprend l'exposant privé d.

Définition 1 (Campagne d'attaque par fautes, couples fautés)**.** *On dira qu'un attaquant mène la* campagne d'attaque par fautes *s'il exécute K fois l'exponentiation $s = \mu^d \bmod n$, et corrompt ces exécutions en modifiant le module n en modules inconnus n'_i, pour obtenir des* couples fautés $(\mu_i, s_i)_{1 \leqslant i \leqslant K}$.

Paddings

Une contrainte générale provient de l'utilisation du RSA dans la vie réelle : il est bien connu que l'on doit utiliser, avant l'exponentiation, des fonctions (appelées *paddings*, et notées Λ) qui réduisent la malléabilité du RSA. Certains des paddings sont *déterministes* — c'est-à-dire, $\mu = \Lambda(m)$ —, d'autres sont *probabilistes* — c'est-à-dire, $\mu = \Lambda(m, u)$. Dans le cas probabiliste, l'aléa u peut être soit joint, soit auto-déterminé.

À cause de la vérification de redondance du padding après la phase de déchiffrement (*i.e.* dans RSA-OAEP), l'exploitation des attaques par fautes pendant le déchiffrement n'est généralement

pas possible et donc le déchiffrement ne sera pas abordé ici. Concernant les signatures, des attaques par fautes sont possibles si μ_i est connu de l'attaquant. C'est le cas lorsque le padding est déterministe (*i.e.* RSA-FDH), ou lorsque l'aléa est joint à la signature (*i.e.* RSA-PFDH). Au contraire, si l'aléa est auto-déterminé à partir de la signature (*i.e.* RSA-PSS), alors le résultat fauté ne permet pas de retrouver μ_i et notre attaque ne fonctionne pas.

À partir de maintenant nous supposons que l'attaquant peut calculer les bases μ_i utilisées durant les exponentiations fautées.

5.3.2 Dictionnaire de modules

La littérature décrit de nombreux modèles de fautes (voir Section 5.2.1) qui peuvent permettre à l'adversaire de supposer comment il peut avoir modifié le module n en n'_i durant les exponentiations fautées. Dès qu'un tel choix est adopté, l'adversaire est alors capable de construire un dictionnaire.

Définition 2 (Dictionnaire)**.** *En fonction d'un modèle de faute qu'il peut avoir préalablement expérimenté et validé, l'attaquant est capable d'établir* a priori *une liste de valeurs possibles pour les modules fautés n'_i. Une telle liste est appelée un* dictionnaire *(de modules).*

La méthode utilisée par l'attaquant pour retrouver l'exposant privé d sera différente selon qu'il dispose ou non d'un dictionnaire. Comme expliqué à la Section 5.5, si un attaquant a accès à un dictionnaire, alors la partie essentielle de son attaque est d'apprendre quel module possible du dictionnaire a été utilisé pour une faute donnée.

Néanmoins, un dictionnaire n'est pas strictement nécessaire, et une première méthode générale n'utilisant pas de dictionnaire est présentée dans la prochaine section.

5.4 Retrouver l'exposant privé sans dictionnaire

Nous décrivons ici une méthode pour retrouver l'exposant privé d lorsque l'attaquant n'a aucune idée de la valeur que peut prendre un module fauté. Cela correspond à un attaquant qui est incapable de prédire ou d'identifier un quelconque modèle de faute à partir des conditions expérimentales de l'attaque. Notons que dans le cas où l'attaquant a effectivement identifié un modèle de faute et que le dictionnaire qui lui correspond est trop grand pour être aisément manipulé (typiquement 2^{32} entrées ou plus), alors l'attaquant peut choisir d'ignorer ce dictionnaire *inutile* et décider de se placer dans le cas sans dictionnaire.

Par souci de clarté, dans la description des différentes attaques, nous notons p les (possibles) diviseurs de n'_i, et q les (possibles) diviseurs des ordres des sous-groupes considérés. Bien entendu, ces entiers ne doivent pas être confondus avec la factorisation inconnue du module cible n.

5.4.1 Description générale de l'attaque

Dès qu'une campagne de fautes a été menée, l'attaquant connaît des couples de fautes $(\mu_i, s_i)_{1 \leqslant i \leqslant K}$, correspondant à des modules inconnus $n'_i \neq n$, et dans lesquels la base μ_i et la signature fautée s_i sont liées par la relation :

$$s_i = \mu_i^d \bmod n'_i \quad .$$

L'entrée μ_i et la sortie s_i de la fonction de signature sont connues de l'attaquant alors que n'_i est inconnu et modélisé aléatoire et uniformément distribué sur les entiers inférieurs à 2^{ℓ_n}, où ℓ_n est la taille en bits du module.

À partir des données de la campagne de faute, l'exposant privé d est retrouvé "off-line" en déterminant progressivement d modulo r_k pour des petites puissances de premiers r_k. Lorsque le produit $R = \prod_k r_k$ dépasse la valeur du module n (et donc $\varphi(n)$ inconnu), d peut être retrouvé en utilisant le Théorème Chinois des Restes.

Améliorer la fraction des bits de d à connaître

Avant de décrire comment nous allons retrouver d mod r_k, expliquons d'abord pourquoi nous pouvons réduire, dans certains cas, la valeur R qui détermine le nombre de bits d'information modulaire sur d qu'il est nécessaire de connaître.

Si e est petit (typiquement $e = 3$ ou $e = 2^{16} + 1$), alors l'équation reliant les exposants RSA publics et privés

$$ed = 1 + k\varphi(n) = 1 + k(n + 1 - \alpha)$$

peut être utilisée afin de réduire la fraction de bits de d que l'attaquant doit trouver pour déterminer d. Ici, α est une valeur inconnue, et k vérifie $0 < k < e$. Si k est connu, ou supposé par recherche exhaustive quand e est petit, nous avons

$$d = \frac{1 + k(n + 1)}{e} - \frac{k\alpha}{e} \quad ,$$

où la partie inconnue $k\alpha/e$ vérifie (en supposant une factorisation de n équilibrée) :

$$\frac{k\alpha}{e} < \alpha < 2^{\left\lceil \frac{\ell_n}{2}+1 \right\rceil} \quad .$$

La connaissance de d mod R implique la connaissance de la partie inconnue $k\alpha/e$ mod R, si bien que d peut être retrouvé dès lors que R a $\left\lceil \frac{\ell_n}{2} + 1 \right\rceil$ bits de long. En définitive, pour chaque attaque, les deux cas, e petit ou e relativement grand, sont considérés. Il est ainsi possible d'évaluer l'impact sur le nombre de fautes requises.

Remarque 1. Utilisant un résultat de Don Coppersmith permettant de trouver de petites racines de polynômes bivariés [Cop97], Dan Boneh, Glenn Durfee et Yair Frankel [BDF98a, BDF98b] ont montré que, dans le cas d'un petit exposant public, la connaissance des $\frac{\ell_n}{4}$ bits les moins significatifs de d est suffisante pour retrouver d entièrement. Néanmoins, le temps unitaire pour retrouver d par cette méthode ne permettant pas d'envisager de tester facilement un nombre important de candidats pour la valeur de d mod $2^{\ell_n/4}$, nous considérerons qu'il est nécessaire de retrouver la moitié des bits les moins significatifs de d dans le cas d'un petit exposant public.

5.4.2 Une proposition utile

Avant de détailler la partie "off-line", nous donnons la proposition suivante utilisée par la suite.

Proposition 2. *Soit (μ_i, s_i) un couple fauté correspondant au module n'_i, et p^a une puissance de premier telle que $p \nmid \mu_i$ et $p \nmid s_i$. Soit également δ l'ordre multiplicatif de μ_i modulo p^a. Alors, pour tout r divisant δ nous avons :*

$$d \equiv DL(\mu_i, s_i, p^a) \pmod{r} \tag{5.1}$$

avec probabilité 1 si $p^a \mid n'_i$, et avec une probabilité proche de $\frac{1}{r}$ sinon.

Démonstration. Par définition, $s_i = \mu_i^d \bmod n'_i$. Donc, quand $p^a \mid n'_i$, nous avons :

$$\begin{aligned} s_i &\equiv \mu_i^{d \bmod \varphi(p^a)} \pmod{p^a} \\ &\equiv \mu_i^{d \bmod \delta} \pmod{p^a} \end{aligned}$$

si bien que $d \equiv DL(\mu_i, s_i, p^a) \pmod{\delta}$, dont se déduit l'Equation (5.1).

Au contraire, quand $p^a \nmid n'_i$, nous admettons qu'une distribution uniforme de n'_i sur les entiers implique une distribution quasi uniforme de $DL(\mu_i, s_i, p^a)$ sur les classes de résidus modulo r, d'où la proposition. □

Bien entendu, sans connaître n'_i, il est impossible de décider quels p^a divisent n'_i et peuvent donc être utilisés pour déterminer $d \bmod r$, avec certitude, pour certains diviseurs r de $\varphi(p^a)$. Néanmoins, la Proposition 2 suggère que, même si n'_i est inconnu (et donc sa factorisation), on peut monter une attaque basée sur un biais en faveur de la vraie valeur d_r de la classe de résidus de d modulo r.

5.4.3 La phase "off-line"

L'idée de base est que déterminer d_r pour un entier r, peut être réalisé en considérant des p^a pour lesquels $r \mid \varphi(p^a)$, et en considérant le logarithme discret de s_i en base μ_i modulo p^a. D'après la Proposition 2, et pourvu que r divise également l'ordre multiplicatif de μ_i modulo p^a, la distribution de probabilité de $DL(\mu_i, s_i, p^a, r)$ est :

$$\mathcal{P}\left(DL(\mu_i, s_i, p^a, r) = x\right) = \begin{cases} \frac{1}{p^a} + \frac{p^a-1}{r \cdot p^a} & \text{si } x = d_r \\ \frac{p^a-1}{r \cdot p^a} & \text{si } x \neq d_r \end{cases} .$$

En calculant la valeur $DL(\mu_i, s_i, p^a, r)$ pour tous les couples fautés de la campagne de fautes, et en comptant combien de fois chaque classe de résidus est suggérée, nous nous attendons à ce que la valeur correcte d_r émerge du bruit, et soit suggérée plus souvent que les autres.

Notons que la valeur du biais

$$\varepsilon = \frac{\frac{1}{p^a} + \frac{p^a-1}{r \cdot p^a}}{\frac{p^a-1}{r \cdot p^a}} - 1 = \frac{r}{p^a - 1}$$

diminue en raison de $p^a - 1$. Cela signifie qu'étant donné r, plus p^a est petit, plus le biais sera grand et donc plus petit sera le nombre de fautes nécessaires pour déterminer d_r. Cela suggère l'algorithme de la Figure 5.1 qui, étant donné r, essaie de trouver la classe de résidus d_r. Parmi toutes les valeurs possibles de p^a telles que $r \mid \varphi(p^a)$, cet algorithme ne considère que le plus petit premier p pour lequel $r \mid (p-1)$ puisque ce choix donne le biais le plus grand avec une forte probabilité.

L'algorithme de la Figure 5.1 apporte la connaissance de d_r, pour des puissances de premier $r = q^f$ individuelles. L'attaquant peut intégrer cette brique dans une procédure de plus haut niveau qui déterminera d_r pour autant de valeurs r que nécessaire, afin que $R = \prod_k r_k$ soit suffisamment grand pour retrouver d.

5.4.4 Résultats

Cette méthode de comptage a été implémentée en simulations. Il en résulte que 512 bits d'information de classe de résidus sur d sont facilement retrouvés après 25 000 fautes, ce qui

Algorithme 5.1 Détermination de $d \bmod r$ par exploitation d'un biais

Entrée : $r = q^f$, une petite puissance d'un petit premier

Sortie : une valeur supposée de $d_r = d \bmod r$

1. initialiser à zéro un tableau $\texttt{count}[0, \ldots, r-1]$

   /* *recherche du plus petit premier* p *tel que* r *divise* $p-1$ */
2. $p \leftarrow r + 1$
3. **tant que** p n'est pas premier
4. $\quad p \leftarrow p + r$
5. **fin tant que**

   /* *exploitation du biais* */
6. **pour tout** couple fauté (μ_i, s_i)
7. $\quad$ **si** $(p \nmid \mu_i)$ **et** $(p \nmid s_i)$ **alors**
8. $\quad\quad$ calculer δ l'ordre multiplicatif de μ_i modulo p
9. $\quad\quad$ **si** $(r \mid \delta)$ **et** $(DL(\mu_i, s_i, p, r)$ existe$)$ **alors**
10. $\quad\quad\quad \texttt{count}[DL(\mu_i, s_i, p, r)]{+}{+}$
11. $\quad\quad$ **fin si**
12. $\quad$ **fin si**
13. **fin pour**
14. **retourne** d_r tel que $\texttt{count}[d_r] = \max_i(\texttt{count}[i])$

FIG. 5.1 – Détermination de $d \bmod r$ par exploitation d'un biais.

est suffisant pour une clé de 1024 bits avec un petit exposant public. Environ 60 000 fautes permettent de retrouver 1024 bits d'information, ce qui est suffisant aussi bien pour une clé quelconque de 1024 bits, que pour une clé de 2048 bits avec un petit exposant public.

5.5 Retrouver l'exposant privé avec un dictionnaire

Comme mentionné précédemment, aucun dictionnaire n'est nécessaire pour appliquer la méthode de la Section 5.4. Néanmoins, quand l'attaquant dispose d'un dictionnaire S, il lui est alors possible d'être plus efficace que par la méthode de comptage.

5.5.1 Méthodologie générale

L'observation centrale est que, avec un dictionnaire S, il devient possible de relier un module particulier $\nu_j \in S$ à un couple fauté (μ_i, s_i). Nous introduisons donc la définition suivante :

Définition 3 (Touche)**.** *Pour tout* $\nu_j \in S$*, nous disons qu'un attaquant a identifié une* touche *pour* ν_j *s'il a été capable d'identifier un couple fauté* (μ_i, s_i) *pour lequel* $n'_i = \nu_j$*.*

À partir d'une touche une certaine quantité d'information sur d peut être collectée. En effet, d'après l'Équation (5.1), il est possible d'extraire l'information liée à chaque p^a connu divisant ν_j. On peut ainsi retrouver $d \bmod q^f$ pour tout q^f qui divise l'ordre multiplicatif de μ_i modulo p^a.

TAB. 5.1 – Quantité d'information (en bits) dérivée de l'exploitation des touches

Limite DL	Nombre de touches									
	1	2	3	4	5	6	7	8	9	10
10^5	33	62	87	111	136	159	182	206	227	248
10^7	41	75	113	150	184	219	251	285	315	346
10^9	47	93	135	177	214	255	296	334	374	412
	11	12	13	14	15	16	17	18	19	20
10^5	267	289	312	331	352	373	396	416	436	455
10^7	374	406	436	465	493	**522**	553	580	609	639
10^9	452	490	**526**	561	599	638	673	709	744	780
	21	22	23	24	25	26	27	28	29	30
10^5	477	496	**516**	537	533	570	587	602	618	635
10^7	667	695	723	751	778	807	833	861	890	916
10^9	815	849	881	917	953	988	1018	**1055**	1088	1124

Nous insistons sur le fait qu'il est possible de considérer et d'exploiter tout facteur connu de ν_j même si ce module n'a pas été entièrement factorisé. L'attaque consiste donc à identifier des touches pour quelques modules ν_j et à rassembler l'information relative aux facteurs connus de chacun de ces modules. Cela nous amène à la question : combien de touches fournissent suffisamment d'information pour retrouver l'exposant privé ?

La Table 5.1 montre des résultats de simulation où le nombre de bits d'information retrouvés sur d est donné en fonction du nombre de touches exploitées. Les modules de ces touches ont été factorisés par la *méthode des courbes elliptiques* (en anglais, *Elliptic Curve Method*, ECM) jusqu'à des facteurs de 20 à 25 chiffres, et l'information a été extraite par rapport à tous les q inférieurs à une limite donnée à laquelle nous avons donné les valeurs successives 10^5, 10^7 et 10^9. Ces simulations ont été répétées plusieurs fois, et nous présentons leurs moyennes sur 200 expériences.

Quand la limite de calcul sur le logarithme discret est égale à 10^9, 28 touches sont suffisantes pour retrouver une clé RSA de 1024 bits (13 dans le cas d'un petit exposant public), et 59 touches permettent de retrouver l'exposant privé d'une clé RSA de 2048 bits (28 dans le cas d'un petit exposant public).

Connaissant le nombre de touches nécessaires, nous présentons maintenant deux méthodes visant à les identifier. La première méthode, décrite à la Section 5.5.2, les détecte à l'occasion de collisions sur des logarithmes discrets. La deuxième méthode est présentée à la Section 5.5.3. Nous y formalisons précisément ce qui se passe dans cette attaque et nous donnons une expression de la probabilité *a posteriori* d'un module ou d'un tuple de modules. Ces probabilités permettent alors une exploitation efficace des fautes.

5.5.2 Identification de touches par la méthode des collisions

Soit r un entier divisant l'ordre multiplicatif de μ_i modulo p^a. La Proposition 2 implique que calculer $DL(\mu_i, s_i, p^a, r)$ pour différents couples (μ_i, s_i) fournit toujours la valeur correcte de d_r dès lors que p^a divise n'_i. Dans le cas contraire, c'est-à-dire si $p^a \nmid n'_i$, les résultats sont distribués uniformément entre 0 et $(r-1)$.

Cela suggère une méthode qui détecte des collisions du type :

$$DL(\mu_{i_1}, s_{i_1}, p^a, r) = DL(\mu_{i_2}, s_{i_2}, p^a, r) \quad .$$

Pour un choix adéquat des valeurs p^a et r, et avec une forte probabilité (voir Remarque 2 ci-dessous), une telle collision, non seulement révèle que p^a divise à la fois n'_{i_1} et n'_{i_2}, mais permet également de déterminer la valeur $\nu \in S$ telle que $n'_{i_1} = n'_{i_2} = \nu$. Cela identifie donc une touche pour ce module commun.

Définition 4 (Marqueur)**.** *Etant donné un module $\nu \in S$, un couple (p, q) est appelé un* marqueur *pour ν, si p est un facteur premier connu de ν, et q est un premier* pas trop petit[18] *divisant $(p-1)$.*

Phase de préparation

Pour autant de modules $\nu \in S$ que possible, nous essayons de leur trouver à chacun un marqueur spécifique. Nous notons $S^* \subseteq S$ l'ensemble des modules pour lesquels un marqueur a été identifié.

Phase de recherche de collision

Pour chaque $\nu_j \in S^*$ ayant (p_j, q_j) comme marqueur, nous maintenons une liste $\mathcal{D}_{\nu_j}$ de tous les $DL(\mu_i, s_i, p_j, q_j)$ pour tous les couples fautés exploités jusque là. Dès lors que deux couples fautés ont la même valeur de module $\nu_j = n'_{i_1} = n'_{i_2}$, une collision apparaît nécessairement dans $\mathcal{D}_{\nu_j}$. En négligeant les possibles faux positifs, nous pouvons ainsi identifier une touche pour ν_j.

Complexité

Dans le cas idéal où un marqueur a été trouvé pour tous les modules de S (c'est-à-dire, $S^* = S$), le nombre de fautes nécessaires pour obtenir une telle collision est $\mathcal{O}(\sqrt{|S|})$. Pour de petites valeurs de t, obtenir t touches nécessite $\mathcal{O}(\sqrt{t|S|})$ fautes.

Dans le cas plus probable où seulement une fraction $\beta = |S^*|/|S|$ de tous les modules possibles ont obtenu un marqueur, le nombre de fautes nécessaires pour obtenir t touches est $\mathcal{O}(\sqrt{\frac{t}{\beta}|S|})$.

Remarque 2 (Faux positifs). Pour un ν_j donné, une vraie collision apparaît dans $\mathcal{D}_{\nu_j}$ après $2|S|$ fautes en moyenne, alors qu'une fausse collision apparaît après $\mathcal{O}(\sqrt{q_j})$ fautes. En conséquence, le problème de l'occurrence de faux positifs peut être négligé dès lors que $\min_j(\sqrt{q_j}) \gg 2|S|$. Cette inégalité explique la notion de *pas trop petit* introduite dans la Définition 4.

Application

Concrètement, supposons un attaquant ciblant le transfert du module de l'EEPROM vers la RAM[19], capable de modifier aléatoirement n'importe quel octet individuel du module, mais incapable de contrôler quel octet en particulier il modifie. Ce modèle de faute est très réaliste lorsque, comme contre-mesure, les octets du module sont transférés en ordre aléatoire. Le dictionnaire correspondant contient alors $2^8 \times \frac{1024}{8} = 2^{15}$ (resp. 2^{16}) modules pour un clé RSA de 1024 bits (resp. 2048 bits). Supposons également qu'un marqueur a été trouvé pour 80% des

[18] Le fait que q doive être *pas trop petit* est nécessaire pour éviter les faux positifs dans la recherche de collisions (voir Remarque 2).

[19] Opération de chargement de clé dans un crypto-processeur préalable à tout calcul de signature.

modules. D'après la Table 5.1, retrouver une clé dans le cas général nécessite 28 touches donc environ 1100 fautes (resp. environ 2200 fautes pour 2048 bits). Lorsqu'un petit exposant public est utilisé, seulement 750 fautes environ sont nécessaires pour 1024 bits (resp. environ 1500 fautes pour 2048 bits). Cela montre que même lorsqu'elle est appliquée à un dictionnaire aussi grand, cette méthode en *racine carrée* permet de considérablement réduire le nombre de fautes nécessaires par rapport au cas où aucun modèle de faute n'a été identifié.

5.5.3 Exploitation bayésienne des fautes

L'objectif de cette méthode est de calculer la probabilité *a posteriori* d'un t-uple $\vec{\nu} = (\nu_{j_1}, \ldots, \nu_{j_t})$ de candidats pour les modules fautés $(n'_1, \ldots, n'_t)$, conditionnée aux observations de t fautes $(\mu_h, s_h)_{h=1,\ldots,t}$.

Observation d'une seule faute

Considérons tout d'abord le cas $t = 1$. Nous modélisons l'exposant privé d comme ayant été tiré aléatoirement et uniformément dans $[0, \varphi(n)[$, et nous voulons établir la probabilité *a posteriori* d'un module $\nu \in S$, conditionnée à l'observation d'une faute (μ, s). La base μ étant donnée, nous appliquons le théorème de Bayes :

$$\begin{aligned} \mathcal{P}(\nu \mid s) &= \frac{\mathcal{P}(\nu \text{ et } s)}{\mathcal{P}(s)} \\ &= \frac{\mathcal{P}(\nu \text{ et } s)}{\sum_{\nu_j \in S} \mathcal{P}(\nu_j \text{ et } s)} \\ &= \frac{\mathcal{P}(\nu)\,\mathcal{P}(s \mid \nu)}{\sum_{\nu_j \in S} \mathcal{P}(\nu_j)\,\mathcal{P}(s \mid \nu_j)} \quad . \end{aligned}$$

Considérant que les modules du dictionnaire S sont équitablement distribués, nous obtenons :

$$\mathcal{P}(\nu \mid s) = \frac{\mathcal{P}(s \mid \nu)}{\sum_{\nu_j \in S} \mathcal{P}(s \mid \nu_j)} \quad .$$

Notons que dans le but de comparer ces probabilités entre elles, le terme $\sum_{\nu_j \in S} \mathcal{P}(s \mid \nu_j)$ ne joue le rôle que d'un facteur de normalisation qui n'aura aucune influence sur les probabilités relatives des modules candidats, les unes par rapport aux autres. Nous nous intéressons donc au calcul de la probabilité $\mathcal{P}(s \mid \nu)$ qui est égale à la proportion de valeurs de d permettant d'observer s dans le cas où la valeur du module fauté est ν :

$$\mathcal{P}(s \mid \nu) = \frac{\delta(s, \nu)}{\varphi(n)} \quad ,$$

avec

$$\delta(s, \nu) = \#\{0 \leqslant d < \varphi(n) \text{ tels que } s = \mu^d \bmod \nu\} \quad .$$

Notant $\prod_i p_i^{a_i}$ la factorisation de ν, l'évaluation de $\delta(s, \nu)$ se fait en interprétant l'équation d'observation vis-à-vis de chacun des p^a qui divisent exactement ν :

$$s = \mu^d \bmod \nu \quad \Longleftrightarrow \quad \left(\bigwedge_i \mathcal{E}_i \right) \wedge (s < \nu) \quad ,$$

où, pour chaque $p_i^{a_i} \parallel \nu$, nous notons $\mathcal{E}_i$ l'équation

$$\mathcal{E}_i : \quad s \equiv \mu^d \pmod{p_i^{a_i}}$$

qui peut valoir :

$$\mathcal{E}_i \;=\; \begin{cases} \top \text{ (vrai)} & \text{si } (p_i \mid \mu \text{ et } p_i^{a_i} \mid s) \\ \bot \text{ (faux)} & \text{si } (p_i \mid \mu \text{ et } p_i^{a_i} \nmid s)^{20} \\ \bot \text{ (faux)} & \text{si } (p_i \nmid \mu \text{ et } p_i \mid s) \\ s \in \langle \mu \rangle_{p_i^{a_i}} & \text{si } (p_i \nmid \mu \text{ et } p_i \nmid s) \end{cases},$$

où $\langle \mu \rangle_{p_i^{a_i}} \subset (\mathbb{Z}/p_i^{a_i}\mathbb{Z})^\star$ désigne le sous-groupe d'ordre $\sigma_{p_i^{a_i}}(\mu)$ des puissances de μ modulo $p_i^{a_i}$.

Dans le dernier cas, et notant $\prod_j q_j^{f_j}$ la factorisation de $\varphi(p_i^{a_i})$, l'équation

$$\mathcal{E}_i : \quad s \in \langle \mu \rangle_{p_i^{a_i}}$$

peut, à son tour, être décomposée en :

$$\mathcal{E}_i \quad \Longleftrightarrow \quad \bigwedge_j \mathcal{E}_{i,j} \quad ,$$

où, pour chaque j, l'équation $\mathcal{E}_{i,j}$ exprime la contrainte que l'on peut établir sur d en considérant la composante en q_j de $\varphi(p_i^{a_i})$. Omettant les indices i et j pour alléger l'écriture, et notant $v(\mu)$ la valuation[21] de q dans l'ordre multiplicatif $\sigma_{p^a}(\mu)$ de μ modulo p^a (ainsi que $v(s)$ celle de q dans l'ordre multiplicatif $\sigma_{p^a}(s)$ de s modulo p^a), nous avons :

$$\mathcal{E}_{i,j} \;=\; \begin{cases} \top \text{ (vrai)} & \text{si } v(\mu) = v(s) = 0 \\ \bot \text{ (faux)} & \text{si } v(\mu) < v(s) \\ d \;\equiv\; \lambda \pmod{q^{v(\mu)}} & \text{sinon, avec } s^{\varphi(p^a)/q^{v(\mu)}} \equiv \left(\mu^{\varphi(p^a)/q^{v(\mu)}}\right)^\lambda \pmod{p^a} \end{cases}.$$

Dans ce dernier cas, l'équation $d \equiv \lambda \pmod{q^{v(\mu)}}$, que nous pouvons noter de manière plus concise par le triplet $(q, v(\mu), \lambda)$, exprime une contrainte modulaire que doit nécessairement vérifier l'exposant privé d.

Pourvu que $s < \nu$, la condition initiale $s = \mu^d \bmod \nu$ est donc équivalente à la conjonction d'un ensemble de conditions de type $\mathcal{E}_i$ ou $\mathcal{E}_{i,j}$. Cette conjonction, dans laquelle tous les $\top$ sont ignorés, vaut bien évidemment $\bot$ si cette valeur apparaît au moins une fois. Dans le cas contraire, il ne reste que des contraintes modulaires de type (q, v, λ). Lorsque deux contraintes modulaires (q, v', λ') et (q, v'', λ'') impliquent le même premier q, nous devons vérifier qu'elles ne sont pas incompatibles. Supposant sans perte de généralité que $v' \geqslant v''$, nous les remplaçons ainsi par : ou bien (q, v', λ') si $\lambda' \bmod q^{v''} = \lambda''$, ou bien $\bot$ dans le cas où cette cohérence n'est pas vérifiée. Itérant ce processus de combinaison jusqu'à ce que toutes les contraintes modulaires impliquent des premiers distincts, nous obtenons finalement comme évaluation de la condition initiale :

$$\begin{aligned} s = \mu^d \bmod \nu \quad &\Longleftrightarrow \quad \left(\bigwedge_i \mathcal{E}_i\right) \wedge \left(\bigwedge_{i,j} \mathcal{E}_{i,j}\right) \wedge (s < \nu) \\ &\Longleftrightarrow \quad \begin{cases} \bot \text{ (faux)} & \text{si une impossibilité existe,} \\ d \;\equiv\; \Lambda \pmod{\Omega} & \text{sinon.} \end{cases} \end{aligned}$$

[20] Pour tout entier positif $b < a_i$, le cas $(p_i \mid \mu \text{ et } p_i^{a_i} \nmid s \text{ et } p_i^b \mid s)$ est déclaré impossible car s est considéré être une *grande* puissance de μ. Rigoureusement, cela peut être inexact dans le cas d'une exceptionnellement petite valeur de d ($d \leqslant b < a_i$) qui, par ailleurs, serait nécessairement non sûre.

[21] Par définition, la valuation de q dans un entier σ est l'exposant de la plus grande puissance de q qui divise σ.

L'équation $d \equiv \Lambda \pmod{\Omega}$ résulte du processus de recombinaison par le *théorème chinois des restes* (CRT) de l'ensemble des contraintes modulaires individuelles, la valeur de Ω étant le produit de tous les q^v apparaissant dans celles-ci.

Revenant à notre objectif initial de calculer la probabilité $\mathcal{P}(\, s \mid \nu \,) = \delta(s,\nu)/\varphi(n)$ de la signature observée, conditionnellement à la valeur candidate ν pour le module fauté n', nous obtenons que la valeur de $\delta(s,\nu)$ est, soit nulle dans toute une série de cas que nous avons bien définis, soit égale à :

$$\begin{aligned}
\delta(s,\nu) &= \mathrm{Card}\big(\{0 \leqslant d < \varphi(n) \text{ tels que } s = \mu^d \bmod \nu\}\big) \\
&= \mathrm{Card}\big(\{0 \leqslant d < \varphi(n) \text{ tels que } d \equiv \Lambda \pmod{\Omega}\}\big) \\
&\in \left\{ \left\lfloor \frac{\varphi(n)}{\Omega} \right\rfloor, \left\lceil \frac{\varphi(n)}{\Omega} \right\rceil \right\} \quad .
\end{aligned}$$

Résumant heuristiquement $\delta(s,\nu)$ par la valeur $\frac{\varphi(n)}{\Omega}$, nous obtenons :

$$\mathcal{P}(\, s \mid \nu \,) = \frac{\delta(s,\nu)}{\varphi(n)} = \frac{1}{\Omega} \quad .$$

Remarque 3. Comme produit de tous les q^v apparaissant dans les contraintes modulaires après les fusions de celles partageant un même premier, la valeur de Ω est égale au plus petit commun multiple de tous les q^v divisant exactement l'un des $\sigma_{p^a}(\mu)$. Exprimant ν comme $\nu = \hat{\nu} \cdot \check{\nu}$, avec

$$\hat{\nu} = \prod_{\substack{i \\ p_i \nmid \mu}} p_i^{a_i} \quad \text{et} \quad \check{\nu} = \prod_{\substack{i \\ p_i \mid \mu}} p_i^{a_i} \quad ,$$

il apparaît que Ω n'est rien d'autre que $\sigma_{\hat{\nu}}(\mu)$, l'ordre multiplicatif de μ dans $(\mathbb{Z}/\hat{\nu}\mathbb{Z})^\star$, et que Λ est le logarithme discret de s en base μ dans ce sous-groupe.

Observation de plusieurs fautes

Lorsque le nombre t de fautes observées est supérieur à 1, nous pouvons définir de la même façon la probabilité *a posteriori* d'un t-uple $\vec{\nu} = (\nu_{j_1}, \ldots, \nu_{j_t})$ de candidats modules fautés, conditionnée aux t fautes observées $(\mu_h, s_h)_{h=1,\ldots,t}$. Le t-uple $\vec{\mu} = (\mu_1, \ldots, \mu_t)$ étant donné, et désignant par $\vec{s} = (s_1, \ldots, s_t)$ le t-uple des signatures observées, nous avons :

$$\begin{aligned}
\mathcal{P}(\, \vec{\nu} \mid \vec{s} \,) &= \frac{\mathcal{P}(\, \vec{\nu} \text{ et } \vec{s} \,)}{\mathcal{P}(\, \vec{s} \,)} \\
&= \frac{\mathcal{P}(\, \vec{\nu} \text{ et } \vec{s} \,)}{\sum\limits_{\vec{\nu}_{\vec{j}} \in S^t} \mathcal{P}\left(\vec{\nu}_{\vec{j}} \text{ et } \vec{s} \right)} \\
&= \frac{\mathcal{P}(\, \vec{s} \mid \vec{\nu} \,)}{\sum\limits_{\vec{\nu}_{\vec{j}} \in S^t} \mathcal{P}\left(\vec{s} \mid \vec{\nu}_{\vec{j}} \right)} \quad .
\end{aligned}$$

Faisant fi du facteur de normalisation, nous nous intéressons donc aux probabilités $\mathcal{P}(\, \vec{s} \mid \vec{\nu} \,)$ qui reflètent dans leur ensemble les probabilités relatives *a posteriori* de chaque $\vec{\nu}$. Nous avons

$$\mathcal{P}(\, \vec{s} \mid \vec{\nu} \,) = \frac{\delta(\vec{s},\vec{\nu})}{\varphi(n)} \quad ,$$

avec

$$\delta(\vec{s},\vec{\nu}) \;=\; \#\{0 \leqslant d < \varphi(n) \text{ tels que } \forall h = 1,\ldots,t,\; s_h = \mu_h^d \bmod \nu_{j_h}\}\;.$$

Nous généralisons le raisonnement précédent en exprimant que chaque équation d'observation individuelle $s_h = \mu_h^d \bmod \nu_{j_h}$ doit être satisfaite :

$$s_h = \mu_h^d \bmod \nu_{j_h} \quad\Longleftrightarrow\quad \Big(\bigwedge_i \mathcal{E}_{h,i}\Big) \;\wedge\; (s_h < \nu_{j_h}) \quad ,$$

où, pour chaque $p_i^{a_i} \parallel \nu_{j_h}$, nous notons $\mathcal{E}_{h,i}$ l'équation

$$\mathcal{E}_{h,i} : \quad s_h \equiv \mu_h^d \pmod{p_i^{a_i}}$$

qui peut valoir :

$$\mathcal{E}_{h,i} \;=\; \begin{cases} \top \text{ (vrai)} & \text{si } (p_i \mid \mu_h \text{ et } p_i^{a_i} \mid s_h) \\ \bot \text{ (faux)} & \text{si } (p_i \mid \mu_h \text{ et } p_i^{a_i} \nmid s_h)^{21} \\ \bot \text{ (faux)} & \text{si } (p_i \nmid \mu_h \text{ et } p_i \mid s_h) \\ s_h \in \langle \mu_h \rangle_{p_i^{a_i}} & \text{si } (p_i \nmid \mu_h \text{ et } p_i \nmid s_h) \end{cases} .$$

Dans le dernier cas, chaque équation $\mathcal{E}_{h,i}$ peut, à son tour, être décomposée selon le même principe que précédemment en :

$$\mathcal{E}_{h,i} \quad\Longleftrightarrow\quad \bigwedge_j \mathcal{E}_{h,i,j} \quad .$$

Pour chaque équation d'observation individuelle $s_h = \mu_h^d \bmod \nu_{j_h}$, et après le processus de fusion des $\mathcal{E}_{h,i,j}$ relatifs à un même premier q, nous obtenons la formulation équivalente :

$$\begin{aligned} s_h = \mu_h^d \bmod \nu_{j_h} \quad &\Longleftrightarrow\quad \Big(\bigwedge_i \mathcal{E}_{h,i}\Big) \;\wedge\; \Big(\bigwedge_{i,j} \mathcal{E}_{h,i,j}\Big) \;\wedge\; (s_h < \nu_{j_h}) \\ &\Longleftrightarrow\quad \begin{cases} \bot \text{ (faux)} & \text{si une impossibilité existe,} \\ d \;\equiv\; \Lambda_h \pmod{\Omega_h} & \text{sinon.} \end{cases} \end{aligned}$$

Notons que jusqu'ici nous n'avons fait qu'exprimer t fois indépendamment le raisonnement décrit précédemment, lequel conduit à l'évaluation du nombre $\delta(s_h, \nu_{j_h})$ de valeurs de d vérifiant l'équation d'observation pour chacun des $h = 1,\ldots,t$. Ce nombre $\delta(s_h, \nu_{j_h})$ est, ou bien égal à 0 si cette observation est intrinsèquement impossible pour cette valeur de ν_{j_h}, ou bien égal au nombre de solutions de l'équation $d \;\equiv\; \Lambda_h \pmod{\Omega_h}$.

Cependant, l'évaluation de $\delta(\vec{s},\vec{\nu})$, qui est nécessaire à celle de la probabilité $\mathcal{P}\,(\,\vec{s} \mid \vec{\nu}\,)$ qui nous intéresse, requiert une interprétation conjointe de l'ensemble de toutes les contraintes obtenues.

Dans ce but, et si aucune observation individuelle ne s'est révélée intrinsèquement impossible, nous achevons la fusion des contraintes modulaires $\mathcal{E}_{h,i,j}$ de type (q, v, λ) par une fusion croisée (c'est-à-dire, pour des valeurs de h distinctes) de toutes les paires $((q, v', \lambda'), (q, v'', \lambda''))$ relatives à un même premier. Lorsqu'il ne révèle aucune incompatibilité des observations entre elles, le résultat de ce processus de fusion croisée est une contrainte modulaire sur d résumant l'intégralité de l'information apportée par l'observation de $\vec{s}$:

$$d \equiv \Lambda \pmod{\Omega} \quad ,$$

où Λ résulte de la recombinaison CRT de toutes les contraintes modulaires $\mathcal{E}_{h,i,j}$, et où Ω, comme produit de toutes les puissances de premiers apparaissant dans celles-ci, est le plus petit commun multiple des différents Ω_h.

Remarque 4. De manière analogue à la Remarque 3, et exprimant chaque ν_{j_h} comme $\nu_{j_h} = \hat{\nu}_{j_h} \cdot \check{\nu}_{j_h}$, avec

$$\hat{\nu}_{j_h} = \prod_{\substack{i \\ p_i \nmid \mu_h}} p_i^{a_i} \quad \text{et} \quad \check{\nu}_{j_h} = \prod_{\substack{i \\ p_i \mid \mu_h}} p_i^{a_i} \quad ,$$

nous observons que Ω est l'ordre multiplicatif $\sigma_{\vec{\nu}}(\vec{\mu})$ de l'élément $(\mu_1, \ldots, \mu_t)$ dans le sous-groupe produit direct $(\mathbb{Z}/\hat{\nu}_{j_1}\mathbb{Z})^\star \times \ldots \times (\mathbb{Z}/\hat{\nu}_{j_t}\mathbb{Z})^\star$, et que Λ est le logarithme discret de $\vec{s}$ en base $\vec{\mu}$ dans ce sous-groupe.

Le paragraphe suivant discute, entre autre, de la possibilité de déterminer, avec une puissance de calcul limitée, les valeurs exactes de Λ et Ω. Lorsque Ω peut être calculé ou estimé, nous évaluons comme précédemment la probabilité conditionnelle de chaque $\vec{\nu}$ qui ne s'est pas révélé incompatible avec les observations, comme étant proportionnelle à

$$\mathcal{P}\,(\,\vec{s} \mid \vec{\nu}\,) \;=\; \frac{\delta(\vec{s}, \vec{\nu})}{\varphi(n)} \;=\; \frac{1}{\Omega} \quad .$$

Lorsqu'il est possible de déterminer exactement Λ et Ω, et sous l'hypothèse que les logarithmes discrets de $\vec{s}$ sont uniformément répartis modulo Ω lorsque la supposition $\vec{\nu}$ est incorrecte, il est possible de rejeter chaque supposition $\vec{\nu}$ pour laquelle $\Lambda \geqslant n$. Cette observation n'est cependant d'une utilité que théorique dans la mesure où la détermination exacte de Λ nécessite presque sûrement le calcul d'un logarithme discret dans un grand sous-groupe.

Considérations calculatoires

Ayant décrit comment calculer théoriquement les probabilités non normalisées de chaque candidat $\vec{\nu}$, nous considérons maintenant certaines difficultés pratiques que peut rencontrer un attaquant aux capacités calculatoires limitées.

Étant donné $\nu \in S$, le premier type de problème qui peut se présenter est de ne pas pouvoir complètement factoriser ce module fauté candidat. Lorsque la factorisation de ν est incomplète, il n'est possible de considérer les équations $\mathcal{E}_i : s \equiv \mu^d \pmod{p_i^{a_i}}$ et les équations $\mathcal{E}_{i,j}$ qui peuvent en découler, que vis-à-vis des "petits" premiers p_i connus.

Pour les valeurs de ν ayant été complètement factorisées, une deuxième difficulté calculatoire apparaît. En effet, avec une très grande probabilité, la factorisation complète de ν contient un grand facteur premier noté $\overline{p}$ qui ne divise ni μ, ni s, et dont la valuation $\overline{a}$ dans ν est égale à 1. Il est alors nécessaire de factoriser $\varphi(\overline{p}) = \overline{p} - 1$. Deux cas peuvent se présenter selon qu'il a été possible ou non d'achever cette factorisation.

Si la factorisation de $\overline{p} - 1$ est complète, alors elle contient presque sûrement un grand premier $\overline{q}$, de valuation $\overline{f}$ égale à 1, et pour lequel $v(\mu)$ et $v(s)$ (qui sont facilement calculables) valent 1 (c'est-à-dire que μ et s sont tous les deux d'ordre $\overline{q}$ relativement à cette composante). La détermination du logarithme discret de s vis-à-vis de cette composante (le λ du triplet $(\overline{q}, 1, \lambda)$) n'est pas possible car $\overline{q}$ est trop grand et l'attaquant n'est donc pas capable de calculer Λ. Notons qu'il lui est toutefois possible de déterminer Ω avec certitude, ce qui lui permet d'évaluer $\mathcal{P}\,(\,s \mid \nu\,)$.

En revanche, s'il n'a pas pu factoriser $\overline{p} - 1$ complètement, l'attaquant s'est heurté à la factorisation d'un grand entier composé $\overline{c}$ dont il sait qu'il est le produit de plusieurs grands premiers $\overline{q}_k$, chacun de valuation $\overline{f}_k$ presque sûrement égale à 1. Bien qu'il ne lui soit pas possible de le vérifier avec certitude (contrairement au cas précédent), l'attaquant peut considérer comme presque sûr que ni μ, ni s ne sont d'ordre 1 vis-à-vis d'aucune des composantes $\overline{q}_k$. Comme dans

le cas précédent, l'attaquant ne peut pas calculer Λ mais peut intégrer la valeur de $\overline{c}$ dans la détermination de Ω. Il lui est donc possible, ici aussi, d'évaluer $\mathcal{P}(s \mid \nu)$.

On remarque que les valeurs de $\overline{q}_k$ et de $\overline{q}$ (relativement auxquelles l'attaquant ne sait pas calculer de logarithme discret) étant très grandes, la probabilité que l'une d'elles apparaisse pour différentes valeurs de ν est très faible. L'ignorance des valeurs de ces logarithmes discrets n'est donc pas un souci pour l'attaquant lors du processus de fusion croisée des contraintes modulaires relativement à plusieurs observations.

Déroulement de l'attaque

S'il sait déterminer les probabilités de chaque ensemble $\vec{\nu}$ de t suppositions, l'attaquant peut mener son attaque de la manière suivante. Il commence par acquérir une première faute (μ_1, s_1) et calcule l'ensemble des $\mathcal{P}(\nu_{j_1} \mid s_1) \sim \mathcal{P}(s_1 \mid \nu_{j_1})$ pour tous les $\nu_{j_1} \in S$. Il maintient une liste épurée de candidats dont il ne conserve que ceux ayant une probabilité non nulle. Il obtient ensuite une deuxième faute (μ_2, s_2), et calcule $\mathcal{P}((\nu_{j_1}, \nu_{j_2}) \mid (s_1, s_2))$ pour chaque ν_{j_1} dans sa liste et pour chaque $\nu_{j_2} \in S$. Cette probabilité est calculée comme expliquée précédemment en n'oubliant pas la phase de fusion croisée. Il ne retient dans sa liste que les probabilités non nulles, et continue ainsi pour autant de fautes acquises qu'il lui est nécessaire.

Si l'attaquant sait déterminer les valeurs des Λ[22], il est alors capable de rejeter, presque sûrement, tout candidat $\vec{\nu}$ pour lequel Ω est significativement plus grand que n. Pour chaque module candidat ν, Ω est l'ordre multiplicatif de μ dans $(\mathbb{Z}/\hat{\nu}\mathbb{Z})^\star$. Cette valeur n'est pas de taille beaucoup plus petite que celle de n et la différence des tailles rend compte, pour une part de la composante perdue $\check{\nu}$ qui n'est très probablement formée que de puissances de petits premiers, pour une autre part du fait que l'ordre maximal d'un élément de $(\mathbb{Z}/\hat{\nu}\mathbb{Z})^\star$ n'est pas $\hat{\nu}$ mais plutôt $\lambda(\hat{\nu})$, et enfin du fait que μ n'est pas nécessairement d'ordre maximal dans $(\mathbb{Z}/\hat{\nu}\mathbb{Z})^\star$. Mises bout à bout, ces considérations impliquent une différence de taille vraisemblablement modérée entre la valeur de n d'une part, et celle du Ω relatif à un module candidat individuel d'autre part. Pour un module n de taille 1024 bits, et considérant que les ν du dictionnaire sont de tailles analogues à celle de n, on peut s'attendre à des valeurs de Ω d'une taille typique de l'ordre de 900 bits. Dès l'exploitation d'une deuxième faute, la valeur de Ω sera de l'ordre de 1800 bits moins une perte de couplage entre les deux fautes. C'est alors très largement suffisant pour pouvoir rejeter les candidats incorrects pour lesquels $\Lambda \geqslant n$, et identifier $d = \Lambda$ pour le(s) seul(s) candidat(s) vérifiant $\Lambda < n$. Nous remarquons donc que, lorsque l'attaquant est capable de calculer les valeurs des Λ, la quantité d'information qu'il peut extraire de chaque faute (c'est-à-dire, la taille en bits de Ω) lui est suffisante pour pouvoir retrouver d avec seulement deux fautes.

Considérons maintenant le cas d'un attaquant incapable de calculer Λ mais qui sait déterminer Ω. C'est notamment le cas lorsque la factorisation de ν est achevée. Une grande partie de l'information contenue dans Λ lui est alors inaccessible, mais il est néanmoins capable de calculer les probabilités de chaque $\vec{\nu}$. Il peut donc établir une liste de $\vec{\nu}$ compatibles avec les observations. Pour chacun d'eux, il connaît une liste de contraintes modulaires sur d, précisément celles qu'il a pu obtenir par calcul de logarithme vis-à-vis de tous les $q^{v(\mu)}$, à l'exception de ceux relatifs aux grands premiers $\overline{q}$ ou $\overline{q}_k$. Lorsque la résultante de ces contraintes modulaires dépasse la taille de n (ou la moitié de la taille de n dans le cas d'un petit exposant public), la valeur de d peut alors être déterminée et testée. L'attaquant considère ainsi exhaustivement tous les $\vec{\nu}$ dans l'ordre décroissant de leurs probabilités.

[22] Cela implique la capacité de factoriser et de calculer des logarithmes discrets pour une taille de l'ordre de celle de n.

Dans la pratique, il est très probable qu'une grande partie des ν n'aient pas pu être complètement factorisés. Les probabilités exactes des différents t-uples de candidats ne sont alors pas disponibles. Dans ce cas de figure, il nous faut remplacer la notion de *probabilité* par celle, assez proche, de *sélectivité*. Pour une signature observée s, et un module candidat ν, notre sélectivité représente un indice de confiance que l'on peut avoir dans le fait que ν a effectivement été le module utilisé. Pour cela, elle rend compte de toutes les occasions constatées qu'une certaine condition sur s (nécessaire pour la valeur correcte de ν) a été effectivement vérifiée alors qu'elle aurait pu ne pas l'être pour les candidats incorrects. Nous détaillons maintenant l'évaluation de cette sélectivité.

Considérons un premier p_i pour lequel $p_i^{a_i}$ divise exactement ν. Supposons tout d'abord que p_i divise μ. Alors, pour les candidats ν incorrects, l'équation $\mathcal{E}_i$ a pour valeur $\perp$ (faux) lorsque $p_i^{a_i} \nmid s$, c'est-à-dire dans la plupart des cas. Il arrive pourtant d'observer que fortuitement $p_i^{a_i} \mid s$. Dans ce cas $\mathcal{E}_i = \top$ (vrai) et le candidat est retenu. Nous rendons compte de cela en multipliant la sélectivité du candidat ν par $p_i^{a_i}$ car, observant s, il n'avait alors qu'une probabilité $1/p_i^{a_i}$ de ne pas être rejeté. Examinons maintenant le cas où p_i ne divise pas μ. Un ν incorrect est alors rejeté toutes les fois où $p_i \mid s$, c'est-à-dire avec une probabilité de $1/p_i$. On multiplie donc par $p_i/(p_i - 1) = p_i^{a_i}/\varphi(p_i^{a_i})$ la sélectivité de ν lorsque l'on observe que $p_i \nmid s$. Allant plus loin, et pour chacun des $q_j^{f_j}$ divisant exactement $\varphi(p_i^{a_i})$, nous pouvons également considérer la probabilité pour un ν incorrect de ne pas se faire rejeter par la condition $v(\mu) < v(s)$. Lorsque $v(s) \leqslant v(\mu)$, c'est-à-dire lorsque la composante en q_j de l'ordre de s modulo $p_i^{a_i}$ divise celle de l'ordre de μ, le candidat n'est pas écarté alors qu'il aurait pu l'être. Dans ce cas, nous intégrons dans la sélectivité le facteur supplémentaire $q_j^{f_j}/q_j^{v(\mu)}$ qui correspond à l'inverse de la probabilité de cet évènement. Enfin, nous devons tenir compte de chaque occasion qu'un candidat incorrect aurait pu avoir de se faire rejeter par le fait d'une incompatibilité des valeurs des logarithmes discrets dans les processus de fusion des contraintes modulaires. Pour chaque paire $((q, v', \lambda'), (q, v'', \lambda''))$, avec $v' \geqslant v''$, la probabilité que ces deux contraintes soient fortuitement compatibles, c'est-à-dire que $\lambda' \equiv \lambda'' \pmod{q^{v''}}$, est égale à $1/q^{v''}$. Là encore, nous tenons compte de ce facteur $q^{v''}$ dans la sélectivité pour chaque fusion n'ayant pas révélé d'incompatibilité.

Informellement, notre sélectivité peut être comprise comme une sorte de probabilité *a posteriori* (non normalisée) de chaque candidat, conditionnellement à la partie *exploitable* de l'observation. L'attaquant peut donc s'en servir pour mener son attaque de la même manière que s'il n'était pas calculatoirement limité. Exploitant successivement les fautes qu'il acquiert, il maintient une liste de candidats $\vec{\nu}$ compatibles avec les observations. À chacun de ces candidats sont associés un paramètre de sélectivité, ainsi qu'une liste de contraintes modulaires sur d. Après un certain nombre de fautes, la résultante CRT des contraintes modulaires de chaque candidat dépasse la taille de n (ou la moitié si e est petit). Il est alors possible de les tester exhaustivement par ordre décroissant de sélectivité.

Remarque 5. Dans le cas où la liste devient trop grande pour être aisément manipulée, l'attaquant peut choisir de n'en retenir que les candidats dont la sélectivité dépasse un certain seuil. Ce faisant, il prend un faible risque d'écarter la bonne supposition.

Résultats

Cette méthode vise à déterminer la liste des t-uples de modules $\vec{\nu}$ compatibles avec un ensemble donné de fautes. Nécessairement, elle réussira toujours[23] à proposer l'hypothèse correcte $\vec{\nu}$, menant ainsi à l'identification de t touches à l'aide de seulement t fautes, ce qui est

[23] Si l'astuce discutée à la Remarque 5 n'est pas utilisée.

évidemment optimal en termes de nombre de fautes requises. Avec une bonne factorisation d'un dictionnaire de 1000 modules, nous avons expérimenté que cette méthode permet de retrouver d avec un petit effort de calcul dans la plupart des cas, et avec aussi peu de fautes requises que ce qui est nécessaire d'après la Table 5.1. Nous pensons que des résultats similaires peuvent être obtenus avec un effort modéré dans le cas d'un dictionnaire de 10 000 modules.

5.6 Conclusion

Dans ce chapitre nous avons proposé la première attaque par fautes qui peut être réalisée contre le RSA en mode standard, pour retrouver l'exposant privé, en ne corrompant que des *éléments publics de la clé*. Dans ce sens, notre contribution peut être vue comme une généralisation des articles de Jean-Pierre Seifert et de James Muir sur l'obtention d'une acceptation d'une fausse signature en corrompant le module. Néanmoins, ce dernier type d'attaque ne permet que de passer une vérification de signature, alors que la nôtre permet une *révélation complète de la clé*.

Notre attaque est proposée en deux modes. Dans le premier mode, l'attaquant n'a besoin d'*absolument aucune connaissance* sur le comportement de la faute pour retrouver l'exposant privé. Cette attaque est très attractive d'un point de vue pratique et représente, à notre connaissance, la seule attaque par faute sur le RSA en mode standard ne requérant aucun modèle de faute ni aucune hypothèse sur la méthode d'exponentiation[24]. Le second mode, basé sur un modèle de faute, s'est révélé particulièrement efficace. Il réduit considérablement le nombre de fautes nécessaires pour retrouver complètement la clé privée. Pour que cette technique fonctionne, l'attaquant n'a pas besoin d'être particulièrement puissant dans le sens où il n'est pas nécessaire qu'il maîtrise l'effet exact de la faute. La faute qu'il produit peut être probabiliste ou imprécise. Deux variantes sont proposées, chacune avec ses avantages, ses inconvénients et ses cas d'utilisation.

Pour résumer, cette contribution nous enseigne que, comme dans le cas des courbes elliptiques [BMM00, CJ05], *on doit également protéger les éléments publics de clé* RSA contre les attaques par fautes.

Problèmes ouverts : Notre attaque peut-elle être adaptée au cas d'une implémentation protégée (contre l'analyse de canaux auxiliaires) par randomisation de l'exposant ? Peut-elle attaquer un schéma de padding aléatoire avec auto-détermination d'aléa tel que RSA-PSS ?

[24] Une attaque par fautes sur le RSA standard indépendante de tout modèle de faute mais supposant une exponentiation modulaire par balayage de l'exposant de la droite vers la gauche a été publiée par Michele Boreale à Fdtc '06 [Bor06].

Chapitre 6

Étude de cas d'une attaque par fautes sur un crypto-processeur DES asynchrone

Sommaire

Nous décrivons dans ce chapitre une étude de cas d'une attaque par fautes sur un crypto-processeur DES asynchrone. Les résultats obtenus et leur analyse ont été publiés à FDTC '06 avec YANNICK MONNET, MARC RENAUDIN, RÉGIS LEVEUGLE et PASCAL MOITREL [MRL+06b].

Ce travail a été partiellement aidé par le Ministère Français de la Recherche à travers le projet RNRT Duracell.

Nous souhaitons remercier FÉLIE MBUWA NZENGUET, JEAN-BAPTISTE RIGAUD et ASSIA TRIA du laboratoire de Gardanne de l'École Nationale Supérieure des Mines de Saint-Étienne pour le développement de l'outil de communication entre le dispositif testé et le banc. Nous sommes également reconnaissants à CHRISTOPHE MOURTEL et à NATHALIE FEYT du laboratoire *Sécurité des Cartes à Puce* de Gemalto, pour d'intéressantes discussions et pour leur support.

Ce chapitre propose une attaque pratique par fautes sur deux crypto-processeurs DES asynchrones, une version de référence et une version durcie. Cette attaque utilise une réduction du nombre de tours. Du fait de leur architecture, les circuits asynchrones ont un comportement très spécifique en présence de fautes. Des travaux passés ont montré qu'ils sont une alternative

intéressante pour la conception de systèmes robustes. Notre contribution montre cependant qu'il existe encore des faiblesses que nous sommes capables tout à la fois d'identifier et d'exploiter. L'effet observé de la faute est de réduire le nombre de tours en corrompant le compteur de tours multi-rails protégé par des cellules d'alarme. Le moyen d'injection de faute est un laser. Une description du processus d'injection de faute est présentée, suivie par une discussion sur l'utilisation des résultats pour retrouver la clé. Les faiblesses sont alors théoriquement identifiées et analysées. Enfin, des contre-mesures sont proposées.

6.1 Introduction

Les attaques par fautes sont considérées comme une menace sérieuse par les concepteurs de systèmes embarqués sécurisés. Le niveau de sécurité des algorithmes cryptographiques comme le DES et l'AES repose sur le nombre d'itérations (tours) qui sont calculées. Ross Anderson et Markus Kuhn ont suggéré dans [AK98] qu'un des moyens les plus efficaces pour casser de tels algorithmes est d'implémenter une attaque par fautes qui réduit le nombre de tours. Le potentiel de cette classe d'attaques a été démontré en pratique par Hamid Choukri et Michael Tunstall [CT05] dans le cas d'une implémentation synchrone de l'AES.

Les circuits asynchrones représentent une classe de circuits qui ne sont pas contrôlés par une horloge globale mais par les données elles-mêmes. Du fait de leur architecture spécifique, les circuits asynchrones ont un comportement très différent des circuits synchrones en présence de fautes. Les circuits *quasiment insensibles aux délais* (en anglais, *Quasi Delay-Insensitive*, QDI) sont des circuits asynchrones qui fonctionnent correctement sans considération du délai de propagation des portes. Cette classe de circuits a des potentiels bien reconnus en termes de puissance, vitesse, bruit et robustesse contre les variations de processus, tension, et température [Ren00]. Leur propriété d'insensibilité aux délais les rend intrinsèquement robustes à certaines catégories de fautes comme les fautes de délai [LM04]. De plus cette classe de circuits utilise un codage multi-rails qui peut être considéré, au-delà de son rôle de protection contre l'analyse du courant, comme une contre-mesure native contre les fautes. Ainsi, les circuits *quasiment insensibles aux délais* sont attractifs pour la conception de systèmes tolérants ou résistants aux fautes. Néanmoins, seules des études théoriques ont été faites et, à notre connaissance, aucun résultat concret qui viendrait confirmer ces considérations n'a encore été rapporté.

Yannick Monnet, Marc Renaudin, Régis Leveugle, Sophie Dumont et Fraidy Bouesse ont présenté dans [MRL+05] deux implémentations de DES asynchrones : une implémentation de référence et une version durcie. Des contre-mesures ont été implémentées pour protéger les S-Box contre la cryptanalyse différentielle. Ces contre-mesures ont été validées dans [MRL+06a]. Dans la pratique, la version de référence ne laissait fuir aucune information valable permettant de retrouver la clé secrète, et la version durcie ne laissait fuir aucune information du tout.

Dans ce chapitre, nous présentons une attaque concrète par fautes sur ce DES asynchrone, qui permet de réduire le nombre de tours et révèle ainsi des faiblesses du circuit. Les fautes sont injectées au moyen d'un rayon laser. Les faiblesses sont analysées et des contre-mesures sont proposées pour empêcher cette attaque.

Le compteur de tours de la version de référence est une machine d'état asynchrone qui utilise un code 1-parmi-17 : chaque numéro de tour utilise une ligne. Seize lignes sont utilisées pour coder les seize tours. La version durcie implémente le même compteur mais protégé par des cellules d'alarme. Ces cellules sont capables de détecter tout code incorrect généré dans le module compteur, c'est-à-dire tout état utilisant 2 lignes ou plus est détecté. Les alarmes

informent l'environnement quand un code incorrect est détecté.

Notre travail présente une attaque concrète par fautes tout à la fois sur la version de circuit de référence et sur la version durcie. Dans la Section 6.2, nous présentons brièvement la logique *quasiment insensible aux délais*. La Section 6.3 présente les crypto-processeurs DES qui ont été conçus et fabriqués, et détaille l'architecture du compteur module. La Section 6.4 décrit le banc laser qui a été utilisé pour réaliser l'injection de faute. La Section 6.5 explique comment les résultats sont interprétés pour retrouver la clé secrète. Nous présentons les résultats des attaques aussi bien sur le DES de référence que sur le DES durci, montrant une attaque réussie qui n'a pas été détectée par les cellules d'alarme de la version durcie. Dans la Section 6.6, nous analysons les faiblesses qui ont été exploitées. La Section 6.7 propose des contre-mesures et la Section 6.8 conclut ce chapitre.

6.2 Logique quasiment insensible aux délais

Un circuit asynchrone est composé de modules individuels qui communiquent chacun avec les autres par des canaux de communication point-à-point [Ren00]. Ainsi un module donné devient actif quand il détecte la présence d'une donnée entrante. Il effectue alors son calcul et envoie le résultat sur le canal de sortie. Les communications à travers les canaux sont gouvernées par un protocole qui requiert une signalisation bi-directionnelle entre les émetteurs et les receveurs (requête et acquittement). On les appelle protocoles à accusés de réception. Dans la Section 6.6 nous présentons une injection de faute théorique qui exploite les propriétés d'un protocole à accusés de réception. Un schéma de codage de donnée 1-parmi-n est généralement utilisé pour implémenter des protocoles à accusés de réception insensibles aux délais. Le code double-rail [Ren00], c'est-à-dire 1-parmi-2, est le plus couramment utilisé car c'est un bon compromis entre la vitesse et le coût en surface. Ce schéma peut être étendu à des codes 1-parmi-n, dans lesquels 1 bit est codé avec n lignes. Le crypto-processeur DES décrit dans les sections suivantes utilise un schéma de codage double-rail pour le chemin des données et un schéma de codage 1-parmi-17 pour le module compteur.

6.3 L'architecture du DES asynchrone

À la base, le crypto-processeur DES asynchrone est une structure itérative basée sur trois boucles à autogestion temporelle synchronisées par des canaux de communication (Figure 6.1). Le canal "Sub-Key" synchronise le chemin des données du chiffrement. Plus de détails sur la conception de l'architecture peuvent être trouvés dans [MRL+06a].

"CTRL" est un ensemble de canaux générés par le bloc "Controller" (une machine d'état finie montrée à la Figure 6.2) qui contrôle le chemin des données tout au long des seize itérations spécifiées par l'algorithme DES. Ce module prend en entrée le code 1-parmi-17 du tour courant, calcule les canaux de contrôle associés à ce tour, et le code 1-parmi-17 du tour suivant. Seize lignes sont utilisées pour coder les seize tours, et la dernière ligne permet de sortir le résultat final.

La Figure 6.3 montre comment de tels signaux de contrôle sont utilisés dans le chemin des données. Le canal double-rail "Crtl_1" contrôle le démultiplexeur comme on peut l'observer dans la table. Quand "Ctrl_1" vaut "01" la sortie O1 est sélectionnée, quand "Ctrl_1" vaut "10" la sortie O2 est sélectionnée. Dans la pratique, O1 est utilisée tout au long des seize itérations de l'algorithme, et O2 est utilisée à la fin pour sortir le résultat final. L'état "11" est inutilisé dans le processus d'exécution normal. Cependant, dans le cas d'une faute qui génère "11" sur le signal

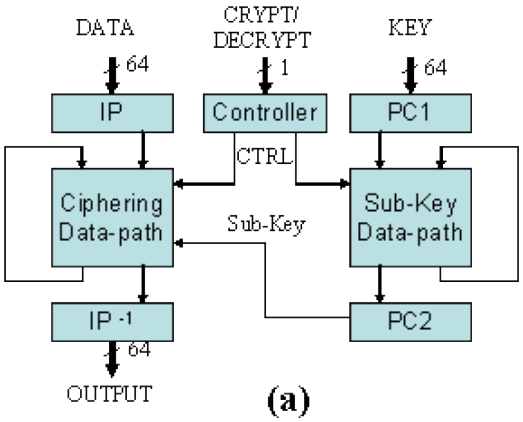


FIG. 6.1 – L'architecture du DES.

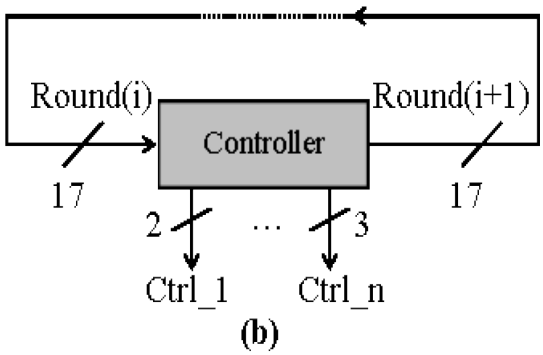
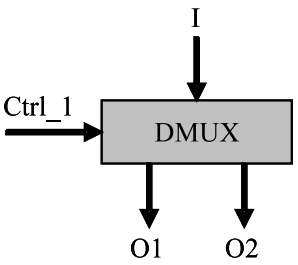


FIG. 6.2 – La machine d'état du contrôleur.



Ctrl_1	Activated output
"01"	O1
"10"	O2
"11 "(faulty)	O1 and O2

FIG. 6.3 – Le comportement du démultiplexeur.

"Ctrl_1", à la fois O1 et O2 sont activés. C'est la façon dont une implémentation standard de démultiplexeur se comporte en présence de fautes (Figure 6.3).

Plusieurs contre-mesures ont été implémentées sur le DES durci pour empêcher les attaques par fautes. Comme établi dans [MRL+05], les S-Box sont protégées en utilisant la technique de synchronisation de bus qui fournit une bonne protection avec un très petit accroissement de surface et une faible baisse des performance. Cette contre-mesure a été validée dans [MRL+06a]. Il n'a cependant pas été possible d'appliquer cette technique sur le bloc compteur puisque ce module n'a pas de bloc concurrent avec lequel se synchroniser.

Le compteur a été protégé en utilisant des cellules d'alarme [MAM+03]. Ces alarmes sont implémentées pour détecter tout code invalide généré sur le compteur 1-parmi-17, et tout code invalide généré sur les signaux de contrôle tels que "Ctrl_1". Le signal de sortie d'alarme est stocké dans un registre, permettant ainsi à l'environnement de lire le statut d'alarme et de savoir quel module du circuit a déclenché cette alarme. Dans un produit réel, l'environnement devrait alors appliquer une stratégie de sécurité comme la remise à zéro du circuit ou l'envoi d'une sortie aléatoire pour empêcher l'attaque. Dans notre cas d'étude, nous collectons le signal d'alarme comme information de statut. Les deux circuits ont été fabriqués dans une technologie 130 nm CMOS de STMicroelectronics, avec un agencement du circuit contraint pour faciliter l'injection de fautes dans des blocs particuliers du circuit. Plus de détails sont présentés dans [MRL+05].

6.4 Le processus d'injection de fautes

Pour réaliser l'injection de faute, nous avions le choix entre plusieurs moyens d'injection. Puisque nous voulons être efficace aussi bien dans l'injection de la faute que dans son exploitation, nous avons rejeté les défaillances subites sur le Vcc, les défaillances subites sur l'horloge et la lumière blanche, étant donné qu'elles ne permettent que de faire varier grossièrement l'instant et la durée de la perturbation, sans possibilité de spécifier des positions précises de la faute sur des parties ciblées du circuit. Les défaillances subites sont appliquées sur tout le circuit logique, et la lumière blanche illumine une large surface du circuit puisque le faisceau le plus mince que nous pouvons obtenir est de 1 mm^2. Au contraire, un laser peut reproduire une grande variété de fautes. L'effet produit est similaire à celui de la lumière blanche mais l'avantage du laser est sa directivité qui permet de cibler précisément une petite surface du circuit (par exemple 5 μm^2).

La plate-forme laser est composée d'un ordinateur pour organiser l'injection de faute et la commande du dispositif sous test, une table X-Y pour effectuer la localisation précise de l'aire ciblée du dispositif, et un oscilloscope optionnel pour contrôler que le dispositif sous test reçoit les commandes et envoie les résultats. Le laser lui-même est un laser pulsé *Yag* avec une sortie verte à 532 nm, une énergie réglable de 0 à 100%, et la possibilité de contrôler la taille du faisceau.

Balayage temporel, balayage spatial et réglage de l'énergie

La première étape d'une campagne d'injection de fautes consiste à déterminer l'instant correct où forcer un comportement fauté du dispositif. L'objectif est d'injecter une faute pendant que le processus s'exécute. Dans notre cas, nous savons que le module DES exécute ses 16 tours en 200 ns environ pour la version durcie. Sachant que notre fréquence d'horloge est de 100 MHz, le plus petit intervalle possible entre deux tirs est de 10 ns. Nous avons alors 20 positions temporelles durant le calcul du DES. De plus, si nous voulons mesurer la reproductibilité de la faute, quand une faute ou une alarme est détectée nous pouvons alors décider de ré-exécuter le test N fois avec la même position spatiale et temporelle. Notons qu'avec une telle granularité, nous avons au moins une chance de produire une faute durant chaque tour du DES.

La seconde étape consiste à choisir la bonne aire à balayer. Dans le contexte de notre étude de cas, nous choisissons de cibler le compteur dans une approche "boîte blanche", c'est-à-dire avec la connaissance des coordonnées de ce bloc. Du fait que l'agencement du circuit est contraint, la position du bloc peut être facilement déterminée. Après plusieurs configurations essayées, nous choisissons une taille de faisceau de 220 μm^2 pour balayer le bloc compteur.

Dans le contexte d'un produit réel, un balayage complet du dispositif sous test devrait être effectué pour identifier les aires les plus intéressantes. Cela conduit à un nombre énorme d'injections de fautes et à des campagnes de tests de longue durée.

Les paramètres adéquats pour le laser sont complètement dépendants du dispositif cible. Dans ce travail, les modules DES étaient illuminés en face avant, sachant qu'ils ont été conçus dans une technologie STmicroelectronics de 0,13 μm, avec 6 couches de métal. Une densité d'énergie de 0,8 $pJ/\mu m^2$ était adéquate pour provoquer les erreurs.

6.5 Interprétation des résultats

La campagne d'injection de fautes a consisté en un balayage spatial et temporel du bloc compteur, ce qui représente plus de 5000 tirs pour chaque circuit durant un simple calcul de chiffrement. Environ 40% des tirs ont provoqué des erreurs. Parmi les erreurs révélées, certaines

ont été identifiées comme étant dues à une modification de la séquence de tours. La sous-section suivante présente l'analyse de ce phénomène et la sous-section 6.5.2 montre comment ces résultats peuvent être interprétés pour retrouver la clé. Les résultats obtenus pour le DES de référence et le DES durci sont présentés dans les sous-sections 6.5.3 et 6.5.4.

6.5.1 Modification des séquences de tours

Plusieurs résultats fautés correspondent au calcul d'une séquence de tours corrompue. La séquence de tours correspondant au processus d'exécution correct est noté de la manière suivante :

$$T_c = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17) \quad .$$

T_c représente le calcul des 16 tours augmenté de la sortie du résultat final, notée "17". Dans la machine d'état décrite précédemment, l'opération de sortie (qui comprend la permutation finale IP^{-1}) est effectivement codée 17.

L'expression $i \rightarrow j$ représente une faute injectée au début du tour numéro i et ayant corrompu le compteur en la valeur j. Il s'en suit que le tour numéro i n'est pas calculé et que l'exécution saute alors au tour numéro j. Par exemple, $7 \rightarrow 10$ induit la séquence de tours suivante :

$$T_1 = (1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14, 15, 16, 17) \quad .$$

Le tour 7 n'est pas exécuté. L'exécution saute au tour 10 et poursuit la séquence normale jusqu'à la fin. Des sauts tout aussi bien en avant ($i < j$) qu'en arrière ($i > j$) ont pu être observés sur les deux circuits.

Avant de poursuivre notre analyse, nous devons rappeler brièvement quelques détails de la structure du schéma de dérivations des clés de tour du DES. Les 56 bits de la clé sont tout d'abord séparés en deux moitiés de 28 bits. Ces deux ensembles de bits sont affectés à deux registres nommés C et D. À chaque tour, chacun de ces registres subit une rotation (souvent improprement appelée décalage) de 1 ou 2 bits[25]. Après ce décalage, 24 des 28 bits de C (respectivement, de D) sont utilisés, via une permutation compressive nommée PC2, comme valeurs des sous-clés pour les S-Box 1 à 4 (respectivement, 5 à 8) de ce tour. Chaque clé de tour K_r[26] de 48 bits est ainsi composée d'une partie K_r^C de 24 bits issus de C, et d'une partie K_r^D de 24 bits issus de D. Intrinsèquement à ce schéma de dérivation, il y a donc séparation entre les bits de clé impliqués à chaque tour dans les S-Box 1 à 4, et ceux utilisés dans les S-Box 5 à 8.

Puisque le compteur contrôle aussi bien le chemin de données que le chemin des clés de tour, la faute $i \rightarrow j$ corrompt également la séquence des rotations de la clé :

$$\begin{aligned} S_c &= (1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1) \\ S_1 &= (1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1) \quad . \end{aligned}$$

S_c représente la séquence des rotations de la clé pour une exécution correcte, et S_1 est la séquence qui est calculée dans le cas de la faute $7 \rightarrow 10$ décrite ci-dessus. Notons que différentes séquences de tours peuvent correspondre à une même séquence de rotations.

La clé de tour utilisée à un tour donné est caractérisée par l'accumulation des rotations subies par les registres C et D, c'est-à-dire par la somme (modulo 28) du nombre de rotations effectuées à chaque tour depuis le début du DES jusqu'au tour considéré.

[25] Le décalage est vers la gauche ou vers la droite selon qu'il s'agit d'un chiffrement ou d'un déchiffrement.

[26] Tout au long de cette Section 6.5, K_r ne désigne pas la clé de tour du tour numéro r, mais plutôt celle formée après une rotation cumulée de r bits des registres C et D.

D'une séquence de rotations S, on dérive donc une séquence σ de rotations accumulées depuis le début du chiffrement :

$$\begin{aligned}\sigma_c &= (1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28)\\ \sigma_1 &= (1,2,4,6,8,10,12,14,16,18,20,22,23) \quad .\end{aligned}$$

La sous-section suivante montre comment exploiter ces séquences, qui sont cryptographiquement pertinentes puisque ce sont elles qui correspondent à des séquences de clés de tour effectivement utilisées lors du processus de chiffrement.

6.5.2 Exploitation

Dans l'analyse qui suit, nous supposons que l'attaquant a choisi un clair qu'il ne modifiera pas durant sa campagne, et qu'il connaît le chiffré correspondant. Pour ce clair, l'attaquant obtient un ensemble $\{C_i\}_{1\leqslant i\leqslant I}$ de chiffrés fautés. À chaque chiffré fauté C_i est associée une séquences fautée σ_i de clés de tour. A cet ensemble, il convient d'adjoindre les deux séquences de clés de tour $\underline{\sigma}$ et $\overline{\sigma}$. La séquence $\underline{\sigma} = ()$[27] ne contient aucune clé de tour car elle correspond au clair vu comme le résultat de l'exécution d'un DES à zéro tour, et la séquence $\overline{\sigma} = \sigma_c = (1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28)$ est celle associée au chiffré correct. L'ensemble

$$\Sigma = \{\sigma_1,\ldots,\sigma_I\} \cup \{\underline{\sigma},\overline{\sigma}\}$$

qui en résulte contient toutes les séquences de clés de tour associées à l'ensemble des chiffrés connus de l'attaquant.

Pour exploiter Σ, nous proposons d'analyser des paires (σ,σ') de séquences de clés de tour. Nous devons cependant remarquer qu'il n'est normalement pas possible pour l'attaquant de déterminer les valeurs des σ_i. Malgré cela, nous allons montrer comment il peut identifier et exploiter, sans les connaître, des paires de séquences de clés de tour "proches". Nous désignons ainsi une paire (σ,σ') ayant un préfixe commun représentant la majeure partie de ces séquences. Pour une telle paire, nous notons π ce préfixe commun, α_0 la dernière valeur de π[28], $(\alpha_1,\ldots,\alpha_t)$ le suffixe de σ et $(\alpha'_1,\ldots,\alpha'_{t'})$ le suffixe de σ'. Sans perte de généralité, nous supposons que les longueurs t et t' des suffixes de σ et σ' vérifient $t' \leqslant t$.

Dans l'exemple suivant, les séquences σ et σ' correspondent aux séquences de tours $9 \rightarrow 12$ et $9 \rightarrow 13$ respectivement :

$$\begin{aligned}\sigma &= (1,2,4,6,8,10,12,14,16,18,20,22,23) &&= \pi \mid \alpha_1 \mid \alpha_2\\ \sigma' &= (1,2,4,6,8,10,12,14,16,18,20,21) &&= \pi \mid \alpha'_1\end{aligned}$$

avec :

$$\begin{aligned}\pi &= (1,2,4,6,8,10,12,14,16,18,20)\\ \alpha_0 &= 20\\ \alpha_1 &= 22\\ \alpha_2 &= 23 \quad (t=2)\\ \alpha'_1 &= 21 \quad (t'=1) \quad .\end{aligned}$$

Nous analysons maintenant certains cas de figure où les suffixes sont courts (t et t' petits). Nous montrons comment chaque cas est identifiable et permet une cryptanalyse amenant à la révélation de la clé.

[27] Séquence vide.
[28] Par convention, $\alpha_0 = 0$ si π est vide.

Nous notons L_i et R_i (respectivement, L'_i et R'_i) les parties de 32 bits de gauche et de droite de la valeur intermédiaire au début du $i^{\text{ème}}$ tour exécuté pour la séquence σ (respectivement, σ'). Nous désignons également par k la longueur du préfixe commun, c'est-à-dire le nombre de tours qui se sont exécutés avec les mêmes clés de tour. Avec ces notations, nous avons $L_k = L'_k$ et $R_k = R'_k$. De plus, les valeurs de L_{k+t}, R_{k+t}, $L'_{k+t'}$ et $R'_{k+t'}$ sont connues.

Cas $t = 1$ et $t' = 0$

Ce cas correspond par exemple à l'exploitation d'une seule faute de type $16 \rightarrow 17$, c'est-à-dire pour laquelle le dernier tour n'a pas été effectué. Dans ce cas, $\sigma = \overline{\sigma}$ correspond au chiffré non fauté, et σ' correspond à la faute $16 \rightarrow 17$. Il est alors possible de déterminer $\alpha_1 = 28$.

Alternativement, ce cas de figure correspond à une seule faute de type $2 \rightarrow 17$, pour laquelle seul le premier tour a été effectué. Dans ce cas, σ est associé au chiffré fauté, $\sigma' = \underline{\sigma}$ correspond au clair, et $\alpha_1 = 1$ est connu.

Enfin, il peut s'agir d'un couple de fautes, comme par exemple $6 \rightarrow 16$ qui donne $\sigma = (1, 2, 4, 6, 8, 9)$, et $6 \rightarrow 17$ qui induit $\sigma' = (1, 2, 4, 6, 8)$, avec des valeurs de α_0 et α_1 égales à 8 et 9 mais inconnues.

Le cas $(t, t') = (1, 0)$ est trivialement identifiable par le fait que L_{k+1} et R'_k, tous deux connus, sont identiques. La cryptanalyse, elle aussi, est immédiate en remarquant que :

- R_k est connu, car $R_k = L_{k+1}$ qui est connu,
- $R_{k+1} \oplus L_k$ est connu, car R_{k+1} est connu, et $L_k = L'_k$ qui est connu,

donc on connaît l'entrée et la sortie de la fonction de tour $f(\cdot, K_{\alpha_1})$ pour le tour $(k+1)$ de la séquence σ. L'attaquant pourra déterminer de l'information sur la clé de tour K_{α_1}. Pour chaque S-Box, l'entrée m (avant XOR avec la sous-clé) sur 6 bits et la sortie y sur 4 bits sont connues. Il est donc possible d'établir la liste des 4 candidats pour la valeur k_{α_1} de la sous-clé de 6 bits, vérifiant :

$$y = S(x) \quad \text{avec} \quad x = m \oplus k_{\alpha_1} \quad .$$

Il en résulte 2 bits d'entropie (4 bits de contrainte) pour chaque S-Box, soit 16 bits d'entropie (32 bits de contrainte) pour la clé de tour K_{α_1}. Bien que l'attaquant ne connaisse pas nécessairement la valeur de α_1, il peut néanmoins les considérer toutes successivement. Considérant les 28 valeurs possibles de α_1 et les 8 bits de clé inconnus qui ne sont pas impliqués dans ce tour, la clé K est retrouvée par recherche exhaustive dans un espace de taille 28×2^{24} candidats.

Cas $t = 2$ et $t' = 0$

Nous présentons d'emblée la cryptanalyse de ce cas de figure plutôt que son identification. Cela est justifié par le fait que l'attaquant identifiera qu'il est bien dans le cas $(t, t') = (2, 0)$ d'après le résultat de sa cryptanalyse, par le simple fait que celle-ci lui aura révélé au moins une valeur possible de la clé.

Lorsque $t = 2$ et $t' = 0$, l'attaquant connaît les entrées et les sorties de chacun des deux derniers tours de la fonction f dans la séquence σ. Ces deux tours impliquent respectivement les clés de tour K_{α_1} et K_{α_2}. Les valeurs de α_1 et de α_2 ne sont pas connues mais elles doivent nécessairement vérifier la relation $\alpha_2 \in \{\alpha_1 + 1, \alpha_1 + 2\}$.

Connaissant l'entrée et la sortie de la fonction $f(\cdot, K_{\alpha_1})$ à l'avant-dernier tour de la séquence σ, il est possible de déterminer une liste $\mathcal{L}^C_{\alpha_1}$ de 256 candidats pour la valeur $K^C_{\alpha_1}$ des 24 bits de K_{α_1} issus de C. On construit de même une liste $\mathcal{L}^D_{\alpha_1}$ de 256 candidats pour la valeur $K^D_{\alpha_1}$ des 24 bits de K_{α_1} issus de D. La connaissance de l'entrée et de la sortie de la fonction $f(\cdot, K_{\alpha_2})$ au dernier tour permet de déterminer de la même manière les listes $\mathcal{L}^C_{\alpha_2}$ et $\mathcal{L}^D_{\alpha_2}$. En supposant par

exemple que $\alpha_2 = \alpha_1 + 1$, il est possible, pour chacun des 2^{16} candidats $(K^C_{\alpha_1}, K^C_{\alpha_2}) \in \mathcal{L}^C_{\alpha_1} \times \mathcal{L}^C_{\alpha_2}$, de tester que les 24 bits de $K^C_{\alpha_2}$ sont compatibles avec une rotation de 1 bit d'une valeur du registre C dont 24 des 28 bits sont déterminés par la valeur de $K^C_{\alpha_1}$. La permutation compressive est telle que ce test implique l'égalité de parties d'exactement 20 bits de $K^C_{\alpha_1}$ et de $K^C_{\alpha_2}$. Chaque candidat $(K^C_{\alpha_1}, K^C_{\alpha_2})$ aura ainsi une probabilité égale à 2^{-20} de vérifier ce test fortuitement, et la probabilité qu'au moins un des 2^{16} candidats survive est bornée supérieurement par 2^{-4}. Un test analogue peut également être mené sur la partie relative au registre D. La probabilité qu'au moins un des 2^{32} candidats $(K_{\alpha_1}, K_{\alpha_2})$, issus de l'observations des entrées et sorties des deux tours, soit fortuitement compatible avec une rotation de 1 bit des registres C et D est au plus égale à $\frac{1}{256}$. Cette analyse doit aussi être menée dans l'hypothèse d'une rotation de deux bits entre les deux tours consécutifs.

Il résulte de cette discussion qu'en menant cette cryptanalyse l'attaquant identifiera qu'il se trouve bien dans le cas de figure $(t, t') = (2, 0)$ par le fait que survive un candidat, probablement unique, $(K_{\alpha_1}, K_{\alpha_2})$. La probabilité de faux positif, c'est-à-dire qu'au moins un candidat ait survécu alors que l'on n'est pas dans le cas $(t, t') = (2, 0)$ est de l'ordre de $\frac{1}{128}$. À partir d'un $(K_{\alpha_1}, K_{\alpha_2})$, l'attaquant déduit les valeurs complètes des registres C et D après une rotation cumulée de α_1 bits. Ne connaissant pas α_1, il reste à l'attaquant 28 candidats possibles pour la valeur de K qu'il discriminera facilement.

Cas $t = 1$ et $t' = 1$

L'identification de ce cas de figure est triviale. Elle résulte de l'observation que l'on doit avoir l'égalité $L_{k+1} = L'_{k+1}$. La probabilité que cette relation soit vérifiée fortuitement alors que l'on n'est pas dans ce cas est de l'ordre de 2^{-32}.

La cryptanalyse du cas $(t, t') = (1, 1)$ exploite le fait que l'on peut établir une relation différentielle entre l'exécution de $f(\cdot, K_{\alpha_1})$ au dernier tour de la séquence σ, et celle de $f(\cdot, K_{\alpha'_1})$ au dernier tour de la séquence σ'. En effet, les entrées de ces fonctions sont identiques et connues. Il s'agit de $R_k = R'_k$, connu comme valant $L_{k+1} = L'_{k+1}$ observé. Quant aux sorties, elles sont différentes mais on connaît leur différentielle comme étant égale à $(R_{k+1} \oplus L_k) \oplus (R'_{k+1} \oplus L'_k) = R_{k+1} \oplus R'_{k+1}$ (car $L_k = L'_k$), où R_{k+1} et R'_{k+1} sont connus. Les valeurs de α_1 et α'_1 ne sont pas connues, mais elles vérifient nécessairement $|\alpha_1 - \alpha'_1| = 1$.

Exploitons maintenant cette relation différentielle entre les exécutions de $f(\cdot, K_{\alpha_1})$ et de $f(\cdot, K_{\alpha'_1})$. Pour chaque S-Box, nous avons :

$$\begin{cases} y = S(x) & \text{avec } \; x = m \oplus k_{\alpha_1} \\ y' = S(x') & \text{avec } \; x' = m \oplus k_{\alpha'_1} \end{cases},$$

où m et $y \oplus y'$ sont connus.

Pour chaque valeur candidate de k_{α_1}, il est possible de calculer y, puis y', dont on peut déduire en retour 4 valeurs possibles de $k_{\alpha'_1}$. Considérant tout d'abord la partie de la clé de tour relative au registre C, pour chaque valeur de $K^C_{\alpha_1}$ il est possible de dériver 20 bits de contrainte (liée à la rotation de C) que doit vérifier chacun des $4^4 = 256$ candidats $K^C_{\alpha'_1}$. Considérant ainsi les $2^{24} \times 256 = 2^{32}$ couples $(K^C_{\alpha_1}, K^C_{\alpha'_1})$, environ 2^{12} seulement d'entre eux survivront à ce test. Procédant de la même manière pour la partie relative au registre D, il apparaît que l'on peut déterminer une liste de 2^{24} couples $(K_{\alpha_1}, K_{\alpha'_1})$ possibles. Pour chacun d'entre eux, les registres C et D après une rotation cumulée de α_1 bits (par exemple) sont totalement déterminés. On peut en déduire 28 valeurs possibles pour la clés K correspondant aux 28 hypothèses pour α_1. L'attaquant détermine alors la clé par une recherche exhaustive parmi 28×2^{24} candidats.

Cas $t = 2$ et $t' = 1$

Dans le cas où $t = 2$ et $t' = 1$, l'entrée et la sortie de la fonction $f(\cdot, K_{\alpha_2})$ correspondant au dernier tour de la séquence σ sont connues. En effet, d'une part l'entrée R_{k+1} de cette fonction est connue car égale à la partie gauche L_{k+2} du chiffré observé, d'autre part la sortie $L_{k+1} \oplus R_{k+2}$ de $f(\cdot, K_{\alpha_2})$ est connue également car R_{k+2} est la partie droite du chiffré et $L_{k+1} = R_k = R'_k = L'_{k+1}$ est égal à la partie gauche du chiffré observé correspondant à la séquence σ'.

Par ailleurs, il est possible d'établir, comme dans le cas $(t, t') = (1, 1)$, une relation différentielle entre l'exécution de $f(\cdot, K_{\alpha_1})$ à l'avant-dernier tour de la séquence σ, et celle de $f(\cdot, K_{\alpha'_1})$ au dernier tour de la séquence σ'.

Les valeurs de α_1, α_2 et α'_1 ne sont pas connues, mais il est possible de se convaincre qu'elles vérifient nécessairement $\alpha_1 = \alpha'_1 + 1$ et $\alpha_2 \in \{\alpha_1 + 1, \alpha_1 + 2\}$.

L'exploitation de la relation différentielle entre les exécutions de $f(\cdot, K_{\alpha_1})$ et de $f(\cdot, K_{\alpha'_1})$, peut être menée comme dans le cas de figure précédent et aboutit à la détermination d'une liste de 2^{24} couples $(K_{\alpha_1}, K_{\alpha'_1})$ possibles.

Nous pouvons alors exploiter la connaissance de l'entrée et de la sortie de la fonction $f(\cdot, K_{\alpha_2})$. Supposons par exemple que $\alpha_2 = \alpha_1 + 1$. Pour chacun des 2^{12} couples candidats $(K^C_{\alpha_1}, K^C_{\alpha'_1})$ issus de la phase précédente, il est possible de déduire l'intégralité de la valeur du registre C après une rotation cumulée de α_1 bits. La valeur de $K^C_{\alpha_2}$ est donc déterminée de manière unique, et peut être testée vis-à-vis des 16 bits de contrainte liée à l'observation des 16 bits de sortie des S-Box 1 à 4 dont les entrées sont connues. La probabilité qu'au moins un des couples $(K^C_{\alpha_1}, K^C_{\alpha'_1})$ passe ce test est de l'ordre de $2^{12} \times 2^{-16} = 2^{-4}$. Menant la même analyse relativement au registre D, la probabilité que survive au moins un couple $(K_{\alpha_1}, K_{\alpha'_1})$ est de l'ordre de 2^{-8}.

Il est donc possible à l'attaquant d'identifier qu'il est bien dans le cas $t = 2$ et $t' = 1$ par le fait que la cryptanalyse décrite ci-dessus suggère au moins une valeur de $(K_{\alpha_1}, K_{\alpha'_1})$. Considérant la possibilité que α_2 peut aussi être égal à $(\alpha_1 + 2)$, la probabilité de décider à tort que l'on est dans le cas de figure $(t, t') = (2, 1)$ est ici aussi de l'ordre de $\frac{1}{128}$. Dans le cas où une valeur de $(K_{\alpha_1}, K_{\alpha'_1})$ est suggérée, celle-ci est très probablement unique, et détermine de manière unique le contenu complet des registres C et D. Du fait de l'ignorance du nombre de rotations, il est alors nécessaire de discriminer la valeur correcte de K parmi les 28 candidats qui s'en déduisent.

Cas $t = 2$ et $t' = 2$

Lorsque $t = 2$ et $t' = 2$, il est également possible d'exploiter une relation différentielle entre l'exécution de $f(\cdot, K_{\alpha_2})$ au dernier tour de la séquence σ, et celle de $f(\cdot, K_{\alpha'_2})$ au dernier tour de la séquence σ'. Une différence ici avec les cas analogues précédents est que les entrées de ces deux fonctions, bien que connues, sont différentes. Cela ne nuit cependant pas à l'exploitation de cette situation, et pour chaque S-Box, il est possible d'écrire :

$$\begin{cases} y = S(x) & \text{avec } x = m \oplus k_{\alpha_2} \\ y' = S(x') & \text{avec } x' = m' \oplus k_{\alpha'_2} \end{cases},$$

où m, m' et $y \oplus y'$ sont connus.

Les valeurs de α_1, α_2, α'_1 et α'_2 sont inconnues, mais sans perte de généralité nous pouvons

poser que $\alpha_1 = \alpha'_1 + 1$. Il n'existe alors que deux possibilités pour les valeurs de α_2 et α'_2 :

$$\text{Cas I : } \begin{cases} \alpha_2 = \alpha_1 + 1 \\ \alpha'_2 = \alpha_1 \end{cases},$$

$$\text{Cas II : } \begin{cases} \alpha_2 = \alpha_1 + 1 \\ \alpha'_2 = \alpha_2 \end{cases}.$$

Commençons par supposer que nous sommes dans le cas I. Nous pouvons exploiter la relation différentielle entre $f(\cdot, K_{\alpha_2})$ et $f(\cdot, K_{\alpha'_2})$. Pour chacun des 2^{24} couples $(K_{\alpha_2}, K_{\alpha'_2})$ qui sont ainsi suggérés, il est possible de déterminer l'intégralité des registres C et D, et donc également la valeur de $K_{\alpha'_1}$ (K_{α_1} est connu comme étant égal à $K_{\alpha'_2}$). De L_{k+2}, R_{k+2} et K_{α_2}, on déduit alors la valeur de R_k. Remarquons que si l'on procède de même à partir de L'_{k+2}, R'_{k+2} et $K_{\alpha'_2}$, on doit nécessairement trouver la même valeur $R'_k = R_k$ puisque les couples $(K_{\alpha_2}, K_{\alpha'_2})$ vérifient justement que la différentielle de sortie des derniers tours est égale à $R_{k+2} \oplus R'_{k+2}$.

À partir des valeurs de K_{α_1} et $K_{\alpha'_1}$, ainsi que de la valeur d'entrée commune des fonctions $f(\cdot, K_{\alpha_1})$ et $f(\cdot, K_{\alpha'_1})$ aux avant-derniers tours, il est possible, connaissant également R_{k+1} et R'_{k+1}, de vérifier si la contrainte de 32 bits $L_k = L'_k$ est satisfaite. La probabilité pour qu'au moins une des 2^{24} paires $(K_{\alpha_2}, K_{\alpha'_2})$ survive fortuitement à ce test est de l'ordre de 2^{-8}.

Examinons maintenant le cas II. Dans ce cas, $K_{\alpha_2} = K_{\alpha'_2}$ et l'analyse de la relation différentielle sur les derniers tours n'est pas identique aux cas précédents. Nous sommes plutôt ici dans une situation similaire à celle de l'*analyse différentielle de fautes* d'Eli Biham et Adi Shamir [BS97] sur le dernier tour du DES. Pour chaque S-Box, nous connaissons les entrées et la différentielle de sortie. Il est donc possible de déterminer un ensemble de valeurs possibles pour chaque sous-clé k_{α_2}[29], dont le produit ensembliste fournit une liste de candidats pour K_{α_2}.

Les espérances du nombre de candidats pour chacune des 8 sous-clés sont respectivement égales à 7,54, 7,66, 7,58, 8,36, 7,73, 7,41, 7,91 et 7,66. Cela donne en moyenne $2^{23,6}$ valeurs possibles pour K_{α_2}. Considérant chacun de ces candidats, il est possible de remonter à la valeur commune de R_k et R'_k. Étant donné que 4 bits de chacun des registres C et D sont encore indéterminés, il existe, pour chaque K_{α_2}, 256 valeurs possibles de couples $(K_{\alpha_1}, K_{\alpha'_1})$. Pour chacun d'eux, on peut calculer les valeurs de sorties des fonctions $f(\cdot, K_{\alpha_1})$ et $f(\cdot, K_{\alpha'_1})$, desquelles on dérive L_k et L'_k. La vérification de la relation $L_k = L'_k$ permet de ne retenir en moyenne que $2^{23,6+8-32} = 2^{-0,4}$ valeurs fortuites de triplets $(K_{\alpha_2}, K_{\alpha_1}, K_{\alpha'_1})$.

Lorsqu'aucun tel triplet n'est suggéré, l'attaquant est certain qu'il ne se trouve pas dans le cas de figure $(t, t') = (2, 2)$. En revanche, si un ou plusieurs triplets sont suggérés alors le doute est permis puisque la probabilité de faux positif n'est pas faible. Il lui est alors néanmoins possible de tester exhaustivement les 28 variantes de clés K associées à chacun de ces triplets.

Autres cas

Nous avons montré que dans chacun des 5 cas de figures où $t \leqslant 2$, il est possible à l'attaquant, d'une part d'identifier quasiment sûrement (sauf éventuellement pour le cas $t = t' = 2$) qu'il se trouve dans ce cas, d'autre part de mener une cryptanalyse efficace permettant de retrouver la clé (parfois simultanément à l'identification du cas).

[29] Si l'on ne se trouve pas dans le cas de figure où $t = 2$ et $t' = 2$, alors pour chaque S-Box, la différentielle de sortie n'est plus nécessairement correcte, et il se peut que la liste des valeurs de sous-clés possibles pour cette S-Box soit vide. Si cela devait arriver, l'attaquant identifierait immédiatement qu'il n'est pas dans le cas de figure envisagé.

Ces cas correspondent à un grand nombre de paires de fautés obtenues dans nos expériences.

Dans les cas où $t > 2$, la difficulté de la cryptanalyse augmente avec t. Notons que le cas $t' = 0$ correspond à un attaquant qui connaît l'entrée et la sortie d'un DES à t tours. Pour une valeur de t pas trop grande, et si l'attaquant fait varier l'entrée, il est possible d'appliquer les techniques classiques de cryptanalyse linéaire [Mat94a, Mat94b]. On remarque que la cryptanalyse différentielle [BS91, BS93] n'est pas applicable dans ce cas car l'attaquant connaît l'entrée des t derniers tours de la séquence σ mais ne la choisit pas. On note également que l'applicabilité de la cryptanalyse linéaire nécessite de trouver un moyen d'identification du couple (t, t'). L'exploitation des cas où $t > 2$ semble donc être assez difficile.

6.5.3 Résultats sur le DES de référence

La moitié des erreurs obtenues expérimentalement ont conduit le circuit à un blocage. Dans ce cas, l'attaque est détectée et aucune information exploitable n'est donnée à l'attaquant pour retrouver la clé. Parmi les erreurs restantes, 50 résultats ont été identifiés comme des séquences fautées qui nous ont donné un ensemble Σ assez large pour permettre une cryptanalyse. 8 paires de chiffrés ont offert la situation très favorable où $t = 1$ et $t' = 0$.

De plus, certaines des séquences correspondent à un calcul de DES réduit de 2 jusqu'à 10 tours. Les fautes produisant ces échantillons sont reproductibles et peuvent être utilisées pour mener une cryptanalyse différentielle ou linéaire [LH94].

Dans la plupart des cas, l'injection de faute a probablement produit un code incorrect sur le compteur. Cela génère alors des signaux de contrôle erronés. Comme expliqué à la Section 6.3, un code de contrôle incorrect généré dans une structure similaire à un démultiplexeur est capable de sortir un résultat très précoce. Puisqu'aucune alarme n'est implémentée, celui-ci n'est pas détecté.

6.5.4 Résultats sur le DES durci

Le compteur du DES durci a le même comportement que celui de la version de référence, à la différence qu'ici les codes erronés déclenchent des alarmes. Comme on pouvait s'y attendre, la plupart des erreurs introduites dans le circuit ont provoqué un code erroné. Néanmoins, quelques erreurs sont restées non détectées. L'injection de faute corrompait le compteur, mais générait un code valide. Nous avons déterminé deux types d'erreurs :

- Les erreurs qui peuvent être reproduites : la Figure 6.4 montre une injection de faute de type $16 \rightarrow 17$. Le signal "End_DES" indique que le calcul du DES est terminé. Comme on peut le voir sur la figure, le calcul fauté est plus court d'environ 12 ns que le calcul correct, ce qui correspond au temps d'exécution d'un tour. Dans cet exemple un DES à 15 tours a été calculé et aucune alarme n'a été déclenchée. Cette erreur est reproductible : 10 tirs avec les mêmes paramètres spatiaux/temporels ont produit 10 fois la même erreur.
- Les erreurs qui ne semblent pas ou difficilement reproductibles : plusieurs séquences fautées qui correspondent à des réductions de tours ont été obtenues. Par exemple, un échantillon a montré un DES à 9 tours qui peut aider à retrouver la clé s'il est combiné à une autre séquence proche (en termes de courts suffixes distincts). Ce résultat est cependant apparu difficile à reproduire.

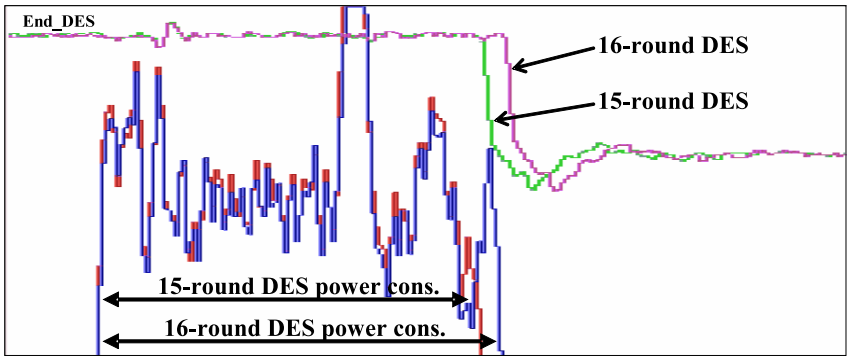


FIG. 6.4 – Signal de consommation électrique d'une séquence fautée non détectée sur le DES durci, où l'on voit le chiffrement tronqué du 16ème tour.

6.6 Analyse théorique des faiblesses

Nous proposons deux modèles de comportement pour comprendre les résultats non détectés sur le DES durci. La première hypothèse est une transition de bit multiple : 2 transitions de bits interviennent. La première transition de bit désactive la ligne active du compteur 1-parmi-17, alors que la deuxième transition de bit en active une autre. Ces deux transitions de bits doivent intervenir au même instant de sorte qu'un code fauté transitoire n'ait pas suffisamment de temps pour se propager à la cellule d'alarme. De plus, puisque les attaques sont localisées, les portes doivent être situées proches l'une de l'autre dans l'agencement du circuit.

La seconde hypothèse est une transition de bit simple qui se produit à l'instant idoine dans le protocole de communication. La Figure 6.5 décrit le protocole en quatre phases qui requiert une phase de remise à zéro aussi bien pour les requêtes de données que pour les acquittements. À la phase 1, une donnée valide est détectée. Cette donnée est acquittée en phase 2. La donnée est alors ré-initialisée en phase 3 (phase de remise à zéro) et le signal d'acquittement est baissé en phase 4. Si une faute se produit en phase 3 ou en phase 4 (quand toutes les lignes sont baissées) alors une transition de bit simple est capable de générer un code 1-parmi-n valide. En fonction de certaines conditions temporelles entre les évènements qui séquencent les accusés de réception, la faute peut être insérée sans être détectée.

Ces deux hypothèses doivent être vérifiées et caractérisées dans un environnement de simulation avec l'aide de la base de donnée de l'agencement du circuit.

6.7 Contre-mesures

Pour empêcher l'hypothèse de transition de bit simple, une synchronisation par circuit de contrôle serait suffisante [MRL05]. Cette technique consiste à implémenter un circuit de contrôle redondant pour les accusés de réception avec le compteur. Le circuit de contrôle dédouble la fonction d'accusé de réception, mais pas la fonction logique du compteur (faible charge supplémentaire). On force la faute à se synchroniser avec la donnée réelle. En conséquence, elle est soit filtrée, soit détectée.

Cependant, un schéma complètement redondant est nécessaire pour empêcher les injections de fautes multiples. Cette contre-mesure est coûteuse en terme de surface, mais comme le compteur est un petit module, la charge supplémentaire totale est raisonnable.

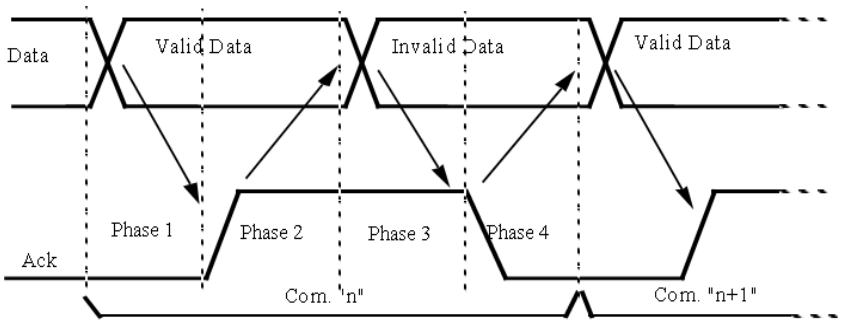


FIG. 6.5 – Protocole à accusés de réception en 4 phases.

6.8 Conclusion

Nous avons présenté une attaque concrète par fautes sur deux crypto-processeurs DES asynchrones. La version durcie du DES a montré un niveau de sécurité plus élevé que celui de la version de référence, puisque la plupart des attaques ont été détectées. Cependant, nous avons montré que la protection proposée ne fournissait pas la sécurité espérée. Des faiblesses ont été rapportées et clairement identifiées. Des techniques de durcissement plus avancées peuvent être implémentées dans certaines parties du circuit, en sus de celles déjà implémentées et caractérisées, pour garantir un très haut niveau de sécurité à un faible coût (surface, vitesse, puissance).

Ce travail a montré que même des systèmes complexes utilisant des schémas de codage redondants et des mécanismes de détection sont sensibles aux attaques par fautes. Néanmoins, cette attaque a été réalisée dans des conditions d'étude avec un agencement du circuit contraint et une bonne connaissance de la conception du dispositif testé. Bien qu'une attaque en cas réel serait beaucoup plus difficile à mener, ce travail a montré sa faisabilité de principe.

Chapitre 7

Cryptanalyse différentielle par fautes de l'algorithme IDEA

Sommaire

Nous proposons dans ce chapitre une étude de la vulnérabilité aux attaques physiques de l'algorithme de chiffrement par bloc IDEA. Portant une attention particulière aux analyses de fautes, nous proposons deux attaques : une analyse de fautes sans effet qui nécessite un modèle de faute particulier et un grand nombre d'injections de fautes, et surtout une analyse différentielle beaucoup plus utile en pratique puisqu'elle permet de retrouver la majeure partie de la clé secrète à l'aide d'un faible nombre d'injections (environ 10 fautes suffisent) dans un modèle de faute aussi général que possible.

La description de ces attaques, découvertes conjointement avec Benedikt Gierlichs, est en cours de soumission.

7.1 Introduction

IDEA (en anglais, *International Data Encryption Algorithm*) est un algorithme de chiffrement par bloc à 8,5 tours introduit en 1991 par Xuejia Lai et James Massey [LM91]. Il permet de chiffrer des blocs de texte clair de 64 bits en des blocs de chiffré de 64 bits, en utilisant une clé de 128 bits. Cet algorithme est aujourd'hui communément utilisé dans différents produits cryptographiques comme PGP, SSH et OpenSSL, et trouve sa place dans des dispositifs embarqués déployés par plusieurs opérateurs de téléphonie mobile ou de télévision à péage.

IDEA a été conçu pour être immune à la cryptanalyse différentielle, et constitue de fait un algorithme difficile à cryptanalyser même dans des versions à nombre réduit de tours. Une partie de la sécurité d'IDEA provient de ce qu'il utilise des opérations dans trois groupes aux propriétés algébriques de natures différentes. Il en résulte que les progrès réalisés pour trouver une faiblesse à cet algorithme sont relativement lents. La meilleure cryptanalyse classique de IDEA connue à ce jour est due à Eli Biham, Orr Dunkelman et Nathan Keller [BDK07] et permet d'attaquer une version réduite à 6 tours à l'aide des chiffrés d'environ 2^{64} textes clairs connus et une complexité équivalente à $2^{126,8}$ chiffrements.

Cet algorithme étant tout à la fois robuste et largement déployé, il nous a paru intéressant d'étudier, malgré une littérature rare à ce sujet[30] [LSP04], sa résistance aux attaques physiques. Sans avoir exploré toutes les voies d'attaques possibles, nous présentons dans la suite de ce chapitre des raisons pour lesquelles certaines *analyses de canaux auxiliaires* (SCA) sur IDEA sont difficiles et/ou peu efficaces. De même, nous expliquons pourquoi l'*analyse de fautes par collision* (CFA) ne permet pas, seule, de retrouver une quantité suffisante d'information sur la clé. Associée à une *analyse de fautes sans effet* (IFA), cette méthode permet néanmoins de retrouver la clé sous l'hypothèse d'un modèle de faute particulier et au prix d'un grand nombre de fautes produites. Un attaquant aura donc tout intérêt à considérer plutôt notre *analyse différentielle de fautes* (DFA) que nous décrivons en détail et qui rend possible la révélation de la majeure partie de la clé secrète à l'aide d'un faible nombre d'injections et dans un modèle de faute aussi général que possible.

Après avoir décrit l'algorithme à la Section 7.2, nous considérons et analysons à la Section 7.3 différentes attaques physiques possibles sur IDEA. Nous y détaillons notamment une attaque différentielle par faute efficace qui constitue la partie substantielle de notre contribution. Le principe de cette attaque à été validé par des simulations intensives dont les résultats sont présentés à la Section 7.4. Nous concluons enfin cette étude à la Section 7.5.

7.2 L'algorithme IDEA

L'algorithme IDEA consiste en la répétition de 8 tours identiques suivis d'une transformation de sortie que l'on peut assimiler à un demi tour supplémentaire. Les données intervenant dans l'algorithme sont toutes des mots de taille 16 bits et sont impliquées dans trois différents types d'opérations :

- le XOR (ou-exclusif) bit à bit, noté $\oplus$,
- l'addition modulo 2^{16}, notée $\boxplus$,
- et la multiplication dans $(\mathbb{Z}/(2^{16}+1)\mathbb{Z})^\star$, notée $\odot$, pour laquelle la valeur 2^{16} est représentée concrètement par le mot de 16 bits égal à 0.

Chaque tour n ($1 \leqslant n \leqslant 8$) utilise six sous-clés Z_i^n ($1 \leqslant i \leqslant 6$). La transformation de sortie utilise quatre sous-clés supplémentaires Z_i^9 ($1 \leqslant i \leqslant 4$). En mode chiffrement, chaque sous-clé est formée de bits de la clé sélectionnés selon le schéma de dérivation des sous-clés décrit à la Table 7.1.

L'algorithme est symétrique dans le sens où la même fonction sert à chiffrer et à déchiffrer. La seule différence se situe au niveau de la dérivation des sous-clés. Pour le déchiffrement, les

[30] Eli Biham et Adi Shamir mentionnent dans [BS97] plusieurs algorithmes de chiffrement par bloc, dont IDEA, vulnérables, selon eux, à l'analyse différentielle de fautes qu'ils présentent dans cet article sur le DES. Il semble toutefois qu'il ne s'agisse là que d'une présomption de vulnérabilité car ils ne détaillent l'application de leur méthode pour aucun des algorithmes qu'ils citent (autres que le DES).

TAB. 7.1 – Les bits sélectionnés dans la dérivation des sous-clés de IDEA.

Tour	Z_1^n	Z_2^n	Z_3^n	Z_4^n	Z_5^n	Z_6^n
$n = 1$	0 – 15	16 – 31	32 – 47	48 – 63	64 – 79	80 – 95
$n = 2$	96 – 111	112 – 127	25 – 40	41 – 56	57 – 72	73 – 88
$n = 3$	89 – 104	105 – 120	121 – 8	9 – 24	50 – 65	66 – 81
$n = 4$	82 – 97	98 – 113	114 – 1	2 – 17	18 – 33	34 – 49
$n = 5$	75 – 90	91 – 106	107 – 122	123 – 10	11 – 26	27 – 42
$n = 6$	43 – 58	59 – 74	100 – 115	116 – 3	4 – 19	20 – 35
$n = 7$	36 – 51	52 – 67	68 – 83	84 – 99	125 – 12	13 – 28
$n = 8$	29 – 44	45 – 60	61 – 76	77 – 92	93 – 108	109 – 124
$n = 9$	22 – 37	38 – 53	54 – 69	70 – 85		

sous-clés sont dérivées de celles utilisées pour le chiffrement en les considérant à rebours, et en en calculant l'inverse vis-à-vis des opérations $\boxplus$ ou $\odot$.

L'entrée de chaque tour n ($1 \leqslant n \leqslant 8$) est formée de quatre mots X_i^n ($1 \leqslant i \leqslant 4$) qui sont tout d'abord mixés avec les sous-clés Z_1^n à Z_4^n. Les valeurs intermédiaires Y_1^n à Y_4^n produites permettent alors de générer les deux mots d'entrée p et q d'une transformation interne MA[31] qui implique les deux sous-clés Z_5^n et Z_6^n et est décrite sur la Figure 7.2. Les deux mots de sortie t et u de cette transformation sont enfin XOR-és avec Y_1^n à Y_4^n pour former l'entrée du tour suivant. La figure 7.1 décrit cette fonction de tour ainsi que la transformation de sortie qui utilise Z_1^9 à Z_4^9 comme pour le début d'un tour normal afin de produire le chiffré C formé des mots C_i ($1 \leqslant i \leqslant 4$).

7.3 Les attaques physiques sur IDEA

7.3.1 Analyse de canaux auxiliaires

Intrinsèquement, et pour une raison différente de son immunité à la cryptanalyse différentielle classique, IDEA est relativement résistant à la variante la plus commune des analyses différentielles de canaux auxiliaires de type DPA ou CPA. La raison en est qu'IDEA ne possède pas de S-Box au sens classique de table de substitution. À la place de l'utilisation de tables, la fonction de confusion est essentiellement assurée par l'opération $\odot$ de multiplication modulo $(2^{16} + 1)$. Il a donc été possible de choisir une taille de sous-clé de 16 bits, plus importante qu'à l'accoutumé, qui rend plus longue la tâche de considérer successivement toutes les suppositions sur la sous-clé jusqu'à identifier celle provoquant le pic indiquant une bonne corrélation entre la valeur réelle de la sortie de la multiplication et sa valeur prédite. Cette remarque n'exprime cependant pas une impossibilité absolue. Une étude de l'applicabilité de la CPA (dans le modèle du poids de Hamming) aux opérations booléennes et arithmétiques utilisées notamment dans IDEA a été menée par KERSTIN LEMKE, KAI SCHRAMM et CHRISTOF PAAR [LSP04] et montre qu'une implémentation non protégée de cet algorithme peut être vulnérable si l'attaquant est prêt à considérer plusieurs fois la comparaison de courbes de corrélations pour 2^{16} suppositions de sous-clés différentes.

[31] L'appellation MA provient simplement du fait que cette transformation est constituée de multiplications et d'additions.

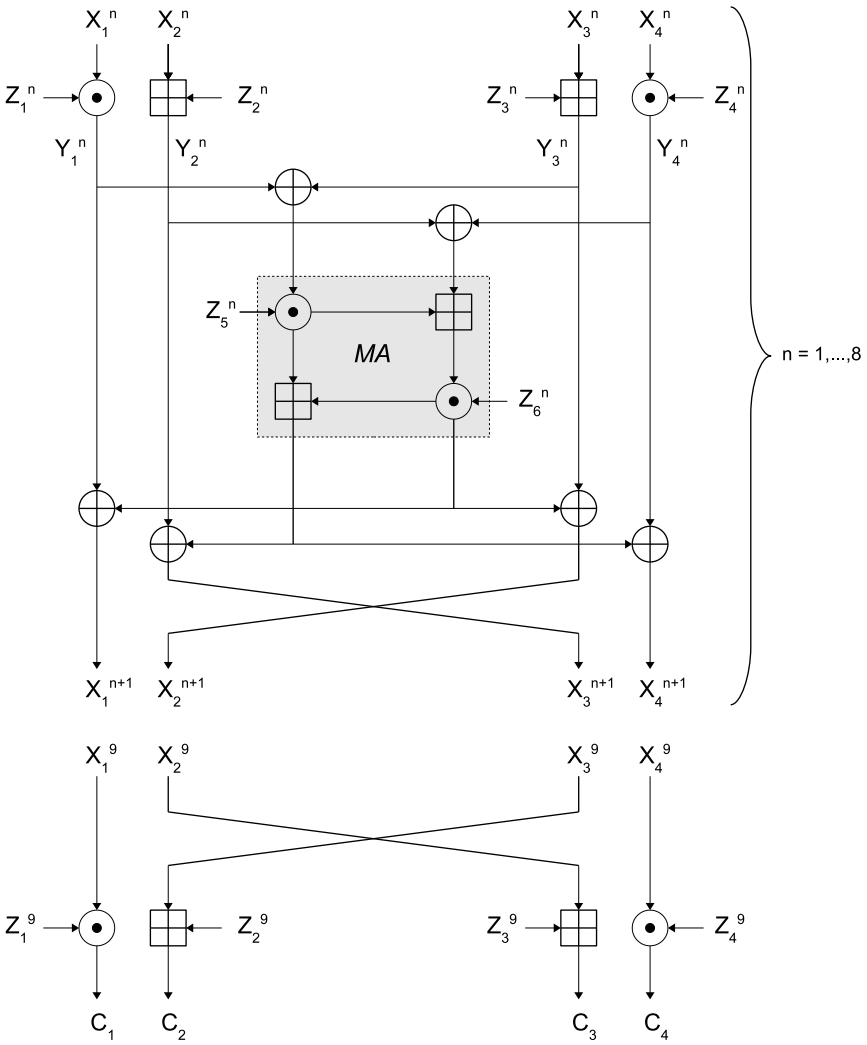


FIG. 7.1 – La fonction tour et la transformation de sortie d'IDEA.

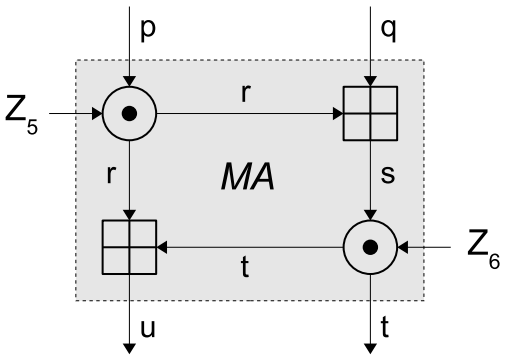


FIG. 7.2 – Détail de la transformation MA.

7.3.2 Analyse de fautes par collision

Concernant la possible vulnérabilité d'IDEA à l'analyse de fautes, nous avons commencé par envisager les attaques par collision. Nous avons trouvé des possibilités de CFA sur IDEA du même type que celle s'appliquant à l'AES décrite en détail au Chapitre 12.

Les analyses qui suivent et les attaques qui en découlent ne requièrent pas que l'attaquant soit capable d'injecter plusieurs fautes durant une même exécution de l'algorithme. Nous supposons un modèle de faute permettant de forcer à une valeur fixe et connue (souvent égale à 0) la sortie d'une opération arithmétique.

Lorsque l'attaquant vise l'opération $\boxplus$, une CFA classique (voir Chapitre 12) permet de retrouver successivement les sous-clés Z_2^1 et Z_3^1 à l'aide de seulement deux injections de fautes et 2×2^{15} commandes de chiffrement sans faute en moyenne.

De la même manière, il est possible de retrouver Z_1^1 et Z_4^1 en supposant qu'une faute peut figer le résultat de l'opération $\odot$. On doit cependant noter que, sur un micro-processeur à usage général, il est plus douteux de pouvoir fixer le résultat de la multiplication que celui de l'addition. En effet, l'addition est très probablement implémentée à l'aide d'une seule instruction machine (au moins sur des architectures à 16 ou 32 bits) sur laquelle on peut vouloir provoquer la faute, alors que la multiplication est nécessairement constituée de plusieurs instructions. Cela est dû, d'une part à la réduction modulo $(2^{16}+1)$ qui doit être réalisée explicitement, d'autre part au test de la valeur particulière 2^{16} qui doit être ramenée à 0. Une possibilité pour réaliser la condition nécessaire à l'attaque sur la multiplication peut néanmoins être envisagée si cette opération se fait par un appel à une routine spécifique et que l'attaquant est capable d'éviter cet appel.

Dans l'hypothèse improbable où un attaquant serait capable de provoquer à sa guise les deux types de fautes considérés ici, il obtiendrait 64 bits de clé. Nous n'entrevoyons pas de possibilité d'obtenir plus d'information sur la clé par CFA[32]. En effet, car même s'il est alors possible de calculer les entrées p et q de la transformation MA du premier tour, provoquer une faute fixant la valeur de sortie d'une des opérations de cette transformation ne permettra pas d'espérer pouvoir trouver une autre entrée de l'algorithme provoquant une collision sur le chiffré. Pour cela, il faudrait que la partie du texte clair que l'attaquant choisit de faire varier pour obtenir cette collision (par exemple X_1^1 ou X_3^1 si la faute a lieu sur l'opération $p \odot Z_5^1$) n'influence que le résultat de l'opération fautée. Ceci constitue une condition nécessaire pour l'applicabilité de la CFA et cette condition n'est pas vérifiée ici puisque tous les mots d'entrée du premier tour sont également impliqués en sortie de ce tour à travers un XOR avec un mot de sortie de la transformation MA. Il n'est donc pas possible d'obtenir une collision pour en dériver Z_5^1 ou Z_6^1 par CFA. Le paragraphe suivant envisage la possibilité de prolonger cette attaque pour obtenir de l'information supplémentaire par rapport à celle obtenue par CFA en employant une méthode plus élaborée.

7.3.3 Analyse de fautes sans effet

Il n'est pas toujours possible d'obtenir une collision avec un chiffré fauté donné en recherchant un message qui la provoque. C'est notamment le cas si l'influence du message dépasse la simple opération fautée. Pour remédier à ce problème, il faut que les chiffrements "sans" et "avec" injection de faute s'appliquent par paires au même texte clair. Les deux chiffrés de la paire seront différents excepté dans le cas rare où la valeur naturelle (sans faute) de la sortie de

[32] Mentionnons toutefois que si l'adversaire est capable aussi bien de chiffrer à messages choisis que de déchiffrer à chiffrés choisis, alors les valeurs de Z_1^1 à Z_4^1 et de Z_1^9 à Z_4^9 lui sont accessibles. Cela représente 86 des 128 bits de la clé.

l'opération fautée est précisément égale à la valeur forcée par la faute. La collision, ou plus exactement l'identité des chiffrés[33], ainsi obtenue met en évidence une faute réellement injectée mais *sans effet* puisque, localement, la donnée pertinente possédait *a priori* la valeur que la faute devait lui faire prendre.

L'observation et l'exploitation de telles coïncidences, appelée *analyse de fautes sans effet* (IFA) est décrite au Chapitre 13 où elle est appliquée à un problème bien plus complexe dans lequel la fonction cryptographique attaquée est inconnue. La Section 13.4 y explique pourquoi ce type d'attaque est particulièrement difficile à mettre en œuvre dans le cas où des contre-mesures de type *ordre aléatoire* ou *délais aléatoires* sont implémentées. Dans ce cas, il n'est plus possible d'interpréter une identité des sorties comme étant équivalente à une valeur précise de la donnée fautée, et l'attaquant doit alors faire face à des faux positifs et à des non détections. De plus, et même si l'implémentation n'est protégée par aucune contre-mesure, une limitation de l'IFA est qu'elle nécessite l'injection de l'ordre de 2^k fautes (où k est la taille du mot machine) là où la CFA n'en requiert qu'une seule.

Malgré les inconvénients inhérents à cette méthode, nous décrivons maintenant comment un attaquant peut utiliser l'IFA pour retrouver une clé d'IDEA.

Considérons tout d'abord le cas où il est possible de figer la sortie de l'addition *et* de la multiplication. Les 64 bits des sous-clés Z_1^1 à Z_4^1 ont pu être préalablement déterminés par CFA et l'attaquant est capable de calculer et de contrôler Y_1^1 à Y_4^1 ainsi que les entrées p et q de la transformation MA. En exécutant successivement des paires de chiffrements (normal/fauté) où la faute est localisée sur la mulplication $(p \odot Z_5^1)$, il est possible de déterminer la valeur de $p \odot Z_5^1$ dès qu'une identité de chiffrés est observée. Une telle paire gagnante permet de déterminer la valeur de Z_5^1 et est obtenue à l'aide de 2^{15} paires en moyenne puisque l'attaquant est capable de contrôler l'entrée p de la multiplication. Connaissant Z_5^1, il est maintenant possible de contrôler l'entrée s de l'opération $(s \odot Z_6^1)$, et de déterminer Z_6^1 de la même manière. 96 bits de clé sont alors connus et les 32 bits restant peuvent être retrouvés par recherche exhaustive. L'attaque a nécessité 4 fautes pour la phase de CFA et 2×2^{15} fautes supplémentaires en moyenne pour la phase d'IFA.

Dans le cas où seule l'addition répond au modèle de faute exigé pour cette attaque, l'attaquant n'a pu déterminer par CFA que les valeurs des sous-clés Z_2^1 et Z_3^1. Il est tout de même possible d'effectuer une IFA sur l'opération $(q \boxplus r)$. Lorsqu'une identité de sorties est obtenue, l'attaquant est alors capable de déterminer $q \boxplus r$. Bien que la valeur de q ne soit pas connue, ce sont 16 bits supplémentaires d'information sur la clé qui sont néanmoins obtenus et qui permettent d'associer une unique valeur de Z_5^1 à chaque supposition pour le couple (Z_1^1, Z_4^1)[34]. Remarquons que même sans connaître la valeur de q, il était néanmoins possible d'en contrôler la différentielle (au sens de l'opération $\oplus$), et donc de choisir la partie X_2^1 du message de sorte que les valeurs de $q \boxplus r$ soient toutes différentes. L'obtention d'une paire gagnante sur $(q \boxplus r)$ nécessite donc 2^{15} fautes en moyenne. L'attaque se poursuit maintenant en visant l'opération $(r \boxplus t)$. Une paire gagnante donnera à nouveau 16 bits d'information sur la clé. À ce stade, l'attaquant est capable, pour chaque supposition sur la valeur de (Z_1^1, Z_4^1), de déterminer complètement l'entrée du deuxième tour. Dans ce deuxième tour, et pour chaque valeur supposée de (Z_1^1, Z_4^1), les valeurs de Z_3^2, Z_4^2, Z_5^2 et Z_6^2 sont d'ores et déjà connues (voir la Table 7.1), l'information sur Z_2^2 est accessible par IFA sur $(X_2^2 \boxplus Z_2^2)$ et celle sur Z_1^2 est obtenue par IFA sur $(q \boxplus r)$ (ou alternativement sur $(r \boxplus t)$). Dès lors, chacune des 2^{32} suppositions sur (Z_1^1, Z_4^1) permet de déterminer complètement le reste

[33] Il est délicat de parler ici de *collision* puisque les entrées de l'algorithme sont les mêmes.

[34] Alternativement, il est aussi possible d'associer une unique valeur de Z_1^1 à chaque supposition pour le couple (Z_4^1, Z_5^1), ou encore une unique valeur de Z_4^1 à chaque supposition pour le couple (Z_1^1, Z_5^1).

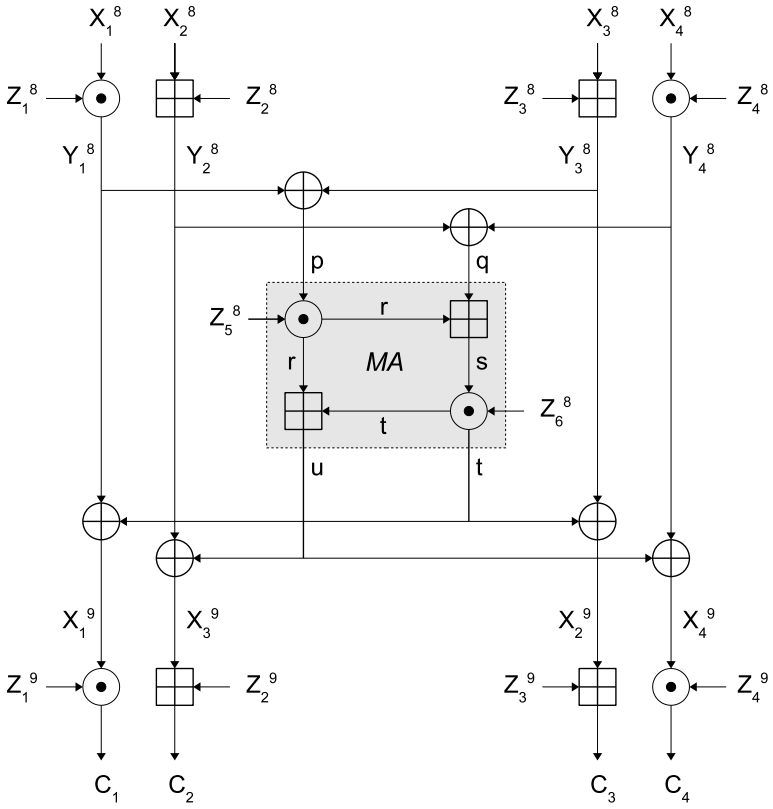


FIG. 7.3 – Le dernier tour et la transformation de sortie d'IDEA.

de la clé qui est donc facilement retrouvée. Dans ce contexte, l'attaque a nécessité 2 fautes pour la phase de CFA et $2^{15} + 3 \times 2^{16}$ fautes supplémentaires en moyenne pour la phase d'IFA.

7.3.4 Une analyse différentielle de fautes efficace

L'attaque que nous proposons maintenant est une *analyse différentielle de fautes* (DFA) qui s'applique sur les 1,5 derniers tours d'IDEA et permet de retrouver de l'information sur la clé à partir de couples (C, C') formés du chiffré normal C et d'un chiffré fauté C'[35] pour le même texte clair M. Le modèle de faute que nous supposons pour cette attaque est tout à fait général puisqu'il s'agit du modèle des fautes aléatoires. L'attaquant est supposé avoir la possibilité de choisir la donnée du dernier tour (tour 8) qu'il souhaite corrompre, mais il n'a pas besoin de se soucier de la valeur qui résultera de cette faute. C'est une propriété intéressante de notre attaque que de ne rien supposer de particulier sur l'effet de la faute, et cela en augmente d'autant sa portée en pratique.

Précisément, nous déterminons 93 des 128 bits de la clé en procédant en trois phases successives. Il reste alors 35 bits de clés indéterminés qu'il est possible de retrouver par recherche exhaustive. Nous décrivons ci-dessous ces différentes phases pour lesquelles nous nous référons à la Figure 7.3 qui présente la partie de l'algorithme concernée par notre attaque en tenant compte de la double permutation des branches 2 et 3 à la fin du tour 8 et au début du tour 9[36].

[35] Tout au long de ce chapitre, les variables notées " ′ " sont relatives à une exécution fautée de l'algorithme.

[36] Il est à noter que cette double permutation produit une étrangeté dans la numérotation des indices puisque

Phase 1 : Retrouver les sous-clés de la transformation de sortie

La première phase permet de retrouver l'ensemble des sous-clés $Z_1^9, \ldots, Z_4^9$ de la transformation de sortie. Tenant compte du fait que notre méthode ne nous permet pas de retrouver les bits de poids fort de Z_2^9 et de Z_3^9, nous révélons ainsi 62 bits de clé.

Supposons que l'attaquant perturbe l'une quelconque des données p, q, r ou s de la transformation MA du dernier tour. Notons que toute modification de l'une des ces valeurs provoque *a priori* une différentielle aussi bien sur u que sur t.

Puisque les valeurs de Y_1^8 et Y_3^8 n'ont pas été modifiées par la faute, nous en déduisons que la différentielle Δ_t se répercute inaltérée sur les mots X_1^9 et X_2^9 :

$$X_1^9 \oplus {X'}_1^9 = X_2^9 \oplus {X'}_2^9 = \Delta_t \ . \tag{7.1}$$

Pour une raison analogue, nous avons également :

$$X_3^9 \oplus {X'}_3^9 = X_4^9 \oplus {X'}_4^9 = \Delta_u \ . \tag{7.2}$$

Connaissant les valeurs des chiffrés de référence et fauté, l'Equation (7.1) permet de déduire une relation liant Z_1^9 et Z_3^9 qui peut être utilisée pour restreindre une liste de candidats possibles pour la valeur du couple (Z_1^9, Z_3^9). En effet, pour chaque supposition sur la valeur de Z_1^9, Δ_t se déduit par la relation :

$$\Delta_t = (C_1 \odot (Z_1^9)^{-1}) \oplus (C'_1 \odot (Z_1^9)^{-1}) \tag{7.3}$$

et l'on peut éliminer toute valeur de Z_3^9 ne vérifiant pas :

$$(C_3 \boxminus Z_3^9) \oplus (C'_3 \boxminus Z_3^9) = \Delta_t \ ,$$

où l'opérateur $\boxminus$ représente la soustraction modulo 2^{16}.

De la même manière, nous pouvons exploiter l'Equation (7.2) pour éliminer tout couple (Z_2^9, Z_4^9) ne vérifiant pas :

$$(C_2 \boxminus Z_2^9) \oplus (C'_2 \boxminus Z_2^9) = (C_4 \odot (Z_4^9)^{-1}) \oplus (C'_4 \odot (Z_4^9)^{-1}) \ . \tag{7.4}$$

Reproduisant cette procédure pour plusieurs couples de chiffrements (normal/fauté), et puisque le candidat correct appartient nécessairement à l'intersection des listes déduites de chaque couple de chiffrés, nous pouvons progressivement restreindre de plus en plus la liste des candidats possibles pour chaque couple de sous-clés. Avec seulement quelques fautes (voir la Section 7.4), il est ainsi possible d'identifier de manière certaine les valeurs de Z_1^9 et Z_4^9, et de déterminer les valeurs de Z_2^9 et Z_3^9 à leur seul bit de poids fort près[37].

Cette première phase de notre attaque permet donc d'identifier 4 candidats pour la valeur de $(Z_1^9, Z_2^9, Z_3^9, Z_4^9)$, qui correspondent à la connaissance des bits 22 à 85 de la clé à l'exception des bits 38 et 54.

c'est la donnée X_3^9 qui est additionnée à la sous-clé Z_2^9 pour former C_2. Même remarque concernant la formation de C_3.

[37] Les bits de poids fort de Z_2^9 et de Z_3^9 ne sont pas identifiables car, pour $i = 2, 3$ et pour toutes valeurs de C_i et C'_i, on a : $\big(C_i \boxminus (Z_i^9 \oplus \text{0x8000})\big) \oplus \big(C'_i \boxminus (Z_i^9 \oplus \text{0x8000})\big) = \big((C_i \boxminus Z_i^9) \oplus \text{0x8000}\big) \oplus \big((C'_i \boxminus Z_i^9) \oplus \text{0x8000}\big) = (C_i \boxminus Z_i^9) \oplus (C'_i \boxminus Z_i^9)$.

Phase 2 : Retrouver la sous-clé Z_6^8

La deuxième phase permet de retrouver les 16 bits de clé correspondant à la valeur de la sous-clé Z_6^8.

Dans cette phase, une faute est injectée pour modifier la valeur de Y_2^8. (Alternativement, l'attaque fonctionnerait de manière équivalente en fautant sur Y_4^8.) Cette faute induit une modification de q qui, tout comme dans la première phase, impacte les sorties t et u de la transformation MA. Y_1^8 n'étant pas modifié, et connaissant les valeurs des chiffrés et de Z_1^9, l'attaquant peut utiliser l'Equation (7.3) pour calculer Δ_t. Il peut aussi, de la même manière, dériver Δ_u de la valeur connue de Z_4^9. Enfin, connaissant Δ_u et Z_2^9 (les deux valeurs possibles pour Z_2^9 sont ici équivalentes), il peut également calculer Δ_q comme étant égal à la différentielle $(Y_2^8 \oplus {Y'}_2^8)$ introduite par la faute.

La possibilité de pouvoir calculer Δ_q est intéressante à plusieurs titres. Non seulement nous allons voir qu'elle permet le succès de cette phase, mais nous pouvons également remarquer que la connaissance de l'effet de la faute peut être utile dans un but plus général de caractérisation du composant attaqué.

Pour chaque paire de chiffrés, et connaissant Δ_t, Δ_u et Δ_q, l'attaquant va commencer par établir la liste TR de toutes les paires (t, r) compatibles avec Δ_t et Δ_u, c'est à dire vérifiant :

$$(r \boxplus t) \oplus (r \boxplus (t \oplus \Delta_t)) = \Delta_u \ .$$

Notons que la résolution de ce type d'équations différentielles d'additions a été étudiée par Frédéric Muller [Mul04], puis par Souradyuti Paul et Bart Preneel [PP05].

Après cela, et pour chaque supposition sur la valeur de Z_6^8, l'attaquant considère chaque paire $(t, r) \in TR$, calcule les valeurs s et s' définies par :

$$\begin{cases} s = t \odot (Z_6^8)^{-1} \\ s' = (t \oplus \Delta_t) \odot (Z_6^8)^{-1} \end{cases}$$

et teste la compatibilité du triplet (r, s, s') avec Δ_q en vérifiant l'équation :

$$(s \boxminus r) \oplus (s' \boxminus r) = \Delta_q \ .$$

Toutes les valeurs supposées de la sous-clé telles que la compatibilité vis-à-vis de Δ_q n'est assurée pour aucun des éléments de TR sont alors écartées de la liste des candidats possibles pour Z_6^8. Faute après faute, cette liste s'amenuise rapidement pour ne conserver finalement que la valeur correcte de la sous-clé recherchée.

Remarque 6. Une difficulté d'ordre pratique peut survenir dans ce processus car la taille de la liste TR peut parfois atteindre plusieurs dizaines de millions d'éléments. Dans un tel cas de figure le nombre de calculs de (s, s') et de vérifications de compatibilité avec Δ_q peut très bien avoisiner 2^{40}. Bien que ce ne soit pas forcément rédhibitoire, il peut être utile de trouver un moyen d'éviter cette complexité calculatoire. Comme solution à ce problème, nous proposons d'acquérir tout d'abord plusieurs fautes, et, pour chacune d'entre elles, de calculer la taille de l'ensemble TR. L'attaquant peut alors choisir d'utiliser en premier la faute générant le plus petit TR pour commencer à épurer la liste, initialement pleine, des candidats sous-clés. Une fois cette liste de candidats réduite, il devient alors possible de considérer les fautes générant de grands ensembles TR sans que cela soit calculatoirement pénalisant.

Phase 3 : Retrouver la sous-clé Z_5^8

Nous décrivons maintenant la troisième phase de notre attaque qui vise à retrouver la valeur de la sous-clé Z_5^8, à partir de la connaissance préalable des sous-clés Z_1^9, Z_2^9, Z_3^9, Z_4^9 et Z_6^8.

Dans cette phase l'attaquant injecte des fautes pour modifier la valeur de Y_1^8 (ou de manière équivalente celle de Y_3^8). Cette modification induit alors une différentielle Δ_p sur l'entrée gauche de la transformation MA dont les sorties t et u sont également affectées. Tout comme dans la deuxième phase, l'attaquant peut utiliser sa connaissance des sous-clés de la transformation de sortie pour déterminer les valeurs de Δ_t, Δ_u et Δ_p.

Nous considérons exhaustivement toutes les valeurs possibles pour le couple (t, r). Pour chacune, nous sommes capables de calculer successivement $t' = t \oplus \Delta_t$, $r' = ((r \boxplus t) \oplus \Delta_u) \boxminus t'$, $s = t \odot (Z_6^8)^{-1}$ et $s' = t' \odot (Z_6^8)^{-1}$. Nous testons alors la compatibilité avec $\Delta_q = 0$ en observant si la condition $(s \boxminus r) = (s' \boxminus r')$ est vérifiée. Pour chaque (t, r) vérifiant la compatibilité avec $\Delta_q = 0$, nous exhaustons toutes les valeurs de Z_5^8, et incrémentons un compteur associé à chacune d'entre elles lorsque la compatibilité avec Δ_p, c'est à dire la condition $(r \odot (Z_5^8)^{-1}) \oplus (r' \odot (Z_5^8)^{-1}) = \Delta_p$ est, elle aussi, vérifiée. Une fois passés en revue tous les couples (t, r), nous pouvons déclarer impossible toutes les suppositions sur Z_5^8 pour lesquelles la valeur du compteur est nulle. Pour ces valeurs de Z_5^8, il n'existe en effet aucun couple (t, r) pour lequel les compatibilités au niveau de Δ_q et de Δ_p sont simultanément vérifiées.

Ici encore, faute après faute, la plupart des valeurs incorrectes de la sous-clé sont progressivement éliminées.

Remarque 7. En simulant cette phase de l'attaque, nous nous sommes aperçus qu'un très petit nombre de fautes (de l'ordre de 5 fautes tout au plus) est toujours suffisant pour réduire la liste des valeurs possibles pour Z_5^8 à seulement 2 candidats. Contrairement au phénomène constaté à la première phase concernant les bits de poids fort de Z_2^9 ou Z_3^9, cette ambiguïte entre ces deux valeurs n'est pas absolue. Dans nos expériences, nous avons remarqué qu'elle pouvait être levée, mais assez difficilement, et parfois seulement après plusieurs dizaines de fautes[38]. Pour simplifier notre analyse, nous considérerons donc que la troisième phase est un succès dès lors qu'au plus deux candidats restent possibles. En conséquence, nous convenons que cette phase retrouve seulement 15 bits supplémentaires de clé.

Bilan des trois phases

Mises bout à bout les trois phases de notre attaque permettent de retrouver 93 des 128 bits de la clé secrète. La section suivante présente des résultats de simulations qui ont validé notre approche et ont permis d'évaluer le nombre d'injections de fautes que chaque phase de notre attaque nécessite.

Remarque 8. Une astuce permet de réduire le nombre de faute nécessaires. En effet, on peut remarquer que les fautes injectées sur Y_2^8 utiles pour la deuxième phase peuvent également être utilisées en partie pour la première phase. En effet, une modification de Y_2^8 provoque une différentielle Δ_t qui peut être exploitée pour la phase 1 puisque, ni Y_1^8, ni Y_3^8 ne sont affectés. De la même manière, une faute modifiant Y_1^8 pour la phase 3 peut être utilisée en phase 1 par l'exploitation de Δ_u rendue possible par le fait que Y_2^8 et Y_4^8 sont inchangés. Il en résulte que n

[38] Sans avoir trouvé d'explication satisfaisante à ce phénomène, nous avons néanmoins remarqué que pour chaque valeur de Z_5, il existe un *compagnon* $\widetilde{Z_5}$ qui possède la propriété que les valeurs $\Delta_r = (p \odot Z_5) \oplus (p' \odot Z_5)$ et $\widetilde{\Delta_r} = (p \odot \widetilde{Z_5}) \oplus (p' \odot \widetilde{Z_5})$, lorsque p et p' sont tirés aléatoirement, sont étonnamment proches en terme de distance de Hamming, et souvent égales. Cette observation mérite une analyse plus poussée et il serait intéressant d'étudier si elle ne pourrait pas ouvrir de nouvelles voies de cryptanalyse d'IDEA.

fautes nécessaires pour la première phase peuvent être économisées par l'exploitation de n fautes dédiées à la phase 2 et autant dédiées à la phase 3.

7.4 Résultats

Nous avons simulé chacune des trois phases de notre attaque. Les résultats que nous présentons ici sont basés sur environ 10 000 simulations pour chacune des deux premières phases, et environ 3000 simulations pour la troisième qui nécessite un temps de calcul plus long.

Nous utilisons deux métriques pour juger de l'efficacité de chacune des phases. La première est l'entropie résiduelle moyenne sur la sous-clé recherchée (l'ensemble des quatre sous-clés dans le cas de la première phase) après avoir exploité m fautes. Il s'agit donc de :

$$\langle h_m(Z_i^n)\rangle = \langle \log_2(\#S_m)\rangle \quad ,$$

où $\langle\cdot\rangle$ est l'opérateur *espérance mathématique* que nous évaluons par une moyenne expérimentale, et où $\#S_m$ est le cardinal de l'ensemble des candidats possibles après avoir exploité m fautes.

Le deuxième critère utilisé est une sorte de probabilité de succès, définie comme la probabilité $p(S_m \leqslant d)$ que le nombre de candidats restant soit inférieur où égal à d.

Phase 1

Pour cette première phase, nous avons commencé par comparer deux stratégies de choix de messages. La première, dite *à message fixe*, consiste à choisir aléatoirement un message et à utiliser celui-ci pour chacune des fautes injectées. La deuxième stratégie, *à messages aléatoires*, renouvelle le choix du message pour chaque faute.

Nous avons observé que la méthode à messages aléatoires était préférable à la méthode à message fixe. Par exemple, sur 2000 simulations dans chacun des cas, l'entropie résiduelle sur l'ensemble Z^9_{1-4} des quatre sous-clés de la transformation de sortie après avoir exploité 3 fautes est de 6,44 bits avec un message fixe et de seulement 4,72 bits dans le cas de messages aléatoires.

Nous avons également cherché à savoir si le choix de la donnée fautée parmi les possibilités p, q, r et s avait une influence sur l'efficacité de l'attaque. Nous avons observé qu'il n'en avait pas ou que celle-ci ne nous apparaissait pas significative.

Nous avons alors réalisé environ 10 000 simulations à messages aléatoires et où la faute modifie la valeur de q. La Table 7.2 présente l'entropie résiduelle moyenne sur Z^9_{1-4} après l'exploitation de $m = 1$ à 8 fautes.

Nous remarquons qu'un petit nombre de fautes suffit pour s'approcher rapidement de la limite intrinsèque d'une entropie résiduelle de 2 bits (les deux bits de poids fort de Z^9_2 et Z^9_3). Notamment, avec seulement 5 fautes, l'entropie résiduelle passe des 64 bits initiaux à seulement 2,38 bits. Toujours avec 5 fautes, la probabilité d'obtenir seulement 4 candidats est de 70%, et la probabilité d'en obtenir au plus 8 est de 93%.

Phase 2

Pour la deuxième phase, tout comme pour la première, un petit nombre de fautes suffisent à rapidement réduire le nombre de candidats pour la sous-clé Z^8_6. La Figure 7.4 permet de remarquer qu'après seulement 5 fautes il reste en moyenne moins de 4 candidats. Après 8 fautes, l'entropie résiduelle moyenne n'est que de 0,41 bits et la sous-clé est déterminée sans aucune ambiguïté dans 62% des cas.

TAB. 7.2 – Phase 1 : Entropie résiduelle moyenne sur Z^9_{1-4}.

Nombre de fautes	Entropie résiduelle sur Z^9_{1-4} (en bits)
$m = 1$	32,03
$m = 2$	9,16
$m = 3$	4,72
$m = 4$	3,02
$\mathbf{m = 5}$	**2,38**
$m = 6$	2,14
$m = 7$	2,06
$m = 8$	2,02

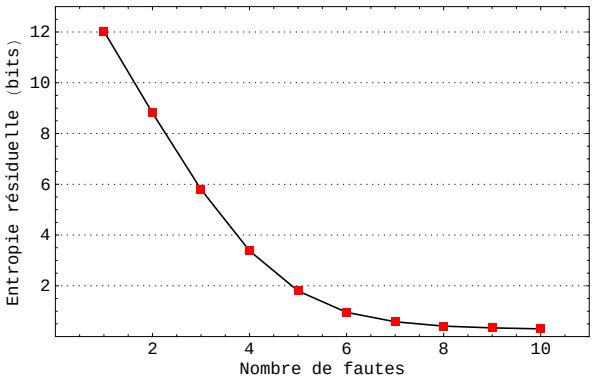


FIG. 7.4 – Phase 2 : Entropie résiduelle sur Z^8_6.

Tab. 7.3 – Phase 3 : Entropie résiduelle moyenne sur Z_5^8.

Nombre de fautes	Entropie résiduelle sur Z_5^8 (en bits)
$m = 1$	9,82
$m = 2$	4,09
$m = 3$	1,43
$\mathbf{m = 4}$	**0,98**
$m = 5$	0,94

Phase 3

La phase 3 de notre attaque est particulièrement efficace. D'après la Remarque 7, nous considérons que cette phase est un succès dès lors que le nombre de candidats pour Z_5^8 est au plus égal à deux. Nous remarquons sur la Table 7.3 que 4 fautes suffisent pour obtenir en moyenne un résultat encore meilleur. Avec seulement 3 fautes, le nombre de candidats pour Z_5^8 est au plus égal à 2 dans 74% des cas.

7.5 Conclusion

Nous avons présenté dans ce chapitre une étude de l'applicabilité des attaques physiques sur l'algorithme de chiffrement par bloc IDEA. Nous nous sommes essentiellement concentrés sur les analyses de fautes, et avons proposé deux attaques. La première est une analyse de fautes sans effet (IFA) qui nécessite un modèle de faute particulier et un nombre très important de fautes. En revanche, notre deuxième attaque est beaucoup plus intéressante d'un point de vue pratique. C'est en effet une analyse différentielle de fautes (DFA) qui ne nécessite aucune hypothèse particulière sur l'effet de la faute et permet de retrouver la majeure partie de la clé secrète à l'aide d'environ seulement 10 fautes.

Troisième partie

Les contre-mesures

Chapitre 8

Algorithme universel d'exponentiation

Sommaire

Nous présentons dans ce chapitre un travail réalisé conjointement avec Marc Joye et publié à Ches '01 [CJ01].

On ne connaît que très peu de contre-mesures pour protéger une exponentiation contre l'analyse simple de canaux auxiliaires. De plus celles-ci sont souvent heuristiques. Ce chapitre présente un algorithme universel d'exponentiation. En liant l'exposant à une chaîne d'additions, notre algorithme peut virtuellement exécuter toute méthode d'exponentiation. Notre objectif est de transférer la sécurité de la méthode d'exponentiation implémentée sur l'exposant lui-même. Nous espérons ainsi concilier les notions de sécurité prouvée de la cryptographie moderne avec les implémentations réelles des crypto-systèmes à base d'exponentiation.

8.1 Introduction

La sécurité d'un crypto-système s'évalue comme la capacité ultime à résister à des attaques dans un modèle d'adversaire donné. Il est très difficile de deviner la stratégie qu'un adversaire va suivre dans sa tentative de casser le système. Les seules hypothèses faites par la cryptographie moderne se réfèrent donc à la *capacité calculatoire* de l'adversaire [Gol97]. Un crypto-système est alors dit *sûr* s'il n'existe aucun adversaire en temps polynomial capable d'acquérir plus d'information utile qu'un utilisateur honnête en déviant du comportement prescrit.

Dans [Koc96, KJJ99], Paul Kocher, Joshua Jaffe et Benjamin Jun ont initié une nouvelle classe d'attaques : les *analyses de canaux auxiliaires* (SCA). Dans ce scénario, un adversaire observe de l'information par canaux auxiliaires (par exemple, la consommation de courant) pendant l'exécution d'un algorithme cryptographique, et ce faisant peut détruire la sécurité du crypto-système pourtant "prouvé sûr". Que signifie donc la sécurité prouvée ? La sécurité est généralement prouvée par réduction : on montre que le seul moyen de casser un crypto-système est de casser une primitive cryptographique sous-jacente (par exemple, la fonction RSA). Lorsque ceci est jugé calculatoirement infaisable, le crypto-système est déclaré sûr. Une attaque par canaux auxiliaires ne viole *pas* cette hypothèse, elle ne fait que considérer d'autres voies pour casser la primitive cryptrographique. En conséquence, nous insistons sur le fait que les notions de sécurité prouvée, ou plus exactement celles de *sécurité prouvée calculatoire*, sont très utiles et doivent entrer dans l'analyse de tout crypto-système.

Malheureusement, il n'existe aucun pendant de la sécurité prouvée traditionnelle, qui serait défini par rapport aux attaques par canaux auxiliaires. Définir un modèle de sécurité pour cette classe entière d'attaques est difficile dans la mesure où l'on ne voit pas comment limiter la puissance de l'attaquant. Malgré certains travaux dans cette direction [MR04, SMY06, IPSW06], l'approche que nous adoptons ici vise plutôt à prouver la sécurité relativement à une attaque particulière.

Ce chapitre s'intéresse particulièrement à l'exponentiation modulaire (par exemple, la fonction RSA) en tant que primitive cryptographique. En utilisant une représentation à base de chaînes d'additions, nous "transférons" la sécurité de la méthode d'exponentiation réellement implémentée vers l'exposant lui-même (qui est la donnée secrète). L'algorithme qui en résulte, et que nous appelons *algorithme universel d'exponentiation*, peut émuler virtuellement toute méthode d'exponentiation. Il lit simplement des triplets de valeurs $(\gamma(i) : \alpha(i), \beta(i))$, chacun signifiant que le contenu du registre $R_{\alpha(i)}$ doit être multiplié par le contenu du registre $R_{\beta(i)}$ et que le résultat doit être écrit dans le registre $R_{\gamma(i)}$. Nous apportons ainsi un certain type de réduction : plutôt que d'analyser avec attention une méthode d'exponentiation spécifique, le programmeur vérifie simplement que l'opération *atomique* $R_{\gamma(i)} \leftarrow R_{\alpha(i)} \cdot R_{\beta(i)}$ ne laisse fuir aucune information "utile" à travers une attaque par canaux auxiliaires. Cette méthodologie est une réminiscence des preuves de sécurité traditionnelles. Dans le cas traditionnel on conjecture la sécurité d'une primitive cryptographique — par exemple, inverser la fonction RSA est infaisable — alors que dans notre cas la sécurité d'une opération atomique est vérifiée expérimentalement — par exemple, on ne peut pas "casser" une multiplication par SPA. La principale différence est que l'hypothèse de sécurité est vérifiée par moins de personnes et donc plus facilement sujette à controverse.

La suite de ce chapitre est organisée de la manière suivante. La prochaine section rappelle la définition d'une chaîne d'additions. Sur la base de celle-ci, nous présentons alors notre algorithme universel d'exponentiation. Dans la Section 8.3 nous discutons des mérites de notre approche d'un point de vue sécuritaire. La Section 8.4 suggère des modifications à notre algorithme de base. Enfin, nous concluons dans la Section 8.5.

8.2 Algorithme universel d'exponentiation

8.2.1 Chaînes d'additions

Nous commençons par une brève introduction aux chaînes d'addition. Nous renvoyons le lecteur à [Knu81] pour plus de détails.

Définition 5. *Une* chaîne d'additions *pour un entier positif d est une séquence $\mathcal{C}(d) = \{d^{(0)}, d^{(1)}, \ldots, d^{(\ell)}\}$ vérifiant*

1. $d^{(0)} = 1$, $d^{(\ell)} = d$, *et*
2. *pour tout* $1 \leqslant i \leqslant \ell$, *il existe* $j(i), k(i) < i$ *tels que* $d^{(i)} = d^{(j(i))} + d^{(k(i))}$.

L'entier ℓ définit la *longueur* de la chaîne $\mathcal{C}$. Une chaîne d'additions est appelée une *chaîne étoile* si pour tout $1 \leqslant i \leqslant \ell$ il existe $k(i) < i$ tel que $d^{(i)} = d^{(i-1)} + d^{(k(i))}$.

Une notion légèrement plus générale est celle de chaîne d'additions-soustractions.

Définition 6. *Une* chaîne d'additions-soustractions *pour un entier d est une sequence $\mathcal{C}(d) = \{d^{(0)}, d^{(1)}, \ldots, d^{(\ell)}\}$ vérifiant*

1. $d^{(0)} = 1$, $d^{(\ell)} = d$, *et*
2. *pour tout* $1 \leqslant i \leqslant \ell$ *il existe* $j(i), k(i) < i$ *tels que* $d^{(i)} = \pm d^{(j(i))} \pm d^{(k(i))}$.

8.2.2 Un algorithme universel

Soit $\mathcal{C}(d) = \{d^{(0)}, d^{(1)}, \ldots, d^{(\ell)}\}$ une chaîne d'additions pour l'exposant d. Ainsi pour tout $1 \leqslant i \leqslant \ell$, nous avons $d^{(i)} = d^{(j(i))} + d^{(k(i))}$. Cela fournit un moyen simple d'évaluer $y = x^d$: Pour $i = 1$ à ℓ calculer

$$x^{d^{(i)}} = x^{d^{(j(i))}} \cdot x^{d^{(k(i))}}$$

et poser $y = x^{d^{(\ell)}}$. Ainsi, pour une chaîne d'additions de longueur ℓ, ℓ multiplications sont nécessaires pour calculer y.

Exemple 1. Une chaîne d'additions pour 5 est $\mathcal{C}(5) = \{1, 2, 3, 5\}$, ainsi $x^1 = x$, $x^2 = x^1 \cdot x^1$, $x^3 = x^2 \cdot x^1$, et finalement $x^5 = x^3 \cdot x^2$.

À l'étape i, $x^{d^{(i)}}$ est évalué comme $x^{d^{(i)}} = x^{d^{(j(i))}} \cdot x^{d^{(k(i))}}$. En supposant que les valeurs $x^{d^{(j(i))}}$ et $x^{d^{(k(i))}}$ sont stockées dans les registres respectifs $R_{\alpha(i)}$ et $R_{\beta(i)}$ et que le résultat $x^{d^{(i)}}$ est écrit dans le registre $R_{\gamma(i)}$, l'exposant d peut être représenté par la *séquence de triplets de registres*

$$\Gamma(d) = \big\{\big(\gamma(i) : \alpha(i), \beta(i)\big)\big\}_{1 \leqslant i \leqslant \ell}, \tag{8.1}$$

chacun signifiant que $R_{\gamma(i)} = R_{\alpha(i)} \cdot R_{\beta(i)}$. (Par convention, la valeur $d = 1$ est représentée par $\Gamma(1) = \emptyset$.)

Nous en déduisons l'algorithme d'exponentiation (pour $d > 1$) présenté à la Figure 8.1.

Notons que $R_{\alpha(1)}$ et $R_{\beta(1)}$ sont initialisés à x car le deuxième élément d'une chaîne d'addition est toujours $d^{(1)} = 2$. Notons également que l'on peut avoir $\alpha(1) = \beta(1)$.

Pour les chaînes étoiles, nous avons $d^{(i)} = d^{(i-1)} + d^{(k(i))}$. Il en résulte que des paires suffisent à représenter d : la valeur de l'indice de registre $\alpha(i)$ est égale à $\gamma(i-1)$ pour tout $1 \leqslant i \leqslant \ell$, et elle peut donc être omise de la représentation. Nous obtenons ainsi la *séquence de paires de registres étoiles*

$$\Gamma^*(d) = \big\{\big(\gamma(i) : \beta(i)\big)\big\}_{1 \leqslant i \leqslant \ell}\ . \tag{8.2}$$

L'algorithme d'exponentiation correspondant est décrit à la Figure 8.2.

Algorithme 8.1 Algorithme universel d'exponentiation

Entrée : $x, \Gamma(d)$

Sortie : $y = x^d$

1. $R_{\alpha(1)} \leftarrow x$
2. $R_{\beta(1)} \leftarrow x$
3. **pour** $i \leftarrow 1$ à ℓ
4. $\quad R_{\gamma(i)} \leftarrow R_{\alpha(i)} \cdot R_{\beta(i)}$
5. **fin pour**
6. **retourne** $R_{\gamma(\ell)}$

FIG. 8.1 – Algorithme universel d'exponentiation.

Algorithme 8.2 Algorithme universel étoile d'exponentiation

Entrée : $x, \Gamma^*(d)$

Sortie : $y = x^d$

1. $R_{\gamma(0)} \leftarrow x$
2. $R_{\beta(1)} \leftarrow x$
3. **pour** $i \leftarrow 1$ à ℓ
4. $\quad R_{\gamma(i)} \leftarrow R_{\gamma(i-1)} \cdot R_{\beta(i)}$
5. **fin pour**
6. **retourne** $R_{\gamma(\ell)}$

FIG. 8.2 – Algorithme universel étoile d'exponentiation.

8.3 Vers la résistance prouvée à la SPA

Un but des fabricants de cartes à puce est de prouver que leurs implémentations résistent aux analyses de canaux auxiliaires. Nous adoptons ici la méthodologie de la cryptographie moderne pour atteindre ce but.

Prenons l'exemple du schéma de chiffrement RSA-OAEP [BR95]. L'exigence minimale de sécurité pour un schéma de chiffrement est la *non inversibilité* (en anglais, *one-wayness*, OW). Cela recouvre la propriété qu'un adversaire ne peut pas retrouver le clair à partir d'un chiffré. Dans certains cas, la moindre information sur un clair peut avoir des conséquences désastreuses. Cette notion est capturée par la *sécurité sémantique* ou la notion équivalente d'*indistinguabilité* [GM84]. Fondamentalement, l'indistinguabilité (en anglais, *indistinguishability*, IND) signifie que la seule stratégie pour un adversaire de faire la différence entre les chiffrés correspondant à deux clairs quelconques est de deviner au hasard. Les attaques les plus fortes que l'on peut imaginer (au niveau protocole) sont celles appelées *attaques adaptatives à chiffrés choisis* (en anglais, *adaptative chosen ciphertext attacks*, CCA2). Ces attaques considèrent un adversaire actif qui peut obtenir le déchiffrement de tout chiffré de son choix. À partir du couple but de l'attaquant (IND) et modèle d'adversaire (CCA2), on déduit la notion de sécurité (IND-CCA2). Dans un scénario (IND-CCA2), un attaquant a accès à un oracle de déchiffrement. Il commence par annoncer une paire de clairs m_0 et m_1. Étant donné un challenge c_b qui est le chiffré de m_0 ou de m_1, l'adversaire doit alors deviner si c_b chiffre m_0 ou m_1 avec une probabilité significativement meilleure que $\frac{1}{2}$. L'attaque est dite adaptative si, après avoir reçu le challenge c_b, l'adversaire peut encore obtenir les déchiffrés de chiffrés choisis, avec la seule restriction de ne pas demander le déchiffrement de c_b.

Mihir Bellare et Philip Rogaway ont introduit en 1994 le schéma de padding OAEP (en anglais, *Optimal Asymmetric Encryption Padding*) [BR95] qui permet de convertir toute permutation à sens unique à trappe en un schéma de chiffrement à clé publique. Bien que Victor Shoup [Sho01] ait démontré que la preuve de sécurité avancée par les inventeurs du schéma général contenait une faille, l'utilisation particulière de ce schéma avec la fonction RSA, nommée RSA-OAEP, a néanmoins été prouvée sûr par la suite [FOPS01]. Si un adversaire est capable de casser la sécurité (IND-CCA2) de RSA-OAEP, alors le même adversaire est capable d'inverser la fonction RSA, c'est-à-dire de calculer une racine $e^{\text{ème}}$ modulo un grand nombre composé $N = pq$ (où p et q sont typiquement des premiers de 512 bits). Comme ceci est réputé infaisable, RSA-OAEP est déclaré prouvé sûr. Notons que cette preuve n'est valide que dans le *modèle de l'oracle aléatoire* [BR93], c'est-à-dire un monde idéal dans lequel les fonctions de hachage se comportent comme des fonctions aléatoires. Pour résumer, la sécurité de RSA-OAEP est prouvée en :

1. identifiant le but de sécurité et le modèle d'adversaire (par exemple, IND-CCA2) ;
2. définissant les hypothèse de travail (par exemple, le modèle de l'oracle aléatoire) ;
3. exhibant une réduction (par exemple, casser la sécurité IND-CCA2 de RSA-OAEP ⇒ inverser la fonction RSA) ;
4. supposant que le problème réduit est infaisable (par exemple, inverser RSA est infaisable) ;
5. déduisant la notion de sécurité (par exemple, la sécurité IND-CCA2 de RSA-OAEP dans le modèle de l'oracle aléatoire).

La sécurité de RSA-OAEP est obtenue au niveau du protocole. Pour casser la sécurité IND-CCA2, l'adversaire a un accès en "boîte noire" à un oracle de déchiffrement : il connaît les entrées et obtient les sorties correspondantes.

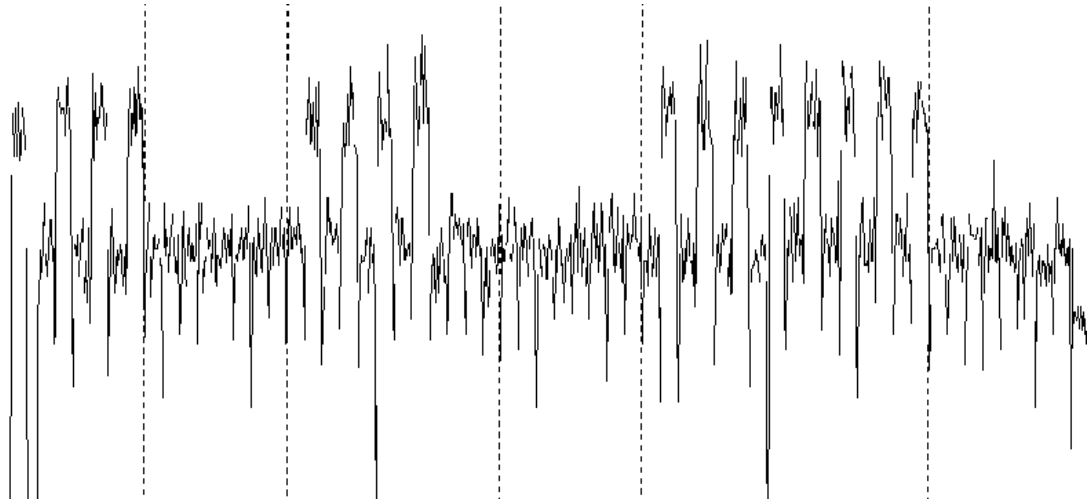


FIG. 8.3 – Trace de courant d'une exponentiation *élever au carré et multiplier*.

Dans le cas des attaques par canaux auxiliaires, l'adversaire est plus puissant : il a accès à certains états internes du calcul. Ainsi, en observant la consommation de courant d'une exponentiation RSA, un attaquant est même parfois capable de retrouver l'exposant secret de déchiffrement d utilisé dans le calcul de $y = x^d \bmod N$, et l'hypothèse (OW) de la fonction RSA n'est donc plus de mise. Supposons par exemple que la fonction RSA soit implémentée naïvement avec la méthode *élever au carré et multiplier*. Comme on peut s'en rendre compte sur la Figure 8.3, l'exposant peut être retrouvé très facilement sur certains anciens composants non sécurisés : un niveau de consommation faible correspond à un carré, et un niveau de consommation élevé correspond à une multiplication.

Dans notre modèle simplifié, nous considérons le but de sécurité fondamental qui rend le crypto-système *incassable* (en anglais, *unbreakability*, UB). Un crypto-système est dit incassable s'il est impossible de retrouver la clé privée. Nous considérons également un attaquant qui a accès à de l'information par canaux auxiliaires. En fonction de l'information par canaux auxiliaires et de la manière dont elle est traitée, nous définissons différents modèles d'adversaire. Dans le modèle d'*analyse simple du courant* (SPA) un attaquant acquiert la trace de courant électrique de l'exécution unique d'un algorithme cryptographique. Nous en déduisons la notion de sécurité UB-SPA. De la même manière, on peut définir UB-DPA (DPA signifiant *analyse différentielle du courant* [KJJ99]) etc... ; on peut aussi considérer d'autres buts de sécurité et en dériver des notions de sécurité comme OW-SPA ou IND-SPA. Il est intéressant de noter que, contrairement à la cryptographie moderne, la définition d'un modèle d'adversaire n'est pas *absolue* : dans une attaque CCA2 un adversaire obtient les clairs correspondant à des chiffrés choisis alors que dans une attaque comme une SPA, la "qualité" de l'information retournée dépend entre autres des outils d'acquisition.

En se concentrant sur la fonction d'exponentiation, et plus particulièrement sur la fonction RSA, on peut montrer que si un adversaire est capable de casser la sécurité UB-SPA de l'algorithme universel d'exponentiation, alors il est aussi capable d'inverser la fonction RSA[39]. Pour casser UB-SPA, un adversaire doit être capable d'obtenir par SPA de l'information sur l'opération

[39]Notons que la menace majeure pour une exponentiation RSA est la SPA ; pour la DPA des contre-mesures efficaces sont connues.

de base $R_{\gamma(i)} \leftarrow R_{\alpha(i)} \cdot R_{\beta(i)}$, c'est-à-dire qu'il doit être capable au minimum de différencier parmi les triplets $(\gamma(i) : \alpha(i), \beta(i))$ et de retrouver leurs valeurs pour casser la propriété UB. En supposant que ceci est infaisable (ce qui peut être vérifié expérimentalement), on a une forte présomption[40] que l'algorithme universel d'exponentiation résiste à la SPA. En conclusion, si RSA-OAEP est implémenté avec l'algorithme universel d'exponentiation, nous avons une forte présomption qu'il résiste aux attaques SPA. On note que la sécurité se situe maintenant au niveau de l'implémentation.

D'un point de vue sécuritaire, l'avantage de notre méthode est évident. Elle ramène le problème d'examiner la sécurité de *tout* algorithme d'exponentiation à celle de l'opération élémentaire $R_{\gamma(i)} \leftarrow R_{\alpha(i)} \cdot R_{\beta(i)}$. Cela rend le travail du programmeur plus facile puisqu'il a une meilleure connaissance des parties sensibles de son algorithme. Qui plus est, la sécurité passe d'un niveau macroscopique (une exponentiation au niveau logiciel) à un niveau microscopique (une multiplication sur crypto-processeur). Enfin, l'analyse ne doit être menée qu'une fois pour toutes et reste valide quel que soit l'algorithme d'exponentiation sous-jacent à la représentation Γ considérée.

Remarque 9. Dans un certain sens, pour assouplir l'hypothèse que l'algorithme universel d'exponentiation est UB-SPA, on peut toujours réaliser des modifications aléatoires et ajouter des opérations fictives au prix d'un temps d'exécution plus long (par exemple, ajouter à une représentation Γ un triplet qui n'affecte pas le résultat final). On peut aussi exploiter la propriété que $R_{\gamma(i)} \leftarrow R_{\alpha(i)} \cdot R_{\beta(i)}$ et $R_{\gamma(i)} \leftarrow R_{\beta(i)} \cdot R_{\alpha(i)}$ conduisent au même résultat. Une autre solution consiste enfin à permuter aléatoirement l'ordre des registres et leurs valeurs dans le cours de l'exponentiation.

En plus de simplifier l'analyse sécuritaire, notre algorithme universel d'exponentiation possède les propriétés suivantes :

- il est *simple* : son implémentation est évidente et les erreurs de programmation seront plus probablement évitées ;
- il est *flexible* : grâce à la généricité de la représentation Γ, il peut virtuellement exécuter tout algorithme d'exponentiation ;
- il est *rapide* : contrairement à la méthode protégée *élever au carré et multiplier systématiquement* (en anglais, *square and multiply always*) qui requiert $2 \log_2 d$ multiplications pour calculer $y = x^d$, notre méthode peut ne requérir qu'à peine $1,25 \log_2 d$ multiplications (voir Section 8.4.1) ;
- il est *économique* : si l'algorithme d'exponentiation sous-jacent à une représentation Γ venait à faillir, il suffirait de ne corriger que cette représentation : une complète reprogrammation n'est pas nécessaire.

Cette dernière propriété est particulièrement intéressante pour une implémentation sur carte à puce. Le code programme est généralement stocké en mémoire ROM via un processus coûteux appelé masquage, alors que la clé privée (c'est-à-dire l'exposant de déchiffrement RSA d) est stockée en mémoire EEPROM lors de la personnalisation de la carte[41]. Ainsi, en cas de fuite du secret ou de mauvaise programmation, il suffit de changer ou de corriger la représentation Γ de l'exposant secret.

[40] Contrairement à la cryptographie moderne, nous ne pouvons pas dire que nous avons une preuve de sécurité car, comme mentionné précédemment, cela dépend de la qualité des mesures.

[41] La personnalisation de la carte comprend le stockage des données relatives à un utilisateur, dont notamment les clés cryptographiques.

8.4 Considérations pratiques

Si nous voulons réaliser une implémentation sur carte à puce des algorithmes proposés (Algorithmes 8.1 et 8.2), nous faisons face à certaines contraintes. Une carte à puce possède un nombre limité de registres et nous devons donc produire des représentations Γ avec un nombre prédéterminé de registres. De plus, une représentation Γ avec moins de registres requiert moins de mémoire pour son stockage. Une autre difficulté peut survenir lorsque l'exposant secret est généré en dehors de la carte par une tierce partie, car il est alors donné dans sa représentation binaire.

Dans cette section nous suggérons deux approches différentes qui modèrent ces limitations.

8.4.1 Génération à bord

Une solution évidente est de produire une représentation Γ à bord de la carte elle-même. On connaît différentes heuristiques efficaces pour produire des chaînes d'additions relativement courtes. Dans [Wal98], Colin Walter suggère la méthode suivante pour calculer $y = x^d$ (voir aussi [BBBD89]).

Définissons $d_0 = d$, $x_0 = x$, et $y_0 = 1$. À chaque étape, écrivons $d_i = m_i\, d_{i+1} + r_i$ pour des valeurs (m_i, r_i) bien choisies. D'où, en posant $x_{i+1} = {x_i}^{m_i}$ et $y_{i+1} = {x_i}^{r_i} y_i$, nous obtenons :

$$\begin{aligned} y = {x_0}^{d_0}\, y_0 &= ({x_0}^{m_0})^{d_1}\, ({x_0}^{r_0}\, y_0) \\ &= {x_1}^{d_1}\, y_1 \;= ({x_1}^{m_1})^{d_2}\, ({x_1}^{r_1}\, y_1) \\ &= {x_2}^{d_2}\, y_2 \;= ({x_2}^{m_2})^{d_3}\, ({x_2}^{r_2}\, y_2) \\ &= {x_3}^{d_3}\, y_3 = \cdots \end{aligned}$$

L'idée derrière la méthode de Colin Walter est de trouver des paires (m_i, r_i) telles que les évaluations de $x_i^{m_i}$ et $x_i^{r_i}$ soient non coûteuses. C'est le cas lorsque r_i se situe sur la chaîne d'additions utilisée pour évaluer $x_i^{m_i}$.

Un telle méthode est très bien adaptée à une implémentation sur carte à puce. Elle est facile à implémenter, et la séquence de registre correspondante, $\Gamma(d)$, ne requiert qu'un registre de plus que la méthode *élever au carré et multiplier* classique. Qui plus est, la longueur moyenne de $\Gamma(d)$ est seulement de $1,25 \log_2 d$, avec une très faible variance. Nous engageons le lecteur à se référer à [Wal98] pour plus de détails.

Notons que le calcul de $\Gamma(d)$ doit être effectué en environnement sécurisé puisque la connaissance de cette chaine d'addition permet de retrouver l'exposant privé d. Par exemple cela peut se faire lors de la personnalisation de la carte.

8.4.2 Fission d'exposant

La seconde solution que nous proposons repose sur l'observation simple que pour tout entier a,

$$x^d = x^a \cdot x^{d-a}\,. \tag{8.3}$$

L'idée de fendre une donnée sensible à déjà été mentionnée dans [CJRR99] comme une contre-mesure générale contre les attaques par analyse différentielle du courant (DPA). On note que les *deux* valeurs, a et $d - a$, sont nécessaires pour retrouver la valeur de d. En d'autres termes, une seule exponentiation x^a ou x^{d-a} nécessite d'être sécurisée.

Étant donnée une séquence de registres pour a, $\Gamma(a)$ ou $\Gamma^*(a)$, nous pouvons calculer $y' = x^a$ et $d' = d - a$, et ainsi $x^d = y' \cdot x^{d'}$. Il y a deux possibilités. La première est, pour un a donné, de stocker une séquence de registres *choisie* (et donc fixe), $\Gamma(a)$, lors de la personnalisation de

la carte. Dans ce cas une représentation étoile, $\Gamma^*(a)$, peut être préférée car requérant moins de mémoire. Un avantage de cette approche est que, puisque la valeur de a est fixe, un effort particulier peut être fourni pour trouver une courte chaîne d'additions pour a.

La deuxième possibilité consiste à calculer "à la volée" la séquence de registres, $\Gamma(a)$ ou $\Gamma^*(a)$, pour un *a aléatoire*. Il y a deux avantages à cette approche. D'abord, il n'est nécessaire de stocker aucune séquence de registre en mémoire non volatile, d'où des économies de place mémoire. Ensuite les évaluations de $y' = x^a$ diffèrent à chaque exécution, rendant plus difficile l'analyse par canaux auxiliaires de cette partie du calcul. Indépendamment, cette fission aléatoire de l'exposant aide également à empêcher les attaques différentielles comme la DPA.

8.5 Conclusion

Dans ce chapitre, nous avons présenté un algorithme universel d'exponentiation. À travers la notion de séquence de triplets de registres, $\Gamma(d) = \{(\gamma(i) : \alpha(i), \beta(i))\}_{1 \leqslant i \leqslant \ell}$ construite sur la base d'une chaîne d'additions pour d, nous avons expliqué comment cette méthode aide à protéger un crypto-système basé sur l'exponentiation contre les attaques simples par canaux auxiliaires de type SPA. En supposant que la multiplication des registres $R_{\gamma(i)} \leftarrow R_{\alpha(i)} \cdot R_{\beta(i)}$, considérée comme opération atomique, ne fuie pas d'information secrète, nous "prouvons" la sécurité de notre implémentation. Il n'y a aucun secret impliqué dans notre algorithme universel d'exponentiation : l'exposant privé d est intimement lié à $\Gamma(d)$, et retrouver d suppose d'être capable de retrouver la séquence complète $\Gamma(d)$. De plus notre algorithme peut être implémenté trivialement et il simplifie grandement l'analyse de sécurité puisque les parties sensibles sont mieux comprises.

Nous espérons que ce premier pas vers la sécurité prouvée des implémentations réelles sera un point de départ motivant de nouvelles recherches sur cet important sujet.

Chapitre 9

Analyse différentielle du courant en présence de contre-mesures matérielles

Sommaire

À la suite des premières annonces d'analyse de la consommation de courant [KJJ98], différentes contre-mesures permettant de contrer ces attaques ont été proposées et mises en œuvre par l'industrie du semi-conducteur. Les premières propositions de protections ne rendaient pas les attaques infaisables, mais ne faisaient que repousser au delà du raisonnable l'effort de travail nécessaire à l'attaquant, tant d'un point de vue expérimental (acquisition des données) que d'un point de vue calculatoire (traitement des données).

Ce chapitre examine différents moyens d'attaquer des dispositifs implémentant des *interruptions aléatoires de processus* (RPI) et une consommation de courant bruitée. Cette contribution a été publiée à Ches '00 conjointement avec Jean-Sébastien Coron et Nora Dabbous [CCD00]. Elle contient, en germe, une démarche et une idée qui se retrouveront exprimées quelques années plus tard dans cette reformalisation que synthétise la CPA [BCO04] présentée au Chapitre 3. La démarche de vouloir exploiter toute l'information disponible est perceptible dès la Section 9.4. Quant à l'idée d'associer un *mot état de référence* à ce qui n'est encore ici que de la DPA *multi-bits*, elle apparaît à la Section 9.5 où est définie une nouvelle fonction de sélection.

9.1 Introduction

Dans les précédentes décennies, la cryptanalyse s'est concentrée sur l'exploitation de faiblesses mathématiques des algorithmes afin de casser les systèmes étudiés. Il en résulte que les crypto-systèmes modernes sont généralement conçus pour bien résister à ces menaces, les attaquants se concentrant maintenant sur l'analyse des fuites par canaux auxiliaires. Parmi celles-ci, les plus connues sont certainement les *analyses du temps d'exécution* (en anglais, *Timing Analysis*, TA), les *analyses simples du courant* (SPA) et les *analyses différentielles du courant* (DPA).

La plupart du temps, les cœurs cryptographiques des produits utilisés ne sont pas isolés dans des lieux parfaitement invulnérables. Il est connu depuis longtemps que la durée d'exécution, la consommation de courant, les émissions dans le domaine des fréquences radio, l'intensité du champ magnétique, etc... peuvent laisser fuir certaines informations sur des données sensibles. Dans un premier temps les cryptographes ont conclu que ces fuites ne pourraient révéler que des informations partielles, ne causant donc aucun danger réel. Ce n'est qu'en 1996 que Paul Kocher [Koc96] a démontré que les attaques par canaux auxiliaires étaient suffisamment efficaces pour permettre de retrouver les clés secrètes de nombreux crypto-systèmes. Des différences dans les durées d'exécution ont été les premières à être exploitées, et en 1999 il a été montré que des mesures de consommation de courant, si elles sont analysées avec attention, peuvent également révéler de l'information sensible [KJJ99]. Ces dangers ayant été mis au jour, analysés et mieux compris, différentes contre-mesures sont maintenant en cours d'étude afin de minimiser l'impact des attaques par canaux auxiliaires en réduisant le signal qu'elles peuvent exploiter ou en le rendant inutile.

En suivant un schéma de réaction plus ou moins uniforme, la plupart des fabricants ont proposé des moyens logiciels et matériels pour cacher et transformer aléatoirement les données sensibles. Ce chapitre s'intéresse particulièrement à la DPA sur des systèmes dans lesquels des contre-mesures au niveau matériel ont été implémentées. Les expériences décrites ci-dessous ont été menées avec succès sur le DES, et prouvent que certaines contre-mesures, initialement considérées comme heuristiquement suffisantes, ne garantissent pas le niveau de sécurité annoncé.

La Section 9.2 rappelle brièvement la DPA et explique comment mener l'attaque sur des dispositifs implémentant les *interruptions aléatoires de processus*, et le bruitage de la consommation de courant. La Section 9.3 s'intéresse à une première méthode pour éliminer cette protection au niveau matériel. La Section 9.4 améliore cette méthode dans la mesure où les recommandations de la Section 9.5 sont prises en compte.

9.2 DPA en présence d'interruptions aléatoires de processus

Les attaques par analyse du courant isolent de l'information corrélée aux opérations ou aux données manipulées en examinant la consommation de courant du dispositif. Suivant la terminologie introduite par Paul Kocher, l'analyse simple du courant (SPA) consiste à analyser directement la consommation de courant d'un dispositif, tandis que l'analyse différentielle du courant (DPA) pointe des corrélations entre la donnée manipulée et l'information du canal auxiliaire.

9.2.1 Analyses différentielles de courant

La DPA peut être facilement menée sur le premier tour d'un DES si le message clair est disponible, ou sur son dernier tour si le chiffré est connu. Nous rappelons ici la DPA classique sur le premier tour d'un DES. Étant donné le clair et la sous-clé de tour, l'attaquant peut calculer

TAB. 9.1 – Nombre de courbes en fonction de la fréquence d'échantillonnage.

Fréquence d'échantillonnage (MHz)	50	100	500	1000
Nombre de courbes	600	500	120	100

l'entrée des fonctions S-Box et, par lecture de table, leur sortie. Telle quelle, la DPA sur le DES est menée une fois par S-Box et permet de déterminer des bits de clés six par six en ciblant la sortie d'une S-Box. Pour mener une DPA, différentes courbes de consommation de courant du dispositif doivent d'abord être recueillies. Dans l'attaque de base les courbes de consommation de courant sont regroupées selon la valeur observée d'un des quatre bits de sortie de la S-Box. Si le bit vaut 1 les valeurs de courant sont ajoutées, s'il vaut 0 elles sont soustraites pour calculer une *courbe différentielle*. Un attaquant étant supposé n'avoir aucune information sur la clé, lorsqu'il mène une DPA il doit calculer 64 traces différentielles, une pour chacune des 2^6 valeurs sur 6 bits de la sous-clé de tour. Un pic apparaîtra sur la courbe différentielle correspondant à la valeur correcte de la sous-clé de tour, à l'endroit où est manipulé le bit dont la valeur est corrélée à la fonction de sélection. Sur les 63 autres courbes différentielles obtenues pour des valeurs incorrectes de la sous-clé, la trace ne montrera qu'un léger bruit (on néglige ici les corrélations entre les différentes valeurs de sous-clé). En théorie, un quelconque des 4 bits de sortie des S-Box peut être analysé pour classifier les courbes de consommation de courant. Une trace différentielle obtenue en analysant les autres bits pourrait être calculée pour confirmer les résultats de la première. Plus de détails seront donnés à la Section 9.4. Le but de cette étude étant de tester l'efficacité des contre-mesures matérielles, nous avons exécuté une attaque DPA sur les clairs. Les attaquants ont plus probablement la connaissance du chiffré que celle du clair et entreprendraient donc une attaque sur le dernier tour du DES. Tous les résultats présentés dans ce chapitre sont valides également pour une attaque DPA sur les chiffrés.

Comme expliqué dans [KJJ99], la trace différentielle est calculée ainsi :

$$\Delta_D[j] = \frac{\sum_{i=1}^{N} D(P_i,K_s)T_i[j]}{\sum_{i=1}^{N} D(P_i,K_s)} - \frac{\sum_{i=1}^{N}(1-D(P_i,K_s))T_i[j]}{\sum_{i=1}^{N}(1-D(P_i,K_s))} = \varepsilon_1 - \varepsilon_0 \quad ,$$

où K_s sont les six bits de sous-clé inconnus, P_i le $i^{\text{ème}}$ clair connu, $D(P_i, K_s)$ la fonction de sélection, $T_i[j]$ le $j^{\text{ème}}$ échantillon de la courbe de consommation de courant, et $\Delta_D[j]$ le $j^{\text{ème}}$ élément de la trace différentielle.

Le nombre de courbes nécessaires pour mener l'attaque dépend fortement des conditions de mesure : plus le bruit est faible et moins de courbes sont nécessaires. Nous renvoyons le lecteur à [CKN01] pour diverses recommandations utiles. Pour que le pic puisse être identifié, on doit avoir

$$\varepsilon_1 - \varepsilon_0 > \sigma/\sqrt{N}, \tag{9.1}$$

où σ représente le bruit et N le nombre de courbes nécessaires.

Un meilleur équipement d'acquisition et une fréquence d'échantillonnage plus élevée entraînent un bruit plus faible. Bien que dépendant du composant, la Table 9.1 donne une idée grossière de la valeur requise pour N en fonction de la fréquence d'échantillonnage de l'équipement d'acquisition exprimée en MHz pour une carte travaillant à 3,68 MHz. La carte utilisée pour l'obtention de ces valeurs ne possédait pas de contre-mesure destinée à contrer les attaques DPA.

9.2.2 Interruptions aléatoires de processus

L'une des contre-mesures les plus usuelles contre la DPA est l'introduction d'*interruptions aléatoires de processus* (RPI). Plutôt que d'exécuter toutes les opérations de manière séquentielle, la CPU entrelace l'exécution du code avec celle d'instructions fictives de sorte que les cycles des opérations ne se superposent pas à cause de décalages temporels. Cela a pour effet d'étaler les pics de DPA sur la trace différentielle à cause d'un effet de désynchronisation connu dans le domaine du traitement numérique du signal sous le nom de *moyennage incohérent* [Lyo96]. Les décalages temporels peuvent être considérés comme du bruit superposé. Il va sans dire que les RPI ne rendent pas l'attaque théoriquement infaisable, mais accroissent considérablement le nombre N de courbes nécessaires.

En supposant que les RPI se produisent avec une probabilité constante p, même si un pic devait être visible sur la trace différentielle du fait que la valeur correcte de la sous-clé a été supposée, le pic pourrait rester brouillé dans le bruit car étalé sur des cycles consécutifs. À cause des RPI, le pic qui apparaît réellement suit une distribution gaussienne grossièrement caractérisée par une position moyenne μ et une variance v qui peuvent être calculées précisément.

Supposons que le pic sur la trace différentielle doivent nominalement apparaître après n cycles. En présence de RPI, un pic apparaîtra après $n + C_n$ cycles, où le délai $C_n = \sum_{i=1}^{n} c_i$, c_i étant le $i^{\text{ème}}$ cycle, avec $c_i = 1$ si un RPI a eu lieu et $c_i = 0$ sinon.

La position moyenne du pic est :

$$\mu = \langle C_n + n \rangle = \sum_{i=1}^{n} \langle c_i \rangle + n = np + n,$$

et la variance v est :

$$v = \langle C_n^2 \rangle - \langle C_n \rangle^2 = \sum_{i=1}^{n} \mathrm{Var}(c_i) = n(1-p)p \simeq np.$$

Nous pouvons donc estimer l'écart-type[42] $\delta \simeq \sqrt{np}$, ce qui signifie que (pour tous les usages expérimentaux) le pic sera étalé sur un intervalle $\pm\delta$ centré autour de μ. En d'autres termes, nous considérons le pic comme distribué sur $k = 2\delta \cong 2\sqrt{np}$ cycles consécutifs. Le pic sera donc visible si :

$$\frac{(\varepsilon_1 - \varepsilon_0)}{k} > \frac{\sigma}{\sqrt{N'}} \,. \tag{9.2}$$

Nous déduisons donc de (9.2) et (9.1) que le nombre de courbes de consommation protégées par RPI nécessaires pour restaurer le pic de DPA est :

$$N' = k^2 N.$$

Comme exemple caractéristique, si un pic de DPA devait être visible après $n = 1600$ cycles (ce qui peut être un cas typique pour un pic observé après le premier tour de DES) alors $p = 12\%$ conduit à $k \cong 28$. Cela signifie que le nombre de courbes de consommation protégées par RPI nécessaires pour rejouer la même attaque doit être multiplié par un facteur 784. Dans un but de recherche cette attaque a effectivement été menée avec succès, mais en conditions réelles une telle attaque est improbable à cause du nombre prohibitif de courbes qui devraient être recueillies. Dans la prochaine section, nous décrivons une méthode qui permet de réduire de manière conséquente le nombre de courbes nécessaires pour mener l'attaque.

[42] Intentionnellement nous utilisons δ à la place de la lettre σ que nous réservons pour un autre usage.

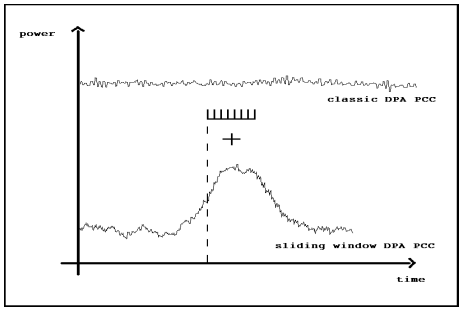


FIG. 9.1 – L'opération d'intégration.

9.3 Reconstruction de pic par intégration

L'amplitude du pic $(\varepsilon_1 - \varepsilon_0)$ peut être reconstruite de manière à diminuer le nombre de courbes de consommation de courant nécessaires. À cause de la désynchronisation, l'amplitude du pic est divisée par un facteur lié à k, mais l'amplitude d'origine peut être restaurée en intégrant sur k cycles consécutifs le signal protégé par RPI. La nouvelle valeur du signal est $k \times \frac{(\varepsilon_1 - \varepsilon_0)}{k}$, et la nouvelle valeur du bruit est $\frac{\sigma}{\sqrt{N''}} \times \sqrt{k}$. Le facteur $\sqrt{k}$ provient de ce que l'on a sommé k valeurs moyennes dont les réalisations du bruit sont supposées indépendantes. En conséquence le pic sera visible si :

$$\varepsilon_1 - \varepsilon_0 > \frac{\sigma}{\sqrt{N''}} \times \sqrt{k}.$$

Donc pour retrouver le même rapport signal sur bruit que dans (9.1), l'équation suivante doit être satisfaite :

$$N'' = kN.$$

Puisque cette méthode implique l'intégration des valeurs de consommation sur k cycles consécutifs, nous l'appelons DPA *par fenêtre glissante* (en anglais, *Sliding-Window* DPA, SW-DPA). Implémenter la SW-DPA se fait en deux étapes. Tout d'abord une courbe différentielle classique doit être obtenue. À moins qu'un très grand nombre de courbes soient utilisées, même pour l'hypothèse correcte aucun pic n'apparaîtra à cause des RPI. Pour que les pics apparaissent, les courbes protégées par RPI doivent être intégrées. Cette étape consiste à ajouter entre eux sur k cycles consécutifs les points de la courbe différentielle obtenue à la première étape. Pour visualiser cette opération, le lecteur peut imaginer un peigne à k dents, chacune correspondant à un point sur la courbe différentielle créée à la première étape (voir la Figure 9.1). La distance entre deux dents consécutives du peigne doit correspondre au nombre d'échantillons temporels séparant deux cycles consécutifs. L'intégration se fait en ajoutant la valeur de courant des points indiqués par le peigne.

Si on considère les mêmes chiffres que précédemment (c'est-à-dire $k = 28$) l'attaque peut être menée en multipliant par un facteur 28 le nombre de courbes nécessaires. Si une DPA peut être menée avec 120 courbes de consommation de courant non protégées acquises à 500 MHz, alors 3360 courbes protégées par RPI devraient être nécessaires.

La Figure 9.2 montre des courbes différentielles réelles obtenues avec une fenêtre d'intégration de 30 pour des hypothèses de clé correcte (courbe du haut) et incorrecte (courbe du bas).

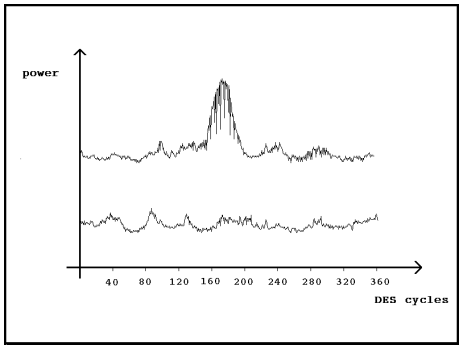


FIG. 9.2 – Traces différentielles pour des hypothèses de clé correcte et incorrecte.

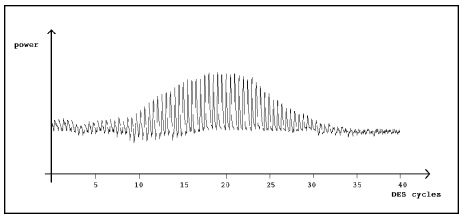


FIG. 9.3 – Vue élargie du pic restauré pour l'hypothèse correcte.

Précisons que tous les numéros de cycle sont reportés pour montrer au lecteur la largeur du pic de DPA. Ils ne constituent pas une référence absolue depuis le début de l'exécution du DES puisque ce nombre dépend fortement de la manière dont la fonction est implémentée.

La Figure 9.3 présente une vue élargie du pic de trace différentielle pour l'hypothèse correcte. On peut voir facilement que ce qui ressemble à un unique pic dans la vue large est constitué de différentes "portions de pic". C'est l'effet de l'intégration, qui ne somme les fractions de tous les pics distribués que lorsqu'ils sont précisément centrés.

9.4 La variante par intégration de Hamming

Lorsqu'on détermine la sous-clé par DPA classique, les courbes sont séparées en n'observant qu'un des quatre bits de sortie de la S-Box. Expérimentalement nous avons obtenu 4 traces différentielles par S-Box, chacune en examinant un bit différent de la sortie, et avons remarqué que certains bits laissent fuir plus d'information que d'autres[43]. Pour mener à bien une DPA, un attaquant pourrait prédéterminer quels bits conduisent au meilleur pic pour l'hypothèse correcte de clé. Notre approche a cependant été de tirer partie de l'information contenue dans les 4 bits de sortie de la S-Box simultanément.

Supposons que la consommation de courant d'un composant soit proportionnelle au *poids de Hamming* de la sortie. Si un seul bit de sortie de la S-Box est observé et si les courbes sont séparées selon la valeur de ce bit, l'amplitude du pic sera proportionnelle à :

$$\langle H \rangle_{s_i=1} - \langle H \rangle_{s_i=0} = (1 + \tfrac{3}{2}) - (0 + \tfrac{3}{2}) = 1,$$

[43] Cette observation est reprise est analysée plus en détail au Chapitre 3. Voir notamment la discussion relative à la Figure 3.3.

où H est la consommation de courant à un certain instant et s_i est la valeur du bit i de sortie de S-Box. En particulier, nous posons $\langle H \rangle_{s_i=1} = (1 + \frac{3}{2})$ car nous considérons les sorties de S-Box pour lesquelles un bit est certainement égal à 1 alors que les trois autres bits sont égaux à 1 avec probabilité $\frac{1}{2}$.

Le rapport signal sur bruit est égal à :

$$\text{SNR} = 1 \,/\, \left(\frac{\sigma}{\sqrt{N}}\right),$$

où σ est l'écart-type du bruit de la courbe différentielle et N le nombre de courbes considérées.

Si au lieu de cela les 4 bits de sortie de S-Box sont considérés simultanément, un nouveau critère de séparation des courbes doit être défini. Les courbes pourraient être classées selon le poids de Hamming total de la S-Box, c'est-à-dire les courbes pour lesquelles quatre ou trois 1 apparaissent pourraient être regroupées d'un côté tandis que les courbes pour lesquelles zéro ou un seul 1 apparaissent seraient regroupées d'un autre côté, les courbes dont la sortie de S-Box aurait deux 1 et deux 0 pouvant être ignorées. Dans ce cas, l'amplitude du pic serait proportionnelle à :

$$\langle H \rangle_{b=4,3} - \langle H \rangle_{b=0,1} = \frac{16}{5} - \frac{4}{5} = \frac{12}{5} = 2,4 \quad , \tag{9.3}$$

où b est le nombre de 1. En particulier, $\langle H \rangle_{b=4,3} = \frac{16}{5}$ est le poids de Hamming moyen lorsque trois ou quatre 1 apparaissent dans toutes les combinaisons de chaînes de quatre bits.

Nous appelons cette méthode la variante par intégration de Hamming. Celle-ci, combinée à une intégration dans le cas où des RPI ont été insérés, conduit à un pic bien plus élevé.

Le rapport signal sur bruit de cette méthode est égal à :

$$\text{SNR} = \frac{12/5}{\sigma/\sqrt{\frac{10}{16} \times N}} \approx 1,9 \,/\, \left(\frac{\sigma}{\sqrt{N}}\right) \quad ,$$

où σ est l'écart-type de la courbe différentielle et $10N/16$ est le nombre de courbes considérées. Dans le calcul ci-dessus nous tenons compte du fait que $6N/16$ courbes sont ignorées puisqu'une courbe n'est pas utilisée lorsque deux 1 (et deux 0) apparaissent dans la sortie de la S-Box.

Le rapport de 1,9 entre les rapports signal sur bruit des deux méthodes est un résultat théorique. Comme exposé précédemment, certains bits de sortie fuient plus d'information que d'autres et le rapport expérimental entre les pics observés dépend grandement du bit de sortie cible. Cela se voit clairement sur la Figure 9.4, où deux courbes différentielles obtenues par SW-DPA sont présentées pour différents bits de sortie de la S-Box.

9.5 Redéfinition de la fonction de sélection

Il existe une forte corrélation entre la consommation de courant d'un composant et l'opération exécutée. Cette valeur est particulièrement élevée pendant le transfert d'une donnée entre la CPU et la RAM externe, donc cette opération, exécutée après avoir déterminé la sortie d'une S-Box, est généralement ciblée pour la DPA. L'hypothèse faite pour créer et interpréter la courbe de trace différentielle est que la consommation de courant est différente selon que le bit de sortie de la S-Box vaut 0 ou 1. La consommation de courant, cependant, ne dépend pas *seulement* de la valeur de la sortie, mais des transitions qui interviennent sur le bus (voir [CKN01] par exemple). En supposant une implémentation ordinaire à base d'inverseurs CMOS, on peut s'attendre à une forte consommation de courant lorsque un 1 est écrit sur une ligne de bus préalablement

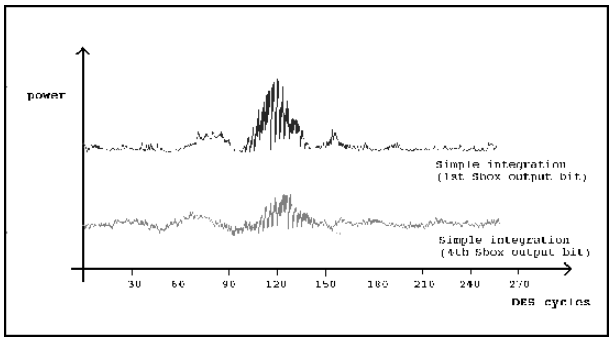


FIG. 9.4 – Traces différentielles obtenue sur la S-Box 1 en appliquant l'intégration de Hamming (SW-DPA) sur le bit 1 (courbe supérieure) ou sur le bit 4 (courbe inférieure).

TAB. 9.2 – Classification des courbes de consommation de courant selon la sortie de la S-Box.

Poids de Hamming élevé	Poids de Hamming faible
1111	0000
1110	0001
1101	0010
1011	0100
0111	1000

déchargée, ou lorsque un 0 est écrit sur une ligne de bus préalablement chargée. Les valeurs dans ces deux cas peuvent ne pas être les mêmes, mais comme la différence n'est pas essentielle pour notre étude, elle a été négligée dans ce qui suit.

La trace différentielle obtenue par la DPA classique montrera un pic pour l'hypothèse de clé correcte même si l'on ne tient pas compte de l'état du bus : les deux groupes de consommation de courant dans lesquels les courbes sont classées contiendront les mêmes éléments. Au pire une erreur sur le signe du pic sera commise, mais ceci n'a pas d'importance pour l'attaque. En revanche l'état du bus doit être pris en compte lorsqu'on observe simultanément les quatre bits de sortie, sans quoi de l'information pourrait être perdue.

Lorsque l'on applique la variante de Hamming, les consommations de courant de quatre lignes de bus sont analysées simultanément. Les valeurs 0 à 15, correspondant à toutes les combinaisons possibles de 0 et de 1 sur les quatre lignes de bus, peuvent avoir été préalablement déposées sur le bus. Afin de réduire le nombre de possibilités à étudier, seules les valeurs de 0 à 7 peuvent être considérées puisque dans notre modèle simplifié nous supposons que les consommations de courant dues aux transitions de 1 à 0 ou de 0 à 1 sont équivalentes.

Classifions les courbes selon les bits de sortie de la S-Box, en négligeant l'état préalable des lignes de bus. Il en résulte les deux groupes présentés sur la Table 9.2 correspondant aux poids de Hamming élevés (3 ou 4 bits valant 1) et aux poids de Hamming faibles (0 ou 1 bit valant 1). Pour l'hypothèse correcte nous nous attendons à un pic d'amplitude proportionnelle à 2,4 (9.3).

Les lignes de bus sur lesquelles une transition a eu lieu sont celles pour lesquelles $S_i \oplus B_i = 1$, où S_i est le $i^{\text{ème}}$ bit de sortie de S-Box et B_i est la $i^{\text{ème}}$ ligne de bus. En conséquence, pour regrouper correctement les courbes selon la consommation de courant, elles devraient être classées selon le nombre de 1 qui résultent de $S_i \oplus B_i$.

TAB. 9.3 – Classification des courbes de consommation de courant en négligeant la valeur préalable du bus.

Poids de Hamming élevé			Poids de Hamming faible		
S-Box	bus	S-Box $\oplus$ bus	S-Box	bus	S-Box $\oplus$ bus
1111	0011	1100	0000	0011	0011
1110	0011	1101	0001	0011	0010
1101	0011	1110	0010	0011	0001
1011	0011	1000	0100	0011	0111
0111	0011	0100	1000	0011	1011

Supposons que la valeur préalable sur le bus était 0011. La classification qui en résulte lorsque l'on néglige cette information est présentée sur la Table 9.3. D'après les colonnes 3 à 6, la hauteur du pic qui en résulte sera proportionnelle à :

$$< H >_{b=4,3} - < H >_{b=0,1} = \frac{10}{5} - \frac{10}{5} = 0,$$

donc toute l'information est perdue.

En revanche, dans le cas où la valeur préalablement présente sur le bus est 1000 ou toute autre configuration dans laquelle trois 1 ou trois 0 sont présents, la hauteur du pic serait proportionnelle à :

$$< H >_{b=4,3} - < H >_{b=0,1} = \frac{13}{5} - \frac{7}{5} = \frac{6}{5},$$

donc une partie, mais pas l'intégralité, de l'information utile est perdue.

Seulement dans le cas où la valeur sur le bus était précédemment 0000, ou de manière équivalente 1111, alors même en négligeant cette valeur la hauteur du pic serait proportionnelle à 2,4.

Si l'état préalable du bus est inconnu, et afin de trouver l'hypothèse de clé correcte en appliquant l'intégration de Hamming, l'attaque doit être répétée pour les 8 possibilités. Pour la clé correcte, on doit observer :

- une courbe différentielle avec un pic élevé pour la valeur préalable du bus correcte
- quatre courbes différentielles avec un pic moyen pour une erreur d'un bit (ou de trois bits) sur la valeur préalable du bus
- trois courbes différentielles plates pour une erreur de deux bits sur la valeur préalable du bus

Pour que cette attaque puisse être menée, l'état de la ligne de bus avant l'instruction ciblée par la DPA doit être constant sans quoi une classification correcte des courbes de courant ne peut pas être effectuée. Cette valeur est constante lorsque l'opération précédente concernant le bus est le chargement d'un code d'opération, ou lorsqu'une donnée constante mais indépendante du code, transite sur le bus.

9.6 Conclusion

Ce chapitre nous a montré que la DPA peut être menée y compris sur des composants implémentant des contre-mesures au niveau matériel destinées à y résister. La première méthode

proposée consiste à appliquer une fenêtre glissante à la DPA classique décrite dans [KJJ99]. La perte de synchronisation causée par les RPI et l'augmentation qui en résulte du nombre de courbes nécessaires pour mener l'attaque sont calculées. La variante par intégration de Hamming est légèrement plus compliquée car, puisque 4 bits de sortie sont considérés simultanément, la valeur préalable des lignes de bus correspondantes doit être déterminée. Comme cette information est généralement inconnue de l'attaquant, toutes les possibilités doivent être examinées expérimentalement. L'avantage de cette variante est un meilleur rapport signal sur bruit, ou un succès de l'attaque avec un nombre de courbes moindre. Puisque la deuxième méthode implique un coût calculatoire plus élevé, elle pourrait ne s'appliquer que sur un nombre restreint de clés secrètes probables pour lesquelles la première méthode laisse place au doute.

Pour être sûrs, les dispositifs cryptographiques devraient incorporer conjointement des contre-mesures matérielles et logicielles pour diminuer la faisabilité des attaques par canaux auxiliaires. Il est également important de prouver la validité des contre-mesures implémentées car des hypothèses heuristiques peuvent s'avérer insuffisantes.

Chapitre 10

Attaques par analyse de fautes d'algorithmes résistants à la DPA

Sommaire

Ce chapitre présente différentes attaques qui permettent de retrouver de l'information à l'aide de fautes injectées au début d'algorithmes cryptographiques dont les implémentations utilisent un masquage booléen dans le but de se protéger contre l'*analyse différentielle de courant* (DPA). Ces attaques visent les fonctions d'initialisation qui permettent à l'algorithme d'être protégé. Nous présentons également certaines mises en œuvre concrètes de ces attaques qui sont décrites plus en détail dans la thèse de Michael Tunstall [Tun06].

Ce travail a fait l'objet d'une publication à Fdtc '06 avec Frédéric Amiel et Michael Tunstall [ACT06]. Nous avons corrigé quelques chiffres erronés de l'article, notamment ceux, à la Section 10.6, concernant le nombre moyen d'entrées de S-Box utilisées compatibles avec plusieurs messages.

10.1 Introduction

L'utilisation des collisions pour trouver et exploiter une faute au début d'un algorithme est apparue dans plusieurs articles. Dans [Hem04], Ludger Hemme décrit une méthode d'exploitation de fautes dans les premiers tours d'une implémentation de DES. Il y détaille une attaque complexe dans laquelle des fautes sont injectées dans les premiers tours du DES et exploitées en trouvant un message qui donne naturellement (c'est-à-dire sans faute) le même chiffré. Cette information est alors utilisée pour en dériver de l'information sur la clé.

Un cas trivial de ce type d'attaque est donné par Johannes Blömer et Jean-Pierre Seifert dans [BS03] où un bit connu du premier XOR de l'AES est supposé pouvoir être forcé à zéro. Si le chiffré change, alors ce bit valait 1 ; si le chiffré reste le même alors ce bit valait 0. Cela peut casser une implémentation de l'AES avec seulement 128 exécutions à injection de faute réussie. Cependant, modifier des bits de cette manière requiert une trop grande précision dans la pratique.

De manière générale, les attaques qui consistent à exploiter l'observation qu'un chiffré fauté est identique à un chiffré obtenu (éventuellement pour une valeur différente du message) sans faire de faute, sont appelées *analyses de fautes par collision* (CFA). Nous allons présenter certaines variantes de ce type d'attaques qui peuvent être implémentées contre l'AES sur des dispositifs embarqués. Nous décrirons une version au niveau octet de l'attaque présentée dans [BS03]. Nous détaillerons également plusieurs autres attaques, plus complexes, qui sont réalisables en présence de contre-mesures contre la DPA. Des implémentations de certaines de ces attaques, conduites dans des conditions contrôlées, seront présentées. Nous décrirons également un autre type de CFA qui utilise le DES comme exemple, et où les fautes s'appliquent à l'initialisation des S-Box randomisées. On montrera que la version la plus simple de cette attaque s'obtient en combinant la modification de valeurs de S-Box avec une *analyse différentielle de fautes* (DFA) [BS97]. Là encore, nous présenterons des mises en œuvre pratiques de ces attaques qui utilisent des défaillances subites de la tension appliquée ou de l'horloge.

Ce chapitre est organisé de la manière suivante. La Section 10.2 présente une variante au niveau octet de l'attaque décrite dans [BS03], ainsi que les modifications qui doivent être apportées à cette attaque. La Section 10.3 rappelle la méthode la plus connue pour protéger les algorithmes contre les attaques différentielles comme la DPA. La suite de ce chapitre supposera l'utilisation de cette contre-mesure, et proposera successivement plusieurs attaques réalisables sur des implémentations protégées de cette manière. Une adaptation de l'attaque présentée à la Section 10.2 sera proposée à la Section 10.4. Elle décrit comment adapter cette CFA sur le premier XOR de l'AES dans le cas où l'algorithme est protégé par masquage booléen. La Section 10.5 présente une attaque contre le masquage de la clé effectué pour l'initialiser. Les deux sections suivantes décrivent des attaques permettant d'exploiter des fautes injectées durant l'initialisation des boîtes de substitution randomisées. La Section 10.6 en présente une première version dans laquelle l'attaquant est supposé connaître les valeurs des S-Box qu'il modifie, et propose une contre-mesure. La Section 10.7 présente une manière différente d'exploiter ces fautes qui permet une attaque efficace même en présence de cette contre-mesure. De nouvelles contre-mesures, plus abouties, sont alors données à la Section 10.8, juste avant la conclusion.

10.2 Une analyse de fautes par collisions sur l'AES

Une attaque simple de type CFA est une implémentation au niveau octet de celle décrite par Johannes Blömer et Jean-Pierre Seifert [BS03]. Dans cette variante, nous supposons qu'une faute permet de forcer à 0 le résultat d'une instruction XOR. Si une telle faute est injectée sur un des XOR de la fonction `AddRoundKey` juste avant le premier tour de l'AES, alors le chiffré fauté obtenu correspond à une valeur intermédiaire nulle au niveau de ce XOR. L'attaquant peut essayer de modifier l'octet de message impliqué dans cette instruction jusqu'à obtenir une exécution pour laquelle la valeur naturelle de sortie du XOR est égale à 0. Une collision est alors observée entre le chiffré sans faute ainsi obtenu et le chiffré fauté. Cette collision permet d'établir que la valeur m^* de l'octet de message provoquant la collision vérifie l'équation $m^* \oplus k = 0$, dont on peut déduire la valeur de l'octet de clé.

TAB. 10.1 – L'attaque de BIHAM et SHAMIR appliquée au début de l'AES.

Entrée	Clé		Sortie
$M \rightarrow$	$K_0 =$	XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX	$\rightarrow C_0$
$M \rightarrow$	$K_1 =$	XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX 00	$\rightarrow C_1$
$M \rightarrow$	$K_2 =$	XX XX XX XX XX XX XX XX XX XX XX XX XX XX 00 00	$\rightarrow C_2$
$M \rightarrow$	$K_3 =$	XX XX XX XX XX XX XX XX XX XX XX XX XX 00 00 00	$\rightarrow C_3$
$\vdots$	$\vdots$		$\vdots$
$M \rightarrow$	$K_{14} =$	XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$\rightarrow C_{14}$
$M \rightarrow$	$K_{15} =$	XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$\rightarrow C_{15}$

Cette procédure nécessite la génération de 16 chiffrés fautés, ainsi que 256 chiffrements avec le dispositif cible pour retrouver chaque octet de clé, soit un effort de recherche total de 2^{12} requêtes de chiffrement.

Dans [NNTW05], DAVID NACCACHE, PHONG NGUYEN, MICHAEL TUNSTALL et CLAIRE WHELAN mentionnent que des fautes peuvent permettent de terminer une boucle **for** avant qu'elle n'ait fini toutes ses itérations. Si la mémoire où est stocké le résultat du XOR entre le message et la clé n'a pas été utilisée auparavant, il y a de grandes chances qu'elle valle `00` ou `FF` en fonction de la représentation logique d'un état physique. En appliquant cette idée au XOR avec la clé, une attaque peut être implémentée en laissant inchangé un octet de la zone mémoire et en générant le chiffré correspondant. Puis deux octets, etc... comme décrit dans la Table 10.1 et originellement proposé par ELI BIHAM et ADI SHAMIR dans [BS97].

Cela peut être réalisé avec 15 fautes successives. Le premier octet de la clé peut alors être trouvé en cherchant parmi les 2^9 valeurs possibles de la clé qui peuvent produire C_{15}. Ces 2^9 valeurs correspondent aux 2^8 valeurs possibles pour `XX`, et à 2 possibilités pour le reste des bits en fonction du codage physique d'un 1 logique. Une fois que le premier octet de clé est connu, le deuxième octet peut être trouvé avec 2^8 chiffrements AES supplémentaires en utilisant C_{14}. On peut continuer ainsi avec 2^8 chiffrements pour chacun des octets suivants, soit un total d'un peu plus de 2^{12} requêtes de chiffrement AES pour retrouver entièrement la clé.

L'attaque basique (contenu mémoire mis à 0) a été implémentée sur un micro-processeur à 8 bits pour cartes à puce, en utilisant la défaillance subite comme injecteur de faute, et en balayant l'intégralité de la boucle qui copie la clé d'AES de la mémoire non volatile vers une zone temporaire de travail en RAM au début de l'exécution de l'AES (juste avant la fonction `AddRoundKey`). Cela a produit 127 chiffrés fautés différents (alors que 15 seulement étaient attendus), donnant comme résultat les 22 clés possibles listées ci-dessous en base 16 en recherchant parmi les résultats de manière orientée octet :

```
00000000000000000000000000000000
FE000000000000000000000000000000
FEDC0000000000000000000000000000
FEDCBA0000000000000000000000000000
FEDCBA9800000000000000000000000000
FEDCBA9876000000000000000000000000
FEDCBA9876540000000000000000000000
FEDCBA9876543200000000000000000000
FEDCBA9876543210000000000000000000
```

```
FEDCBA98765432100100000000000000
FEDCBA98765432100123000000000000
FEDCBA98765432100123450000000000
FEDCBA98765432100123456700000000
FEDCBA98765432100123456789000000
FEDCBA98765432100123456789ABCDEF
FEDCBA98765432100123456789ABCD00
FEDCBA98765432100123456789ABCDCD
FEDCBA98765432100123456789AB0000
FEDCBA98765432100123456789AB00EF
FEDCBA98765432100123456789AB00AB
FEDCBA98765432100123456789ABEF00
FEDCBA98765432100123456789AB5300
```

À cause d'effets de désynchronisation et du balayage exhaustif de la boucle de copie, des chiffrés fautés non attendus ont été produits. Dans les clés possibles, on peut remarquer qu'il y a plusieurs valeurs possibles pour les deux derniers octets, ce qui rend l'attaque légèrement plus compliquée qu'initialement prévu tout en restant réalisable. La valeur correcte de la clé utilisée pour ces expériences était :

```
FEDCBA98765432100123456789ABCDEF
```

.

10.3 Implémentations sûres d'algorithmes

Les implémentations sont rendues sûres contre la DPA [KJJ99] et certaines attaques similaires en masquant la donnée manipulée avec une valeur aléatoire. D'un bout à l'autre de l'exécution, les valeurs des données manipulées présentes en mémoire sont toujours masquées avec le même aléa. Mehdi-Laurent Akkar et Christophe Giraud [AG01] ont proposé un exemple de ce type d'implémentation sur la base d'idées proposées par Suresh Chari, Charanjit Jutla, Josyula Rao et Pankaj Rohatgi dans [CJRR99].

La taille de l'aléa est généralement limitée car, avant chaque exécution, les S-Box nécessitent d'être modifiées avec cet aléa de sorte que les entrées et les sorties de la S-Box ne fuient aucune information. Cela est réalisé grâce à un algorithme tel que celui de la Figure 10.1, où la notation $(\cdot)_x$ représente des valeurs de l'ensemble $\{0, \ldots, x-1\}$, c'est-à-dire $(s_0, s_1, s_2, \ldots, s_{n-1})_x$ contient S découpée en mots appartenant à $\{0, \ldots, x-1\}$.

Comme on le remarque, l'aléa utilisé pour masquer la donnée d'entrée ne peut pas être plus grand que n, et celui utilisé pour la valeur de sortie ne peut dépasser x. Dans le cas de l'AES, $\mathbf{R}$ and r seront tous deux sur un octet, donc le masque aléatoire utilisé pendant le calcul sera probablement un octet. Cela est plus problématique pour le DES car les entrées et les sorties sont de tailles différentes, et la permutation de bits ajoute à la complexité, mais le principe reste le même.

10.4 Attaquer le premier XOR

Nous décrivons dans cette section une attaque utilisant les idées développées à la Section 10.2, mais adaptées au cas d'une implémentation protégée contre la DPA de la manière décrite ci-dessus. Au début de l'AES, la fonction `AddRoundKey` effectue un XOR entre le message et la clé

Algorithme 10.1 Randomisation des valeurs des S-Box

Entrée : $S = (s_0, s_1, s_2, \ldots, s_{n-1})_x$ contenant la S-Box,
un aléa d'entrée $\mathbf{R} \in \{0, \ldots, n-1\}$, et un aléa de sortie $r \in \{0, \ldots, x-1\}$

Sortie : $\tilde{S} = (\tilde{s}_0, \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_{n-1})_x$ contenant la S-Box randomisée

1. **pour** $i \leftarrow 0$ à $n-1$
2. $\quad \tilde{s}_{(i \oplus \mathbf{R})} \leftarrow s_i \oplus r$
3. **fin pour**

FIG. 10.1 – Randomisation des valeurs des S-Box.

Algorithme 10.2 Le premier XOR

Entrée : $M = (m_1, m_2, m_3, \ldots, m_{16})_{256}$ contenant le message,
$\tilde{K} = (\tilde{k}_1, \tilde{k}_2, \tilde{k}_3, \ldots, \tilde{k}_{16})_{256}$ contenant la clé masquée avec l'octet d'aléa $\mathbf{R}$

Sortie : $\widetilde{KM} = (\widetilde{km}_1, \widetilde{km}_2, \widetilde{km}_3, \ldots, \widetilde{km}_{16})_{256}$ le XOR entre le message et la clé, également masqué avec $\mathbf{R}$

1. **pour** $i \leftarrow 1$ à 16
2. $\quad \widetilde{km}_i \leftarrow m_i \oplus \tilde{k}_i$
3. **fin pour**

FIG. 10.2 – Le premier XOR.

avant la fonction `SubBytes` du premier tour. Tous les octets de la clé sont alors masqués par la valeur d'aléa $\mathbf{R}$, et le XOR se fait conformément à l'algorithme de la Figure 10.2.

Comme dans la Section 10.2, on utilise des fautes permettant d'interrompre une boucle **for** avant sa fin normale. Si l'algorithme de la Figure 10.2 est ainsi attaqué, de sorte que la boucle ne réalise que 14 itérations au lieu de 16, alors deux octets ne seront pas écrits dans $\widetilde{KM}$ et resteront donc inchangés. Cela signifie que ces octets vaudront physiquement 0, mais cela correspondra à la valeur logique $\mathbf{R}$ à cause du masquage.

Le chiffré obtenu étant le même que celui correspondant à des valeurs logiques de $m_{15} \oplus k_{15}$ et de $m_{16} \oplus k_{16}$ toutes deux égales à $\mathbf{R}$, le but de l'attaquant sera de rechercher un message identique au message de référence sur les 14 premiers octets, et pour lequel il fera varier les deux derniers octets jusqu'à obtenir deux valeurs m^*_{15} et m^*_{16} telles que

$$\begin{cases} m^*_{15} \oplus k_{15} = \mathbf{R} \\ m^*_{16} \oplus k_{16} = \mathbf{R} \end{cases} . \tag{10.1}$$

Le chiffré obtenu avec un tel message collisionnera avec le chiffré fauté, et l'attaquant saura que l'Equation (10.1) est vérifiée. Il ne connaît pas la valeur de $\mathbf{R}$, mais pourra néanmoins déduire que

$$k_{15} \oplus k_{16} = m^*_{15} \oplus m^*_{16} .$$

La taille de l'espace des clés est réduite de 2^{128} à 2^{120}. On continue en répétant l'attaque avec trois octets de clé laissés inchangés par l'algorithme de la Figure 10.2. Le chiffré obtenu correspond à des valeurs logiques de $m_{14} \oplus k_{14}$, $m_{15} \oplus k_{15}$ et de $m_{16} \oplus k_{16}$ toutes trois égales à une valeur $\mathbf{R}$ inconnue. La recherche exhaustive consiste à trouver $(m_{14}^*, m_{15}^*, m_{16}^*)$ vérifiant

$$\begin{cases} m_{14}^* \oplus k_{14} = \mathbf{R} \\ m_{15}^* \oplus k_{15} = \mathbf{R} \\ m_{16}^* \oplus k_{16} = \mathbf{R} \end{cases}.$$

Connaissant la valeur $\delta_{15,16} = k_{15} \oplus k_{16}$, l'attaquant peut choisir de ne tester que les triplets de la forme $(m_{14}^*, m_{15}^*, m_{15}^* \oplus \delta_{15,16})$ de sorte que les deuxième et troisième conditions soient équivalentes. Une collision apparaîtra dès lors que les deux premières conditions seront vérifiées. On en dérive donc $k_{14} \oplus k_{15}$ avec le même effort que pour $k_{15} \oplus k_{16}$.

Pour chaque faute injectée, l'attaquant a besoin de rechercher parmi 2^{16} valeurs pour déterminer les valeurs d'octets de message qui permettent de dériver de l'information sur la clé. Cela a besoin d'être réalisé en sollicitant la carte à puce, ce qui peut être long.

Dans les algorithmes résistants à la DPA, il est courant de vouloir effectuer le plus d'opérations possibles dans un ordre aléatoire. Cela constitue une contre-mesure additionnelle à celle présentée à la Section 10.3. Dans ce cas, la boucle de l'algorithme de la Figure 10.2 calcule les différents octets de XOR entre le message et la clé dans un ordre choisi aléatoirement parmi 16 ! façons possibles. D'une exécution à l'autre une donnée apparaîtra temporellement en différents points, ce qui est à la base une aide dans la protection contre la DPA.

Si un ordre aléatoire est implémenté et que les deux dernières itérations de la boucle ne sont pas exécutées, les deux octets qui ne seront pas calculés se trouveront à des positions aléatoires dans la zone mémoire de travail. Il existe $\binom{16}{2} = 120$ couples de positions possibles pour ces deux valeurs non calculées. Pour trouver une collision, l'attaquant devra avoir préalablement constitué un dictionnaire contenant les chiffrés de toutes les variations d'un message de référence dans lesquelles deux octets peuvent varier. Ce dictionnaire aura approximativement une taille de $120 \times 2^{16} \approx 2^{23}$ entrées. Lorsque l'attaquant acquiert un chiffré fauté, celui-ci correspond à des valeurs logiques de sortie du XOR égales, à $k \oplus m$ pour tous les octets qui sont passés dans l'une des 14 premières itérations, et à une même valeur $\mathbf{R}$ inconnue pour deux positions aléatoires d'octets i et j.

Ce chiffré coïncide avec celui des chiffrés du dictionnaire correspondant à la variation de m_i et m_j telle que

$$\begin{cases} m_i^* \oplus k_i = \mathbf{R} \\ m_j^* \oplus k_j = \mathbf{R} \end{cases}.$$

On en déduit ici aussi que $k_i \oplus k_j = m_i^* \oplus m_j^*$.

Ce processus peut ensuite être répété jusqu'à ce que de l'information soit dérivée sur chaque octet de la clé. Tout octet de clé est alors déterminé de manière relative à chacun des autres, laissant une recherche exhaustive de 2^8 pour trouver la valeur absolue correcte.

Afin d'être capable de trouver des collisions, l'attaquant doit générer un dictionnaire d'environ 2^{23} entrées. Ces valeurs dépendent de la clé donc elles doivent être générées à l'aide du dispositif attaqué, ce qui peut être vraiment très long pour des dispositifs comme des cartes à puce qui utilisent des protocoles de communication relativement lents. Pour donner une idée du temps requis pour créer un dictionnaire, une carte à puce avec un AES résistant à la DPA a été utilisée. Il lui fallait environ 149 millisecondes pour créer chaque entrée du dictionnaire. Il aurait donc fallu environ 14,5 jours pour créer le dictionnaire entier. Remarquons néanmoins, qu'une

fois créé, ce dictionnaire peut être utilisé à volonté. Ceci peut constituer un avantage par rapport à l'attaque de la version en ordre non aléatoire. En effet, dans ce dernier cas, une recherche exhaustive en 2^{16} exécutions (environ 3 heures d'utilisation de la carte) doit être menée pour chaque faute injectée. S'il s'avérait que l'effet de la faute n'est pas toujours celui que l'on espère, alors l'attaque sur la version en ordre fixe deviendrait paradoxalement plus longue que celle en ordre aléatoire.

Les conditions d'implémentation de l'attaque sont les mêmes que celles décrites à la Section 10.2 à deux différences près. L'implémentation utilisée était résistante à la DPA et l'instant d'injection des défaillances subites était fixe puisque l'ordre aléatoire fournissait la variation temporelle. La clé était connue *a priori*, si bien que le dictionnaire a pu être généré avec un ordinateur plutôt qu'avec la carte.

En conduisant des fautes qui utilisent des défaillances subites sur le Vcc pendant moins d'une heure, 118 chiffrés fautés ont été obtenus. Parmi eux, 72 collisions uniques ont été extraites. L'information dans ces 72 collisions a été traitée pour trouver 31 clés uniques, c'est-à-dire sans relation linéaire entre aucune de ces 31 clés. L'exploitation a duré environ 10 minutes sur un ordinateur standard.

On n'attendait qu'un seul candidat clé de 16 octets, mais 31 ont été produits montrant que certaines injections de fautes ont produit des collisions non reliées aux valeurs de la clé. Cependant, le nombre de clés produites peut facilement être testé pour déterminer la clé correcte, requérant $31 \times 2^8 \approx 2^{12}$ exécutions d'AES.

Comme mentionné précédemment, générer 2^{23} chiffrés avec une carte à puce demande un temps très long. Cela peut être réduit en ne créant qu'une partie du dictionnaire, et en générant plus de chiffrés fautés pour avoir une bonne probabilité de pouvoir dériver la clé à l'aide de l'un d'entre eux.

Par exemple, si on générait un dictionnaire de taille 2^{19}, ce qui demanderait environ 21 heures avec la carte, des chiffrés fautés pourraient être générés jusqu'à ce qu'une collision soit trouvée. On peut espérer qu'un chiffré fauté sur 2^4 soit présent dans le dictionnaire. Un attaquant aurait alors besoin de 16 fois plus de fautes en comparaison avec l'attaque ci-dessus, mais comme seulement quelques chiffrés fautés sont nécessaires pour réaliser l'attaque, c'est une option efficace.

10.5 Attaquer le masquage de la clé

Avant qu'une clé puisse être utilisée par l'algorithme de la manière décrite à la Section 10.3, la valeur aléatoire, $\mathbf{R}$, qui est appliquée aux S-Box doit être appliquée à la clé. Les valeurs de clés sont généralement stockées en mémoire non volatile également masquées, mais cette fois avec une valeur aléatoire de la même taille que la clé. Cela est rendu possible par le fait que cet aléa est une valeur statique pour chaque carte puisqu'il est stocké en EEPROM et diversifié d'une carte à une autre. Juste avant l'utilisation de la clé dans l'algorithme, ce masque doit être enlevé et remplacé avec $\mathbf{R}$ sans que la valeur en clair de la clé ne soit manipulée. Ce processus est montré dans l'algorithme de la Figure 10.3. Ici encore, cela se fait en ordre aléatoire plutôt que comme c'est décrit.

Une attaque similaire à celle de la Section 10.4 peut être envisagée contre la boucle qui réalise ce masquage. Si un seul octet est initialisé par cette boucle, la mémoire résultante sera physiquement composée de 15 zéros et d'un octet valant $k_i \oplus \mathbf{R}$, pour des valeurs aléatoires de i et de $\mathbf{R}$. L'interprétation logique sera donc constituée de 15 octets égaux à l'aléa $\mathbf{R}$, et de l'octet k_i. Un dictionnaire contenant les chiffrés correspondant aux environ 2^{20} clés formées de 15

Algorithme 10.3 Masquage de la clé

Entrée : $KR = (kr_1, kr_2, kr_3, \ldots, kr_{16})_{256}$ la clé masquée telle qu'en mémoire non volatile,
$R = (r_1, r_2, r_3, \ldots, r_{16})_{256}$ le masque aléatoire de pleine taille,
R un octet aléatoire

Sortie : $\tilde{K} = (\tilde{k}_1, \tilde{k}_2, \tilde{k}_3, \ldots, \tilde{k}_{16})_{256}$ la clé masquée avec **R**

1. **pour** $i \leftarrow 1$ à 16
2. $\quad \tilde{k}_i \leftarrow kr_i \oplus \mathbf{R}$
3. $\quad \tilde{k}_i \leftarrow \tilde{k}_i \oplus r_i$
4. **fin pour**

FIG. 10.3 – Masquage de la clé.

valeurs identiques et d'une valeur non contrainte pouvant se trouver à n'importe quelle position devra nécessairement contenir le chiffré fauté. Puisque les valeurs logiques de la clé utilisée lors d'une faute ne dépendent essentiellement pas de la valeur de la vraie clé, ce dictionnaire peut être généré "off-line" sur un ordinateur.

Cette attaque est ensuite répétée jusqu'à ce que suffisamment d'information sur la clé soit retrouvée pour permettre une recherche exhaustive. L'espérance du nombre de chiffrés fautés nécessaires pour retrouver chaque octet de cette manière peut être calculée à l'aide du test du collectionneur de coupons donné dans [Knu81]. Dans le cas de l'AES, cela requerrait environ 54 chiffrés en moyenne pour déduire la clé entière.

Les conditions d'implémentation de l'attaque sont les mêmes que celles décrites à la Section 10.2. La même implémentation logicielle a été exécutée sur la même carte à puce.

Le précalcul du dictionnaire a été généré en quelques minutes sur un ordinateur standard. Contrairement à l'attaque décrite à la Section 10.2, le dictionnaire ne dépend pas de la valeur de la clé utilisée, donc le dictionnaire généré est valide pour toute valeur de clé secrète.

Après avoir attaqué l'implémentation pendant approximativement une heure, environ 60 collisions ont été générées à partir de chiffrés fautés. Après avoir acquis ces chiffrés, le traitement *a posteriori* était trivial car aucune hypothèse incorrecte n'a été produite par les collisions trouvées.

10.6 Modifier des valeurs connues des S-Box

Nous présentons maintenant dans cette section et la suivante un type d'attaque différent de celles présentées précédemment. Le modèle de faute ne suppose pas ici qu'il est possible d'interrompre l'exécution d'une boucle, mais simplement de changer aléatoirement la valeur d'une donnée lors d'une lecture ou écriture en mémoire RAM par exemple.

Si les valeurs des S-Box sont créées comme indiqué à l'algorithme de la Figure 10.1, l'ordre dans lequel la S-Box est construite est alors connu (puisque i est incrémenté de 0 à 63). Une attaque par fautes peut alors être conçue sur la base d'une modification de valeurs connues de S-Box.

La première valeur de la première S-Box est modifiée par une faute et l'algorithme est exécuté avec un message pour lequel on connaît le chiffré. Si le chiffré fauté n'est pas égal au chiffré connu,

alors cette valeur de S-Box a été utilisée par l'algorithme ; s'il reste identique alors la valeur de S-Box n'a été utilisée à aucun endroit de l'algorithme. Les 64 valeurs de la première S-Box peuvent être successivement soumises à une faute de la sorte, et l'algorithme exécuté. À la suite de cela, on a identifié toutes les entrées de la première S-Box utilisées dans l'algorithme pour ce message.

Pour chaque S-Box, l'espérance du nombre d'entrées utilisées lors de chaque exécution du DES peut être calculée en utilisant la solution du problème classique du remplissage [MOV97] et vaut :

$$\sum_{t=1}^{16} t \left\{ {64 \atop t} \right\} \frac{64!}{(64-t)! \cdot 64^{16}} = 14,255 \ ,$$

où $\left\{ {64 \atop t} \right\}$ représente le nombre de Stirling de seconde espèce.

Ainsi, si l'attaque est répétée pour chaque S-Box, une liste d'environ 14 valeurs différentes sera produite pour les entrées utilisées dans chacune d'entre elles. Les valeurs de ces listes sont des candidats possibles pour les entrées de chaque S-Box au premier tour. Chacune de ces entrées peut être convertie en supposition sur la valeur de la sous-clé de la S-Box, simplement en calculant le XOR avec les bits correspondants du message. Cela produit $2^{30,7}$ suppositions pour la clé du 1$^{\text{er}}$ tour, menant à une recherche exhaustive totale de $2^{38,7}$ pour retrouver la clé entière.

Afin de réduire la taille de la recherche exhaustive, l'attaque peut être répétée avec un message différent. La clé du 1$^{\text{er}}$ tour appartient alors à l'intersection des deux espaces de clé. Tenant compte du fait que la valeur correcte appartient nécessairement aux deux ensembles, cela fournit $1 + 13,255 \times (13,255/63) = 3,79$ hypothèses différentes par S-Box, soit $2^{15,4}$ hypothèses pour la clé du tour 1, et une recherche exhaustive totale de $2^{23,4}$ clés.

Dans la pratique, il est improbable que les implémentations embarquées du DES aient leurs 512 valeurs de S-Box nécessaires pour le DES écrites séparément en mémoire. Elles sont généralement compressées pour optimiser la quantité de mémoire requise par l'implémentation de l'algorithme.

Une manière de faire est de stocker les données sur 256 octets où les S-Box impaires sont stockées sur les quartets hauts et les S-Box paires sur les quartets bas. Cela correspond à l'implémentation d'attaque détaillée ci-dessous, et toute la discussion supposera maintenant ce cas là. Il existe d'autres manières de compresser les S-Box, mais ce n'est pas considéré comme quelque chose que l'attaquant doit connaître avant de mener une attaque, car toutes les manières possibles peuvent être essayées jusqu'à ce que la bonne soit trouvée.

L'espérance du nombre d'hypothèses de clé générées en implémentant cette attaque contre un DES utilisant des S-Box compressées est donnée dans la Table 10.2 pour différents nombres de messages utilisés. Considérant que le nombre de fautes nécessaires pour mener l'attaque avec deux messages différents dans le cas des S-Box compressées est le même que celui nécessaire pour un seul message dans le cas de S-Box non compressées, on peut remarquer que, à nombre de fautes équivalent, l'attaque est plus efficace sur des S-Box compressées que sur des S-Box normales. Par exemple, en exploitant deux messages, une recherche exhaustive totale parmi $2^{35,4}$ clés est nécessaire au lieu des $2^{38,7}$ pour le cas des S-Box non compressées.

En essayant d'implémenter les attaques décrites dans [BS97], nous avons observé que lorsque le rapport cyclique[44] de l'horloge fournie à la carte à puce était trop petit un chiffré incorrect était produit. C'était sur un composant différent de celui utilisé dans les attaques précédentes

[44] Le rapport cyclique de l'horloge est la durée pendant laquelle une tension est appliquée sur la broche d'horloge comparé au temps où aucune tension ne lui est appliquée. Par exemple, une horloge standard a un rapport cyclique égal à 50%.

TAB. 10.2 – Nombre d'hypothèses générées en attaquant des S-Box compressées. Entre parenthèses, l'optimisation ignorant les modifications dans les deux derniers tours.

Messages	Hypothèses par paire de S-Box		Hypothèses pour la clé du tour 1		Espace de clé entier	
1	25,3	(22,8)	$2^{37,3}$	$(2^{36,1})$	$2^{45,3}$	$(2^{44,1})$
2	10,8	(8,99)	$2^{27,4}$	$(2^{25,3})$	$2^{35,4}$	$(2^{33,3})$
3	5,29	(4,34)	$2^{19,2}$	$(2^{16,9})$	$2^{27,2}$	$(2^{24,9})$
4	3,23	(2,77)	$2^{13,5}$	$(2^{11,8})$	$2^{21,5}$	$(2^{19,8})$

sur les fonctions d'initialisation de l'algorithme. Des études plus approfondies ont montré que si le rapport cyclique était inférieur à 15%, alors les valeurs des données écrites dans certaines zones de la mémoire du composant étaient incorrectes.

Cette attaque peut donc être implémentée contre des cartes à puce en exploitant cet effet. Comme mentionné ci-dessus, on doit prendre en compte que les S-Box sont écrites dans un format compressé pour économiser de la mémoire. L'attaque a été menée avec trois messages différents, suivie d'une petite recherche exhaustive pour trouver la clé. L'attaque entière à duré 45 minutes en utilisant des outils spécialement créés à cet effet.

Il est possible d'optimiser cette attaque en analysant les chiffrés fautés générés par les S-Box modifiées. À partir du chiffré, il est facile d'identifier qu'une valeur modifiée de S-Box n'a été utilisée que dans le quinzième ou dans le seizième tour. Puisque ce qui nous intéresse est de savoir si l'entrée modifiée de la S-Box a été, ou non, utilisée dans le premier tour, ces chiffrés peuvent être considérés comme équivalents à une valeur de S-Box non utilisée. Les listes se réduisent alors à l'ensemble des entrées de S-Box (ou de paires de S-Box dans le cas compressé) utilisées dans l'un des 14 premiers tours seulement. Les données entre parenthèses de la Table 10.2 présentent les chiffres correspondant à cette optimisation dans le cas de S-Box compressées.

Une contre-mesure pour cette attaque est de rendre aléatoire l'ordre dans lequel les S-Box sont modifiées. Cela peut s'appliquer à l'ordre dans lequel les S-Box sont traitées, ainsi qu'à l'ordre dans lequel les éléments de la S-Box sont masqués. Dans ce dernier cas, le masquage des données d'une S-Box peut alors être réalisé comme montré à l'algorithme de la Figure 10.4, ce qui n'ajoute aucune pénalité dans le temps d'exécution de l'algorithme. Le compteur i est XOR-é avec l'aléa d'entrée avant d'être utilisé, de sorte que l'ordre dans lequel les éléments de la S-Box sont traités est inconnu.

Si seul l'ordre dans lequel les S-Box sont traitées est aléatoire, on peut envisager une attaque basée sur la recherche des éléments de S-Box qui ne changent jamais le chiffré lorsqu'ils sont modifiés. Cela est possible car l'information concernant l'indice de l'élément qui a été changé est toujours disponible. Si, pour une même valeur d'indice i, on injecte de façon répétée des fautes modifiant l'élément $S[i]$ pour une S-Box (ou un couple de S-Box) aléatoire, et si après plusieurs exécutions avec le même message le chiffré ne change pas, alors on peut raisonnablement supposer que cette valeur d'indice ne représente une bonne supposition pour aucune des sous-clés du premier tour. Le nombre moyen d'exécutions nécessaires pour être sûr de cette information est 22 (d'après le test du collectionneur de coupons tels que défini dans [Knu81]). En effet, la probabilité pour que les 4 couples de S-Box n'aient été tous couverts après 22 fautes répétées, est inférieure à 0,01. L'espérance du nombre d'indices qui sont utilisés dans au moins une S-Box à au moins un tour est de 55,5. Pour chacun d'eux, 4 fautes sont en moyenne suffisantes pour

TAB. 10.3 – Nombre d'hypothèses générées en attaquant des S-Box compressées initialisées en ordre aléatoire. Entre parenthèses, l'optimisation ignorant les modifications dans les deux derniers tours.

Messages	Hypothèses par paire de S-Box		Hypothèses pour la clé du tour 1		Espace de clé entier	
1	55,5	(53,0)	$2^{46,4}$	$(2^{45,8})$	$2^{54,4}$	$(2^{53,8})$
2	48,2	(44,2)	$2^{44,7}$	$(2^{43,7})$	$2^{52,7}$	$(2^{51,7})$
3	42,1	(37,1)	$2^{43,2}$	$(2^{41,7})$	$2^{51,2}$	$(2^{49,7})$
5	32,5	(26,7)	$2^{40,2}$	$(2^{37,9})$	$2^{48,2}$	$(2^{45,9})$
10	18,5	(14,1)	$2^{33,7}$	$(2^{30,6})$	$2^{41,7}$	$(2^{38,6})$
15	12,4	(9,78)	$2^{29,1}$	$(2^{26,4})$	$2^{37,1}$	$(2^{34,4})$
20	9,71	(8,32)	$2^{26,4}$	$(2^{24,6})$	$2^{34,4}$	$(2^{32,6})$

mettre en évidence que cet indice est utilisé. Pour les autres, la faute doit être répétée 22 fois. Il en résulte qu'il faut en moyenne 409 fautes pour traiter l'ensemble des indices pour un message donnée. Par ailleurs, la clé est à rechercher dans un plus grand espace. La table 10.3 présente la complexité de l'attaque dans le cas où les S-Box sont considérées dans un ordre aléatoire. Après avoir exploité un seul message, la clé est à rechercher dans un espace de taille $2^{54,4}$. Pas moins de 10 messages différents sont nécessaires pour ramener la recherche exhaustive moyenne à $2^{41,7}$, voire à $2^{38,6}$ si on ignore les modifications dans les deux derniers tours, ce qui devient plus envisageable. Cependant, la quantité de fautes nécessaires peut rendre le nombre d'injections de fautes irréaliste si l'effet de la faute n'est pas déterministe. De plus, et surtout, cette attaque n'est plus possible lorsque les éléments d'une même S-Box sont traités dans un ordre aléatoire tel que décrit dans l'algorithme de la Figure 10.4.

L'attaque présentée dans cette section requiert un haut degré de précision puisque l'information est dérivée d'une faute ayant eu lieu mais n'ayant pas produit d'effet sur le chiffré. Cela a été possible grâce à la manière dont la faute a été injectée, mais pourrait s'avérer difficilement envisageable avec d'autres méthodes d'injections de fautes. On s'attend généralement à ce qu'une faute réussisse avec une certaine probabilité quand elle est appliquée à un composant [BS03], dans notre cas de figure la probabilité de succès était égal à 1.

Algorithme 10.4 Randomisation des valeurs des S-Box du DES

Entrée : $S = (s_0, s_1, \ldots, s_{63})_{16}$ contenant la S-Box,
un aléa d'entrée $\mathbf{R} \in \{0, \ldots, 63\}$, et un aléa de sortie $r \in \{0, \ldots, 15\}$

Sortie : $RS = (rs_0, rs_1, \ldots, rs_{63})_{16}$ contenant la S-Box randomisée

1. **pour** $i \leftarrow 0$ à 63
2. $\quad rs_i \leftarrow s_{(i \oplus \mathbf{R})} \oplus r$
3. **fin pour**
4. **retourne** RS

FIG. 10.4 – Randomisation des valeurs des S-Box du DES.

10.7 Modifier des valeurs inconnues des S-Box

Si un élément de S-Box peut être modifié, mais que l'attaquant ne sait pas quel élément a été modifié (par exemple, si l'algorithme de la Figure 10.4 est utilisé), l'attaque décrite dans la Section 10.6 ne fonctionne pas. Néanmoins, une attaque peut être menée en implémentant les algorithmes donnés dans [BS97, GT04].

Si une valeur de S-Box est modifiée et utilisée dans le tour 15, et seulement dans le tour 15, alors les idées à la base de l'analyse différentielle de fautes décrite dans [BS97] s'appliquent. La modification de la lecture d'une S-Box dans le quinzième tour changera en moyenne les valeurs d'entrée pour 3,2 S-Box différentes dans le seizième tour, fournissant des différentielles à travers ces S-Box pour du test d'hypothèse de clé.

L'avantage de cette attaque sur l'attaque décrite à la Section 10.6 est que l'effet de la faute désirée peut être vu dans le chiffré. La faute est détectée en calculant la différentielle de la sortie de la S-Box dans le quinzième tour, ce qui est possible en observant le chiffré. Si seulement un quartet de cette valeur n'est pas égal à zéro, alors il y a une forte probabilité que la valeur correspondante de la S-Box ait été utilisée seulement dans le quinzième tour. La probabilité que cet évènement ait lieu est $\left(\frac{63}{64}\right)^{15} \frac{1}{64} = 0.0123$.

Cette probabilité est suffisamment élevée pour qu'un attaquant puisse mener l'attaque plusieurs fois jusqu'à ce que l'évènement désiré soit observé. Une certaine information sur la clé peut alors être dérivée et le processus répété jusqu'à obtenir suffisamment d'information pour mener une recherche exhaustive.

Il peut arriver que l'entrée modifiée de la S-Box soit utilisée dans le quatorzième tour, et que cela conduise à un chiffré qui sera interprété comme si l'entrée de la S-Box avait été utilisée dans le quinzième tour seulement. Cela se produit lorsque la modification dans le quatorzième tour produit une différentielle en sortie de S-Box de 1 bit (car tous les bits de sortie vont dans des S-Box différentes). Il y a 4 valeurs possibles parmi les 15 fautes qui produiront cet effet. La moitié de ces valeurs vont modifier plus d'une S-Box dans le quinzième tour, par le fait qu'elles vont couvrir deux S-Box à cause de la permutation expansive. Cela ne laisse donc que deux valeurs possibles parmi les quinze fautes possibles. La probabilité d'un faux positif de ce type est donc de $\frac{2}{15}\left(\frac{63}{64}\right)^{15} \frac{1}{64} = 0.00165$.

La probabilité d'un faux positif est relativement haute quand elle est comparée à la probabilité d'un évènement qui permet l'attaque. Approximativement 1 détection sur 7 sera un faux positif. Cependant, comme décrit dans [GT04], les mauvaises hypothèses introduites par ces faux positifs n'auront pas un effet majeur sur le succès de l'attaque.

Comme il est détaillé dans la Section 10.6, les valeurs des S-Box sont généralement stockées dans un état compressé de sorte qu'un attaquant peut être forcé de modifier les entrées de deux S-Box à la fois. La probabilité qu'une de ces valeurs soit utilisée dans le quinzième tour seulement est $2\left(\frac{63}{64}\right)^{15} \frac{1}{64} \left(\frac{63}{64}\right)^{16} = 0.0192$.

C'est plus efficace que de ne modifier qu'une seule valeur de S-Box, car la probabilité que la valeur de S-Box soit utilisée dans le quinzième tour est plus élevée. La probabilité changera en fonction de la méthode de compression des S-Box, mais nous n'analyserons que la méthode envisagée jusqu'ici.

La probabilité qu'une valeur de S-Box soit utilisée dans le quatorzième tour et cause un faux positif peut être calculée comme précédemment. Si les valeurs modifiées des deux S-Box sont toutes les deux utilisées dans le quatorzième tour, cela peut aussi provoquer un faux positif. C'est en effet le cas si deux erreurs de 1 bit sont produites, et que ces bits sont utilisés dans la même S-Box au quinzième tour sans avoir été dédoublés par la permutation expansive. La probabilité de ce cas de figure a été calculée, en simulant toutes les combinaisons possibles,

comme valant 89/147 456. La probabilité totale d'un faux positif est donc $2\frac{2}{15}\left(\frac{63}{64}\right)^{15}\frac{1}{64}\left(\frac{63}{64}\right)^{16}+\frac{89}{147\,456}\left(\left(\frac{63}{64}\right)^{15}\frac{1}{64}\right)^{2}=0.00256$.

Dans le cas où le chiffré laisse supposer qu'une modification de valeur à la sortie d'une seule S-Box du tour 15 a eu lieu, la probabilité d'un faux positif est environ la même ($\approx 2/15$), que l'implémentation de S-Box compressées soit utilisée ou non. L'implémentation utilisant des S-Box compressées fournira des résultats plus rapidement car l'évènement désiré se produit avec un probabilité plus forte.

Cette attaque a été implémentée sur le même composant que l'attaque décrite à la Section 10.6 car la faute utilisée était idéale pour modifier les valeurs des S-Box lors de leur création. La première tentative de cette attaque a concerné une implémentation du DES qui n'utilisait que le masquage de données et construisait les S-Box en utilisant l'algorithme de la Figure 10.4. L'outil développé pour cette attaque attendait jusqu'à ce qu'au moins une différentielle ait été trouvée à travers chaque S-Box avant de mener une recherche exhaustive des hypothèses dérivées des injections de fautes. L'outil a retrouvé la clé en 8 minutes.

Une seconde tentative a été menée avec l'addition de délais aléatoires aux niveaux matériel et logiciel, de sorte qu'une faute se produisait avec une plus faible probabilité. Le même outil a mis 20 minutes pour retrouver la clé.

Cette attaque a été plus facile à implémenter que l'attaque décrite à la Section 10.6, car seulement une position d'injection de faute était nécessaire pour attaquer une entrée aléatoire de S-Box. Dans l'attaque précédente il était nécessaire de décaler la position de l'injection de faute pour chaque nouvelle entrée de S-Box visée.

Dans le cas d'une implémentation utilisant les S-Box compressées, il serait logique d'utiliser l'évènement où deux valeurs de S-Box sont utilisées dans le quinzième tour. Comme précédemment, cela peut être observé en cherchant deux quartets ayant une différentielle non nulle dans le chiffré. Cette information peut être combinée avec l'évènement où un seul quartet a une différentielle non nulle dans le chiffré. La probabilité résultante que cela arrive est $2\left(\frac{63}{64}\right)^{15}\frac{1}{64}\left(\frac{63}{64}\right)^{16}+\left(\left(\frac{63}{64}\right)^{15}\frac{1}{64}\right)^{2}=0.0193$.

Comme précédemment, il y a une probabilité de faux positif. Dans ce cas, les évènements correspondant à une ou deux valeurs de S-Box modifiées utilisées dans le quatorzième tour peuvent potentiellement simuler des changements de valeurs dans deux S-Box du quinzième tour. Si une valeur de S-Box change au quatorzième tour, la probabilité que deux valeurs soient modifiées au quinzième tour est de 1/5. Si deux valeurs changent au quatorzième tour, la probabilité que deux valeurs changent au quinzième tour est de 914 609/29 491 200. Ici aussi, ces probabilités ont été calculées en simulant toutes les combinaisons possibles. La probabilité d'un faux positif est alors égale à $2\frac{2}{15}\left(\frac{63}{64}\right)^{15}\frac{1}{64}\left(\frac{63}{64}\right)^{16}+\frac{89}{147\,456}\left(\left(\frac{63}{64}\right)^{15}\frac{1}{64}\right)^{2}+2\frac{1}{5}\left(\frac{63}{64}\right)^{15}\frac{1}{64}\left(\frac{63}{64}\right)^{16}+\frac{914\,609}{29\,491\,200}\left(\left(\frac{63}{64}\right)^{15}\frac{1}{64}\right)^{2}=0.00640$.

La probabilité d'obtenir de l'information utile reste approximativement la même que dans le cas où seulement une modification de S-Box est considérée, mais la probabilité d'un faux positif est 2,5 fois plus élevée. Les données acquises vont donc être beaucoup plus bruitées et vont accroître le temps nécessaire pour conduire l'attaque. Il n'y a donc peu d'intérêt à mener l'attaque de cette manière.

10.8 Contre-mesures

Il y a plusieurs contre-mesures qui peuvent être employées pour protéger un algorithme contre le type d'attaque envisagé aux Sections 10.6 et 10.7. Comme il a été décrit plus haut, la

modification aléatoire des données manipulées n'est pas une contre-mesure efficace contre cette attaque par fautes.

Délais aléatoires : Si une grande précision est nécessaire, l'attaque peut être ralentie au point où un attaquant doutera que l'attaque soit réalisable. Cela vaut tout autant pour les délais aléatoires matériels que logiciels. Une étude de l'effet des délais aléatoires sur la DPA est donnée dans [CCD00], des effets similaires seront constatés quand ils seront utilisés contre des attaques par fautes.

Sommes de contrôle : Si les S-Box ont besoin d'être construites en RAM, elles nécessitent d'être protégées par une somme de contrôle. La méthode la plus simple pour cela serait de XOR-er toutes les valeurs ensemble après que la table a été créée, c'est-à-dire après que la table a été écrite en mémoire. Cela présente l'avantage d'être compatible avec le masquage aléatoire des éléments car le nombre d'entrées de la S-Box est pair. Néanmoins, ce n'est pas adéquat pour se défendre contre l'attaque présentée ci-dessus. Si la somme de contrôle est sur un octet, un attaquant peut modifier plusieurs valeurs et avoir une probabilité de 1/256 d'obtenir une somme de contrôle valide. Une seconde somme de contrôle calculée d'une façon différente pourrait supprimer ce problème, puisque la seconde somme de contrôle peut être choisie de sorte qu'il n'y ait aucune faute qui permette aux deux sommes de contrôle d'être valides.

Redondance : Il est déjà connu qu'il est conseillé de répéter les 3 ou 4 premiers tours d'un algorithme à clé secrète pour le protéger d'attaques par collision comme celle décrite dans [Hem04]. Les fonctions d'initialisation peuvent être répétées et le contenu de la mémoire vérifié, de la même manière que des tours d'un algorithme sont répétés pour assurer qu'aucune faute exploitable ne peut être injectée. Cependant, cela est prohibitivement coûteux en temps, spécialement pour la construction des S-Box.

Initialisation aléatoire de la mémoire : Toute zone de "travail" de RAM utilisée peut être remplie avec des valeurs aléatoires indépendantes avant le début de l'algorithme. La faisabilité de l'attaque serait alors fonction de la qualité des valeurs aléatoires utilisées. Si, par exemple, un LFSR était utilisé pour générer ces valeurs, il serait possible de prédire la valeur d'un octet si l'octet précédent était connu. Cela peut signifier que l'attaque décrite à la Section 10.4 serait toujours possible avec très peu de modifications *i.e.* la recherche finale serait de 2^{16} au lieu de 2^8 car l'attaquant aurait à exhauster la valeur initiale possible de l'aléa utilisé.

10.9 Conclusion

Plusieurs attaques différentes où des fautes sont injectées au début d'implémentations sûres de l'AES ou du DES ont été présentées. L'implémentation de certaines de ces attaques a été brièvement décrite.

Ces attaques sont des attaques génériques et peuvent être considérées comme applicables à toute implémentation à clé secrète. Ces attaques montrent que les contre-mesures à la DPA ne sont pas une barrière absolue contre les attaques par fautes, et que dépendre de la redondance de tours n'est pas suffisant pour obtenir une implémentation sûre sur cartes à puce.

Quatrième partie

Les attaques sur algorithmes inconnus

Chapitre 11

Amélioration d'une cryptanalyse SCARE contre un algorithme GSM A3/A8 secret

Sommaire

Les attaques physiques, par analyse de canaux auxiliaires ou de fautes, supposent généralement que l'algorithme cryptographique est connu de l'attaquant. Ignorer la fonction analysée rend beaucoup plus difficile la conception d'une attaque, et place l'adversaire devant deux objectifs possibles. Il peut tout d'abord souhaiter, tout comme s'il connaissait l'algorithme, retrouver la valeur de la clé. Mais il peut également vouloir connaître tout ou partie des détails fonctionnels qui lui sont cachés. Ce chapitre donne un exemple d'une telle rétro-conception réalisée par analyse du courant sur un algorithme secret de téléphonie mobile. Notre contribution, réalisée dès 2003, sera publiée à Iciss '07 [Cla07b].

L'analyse des canaux auxiliaires est reconnue depuis plusieurs années comme un moyen pratique et puissant de révéler les clés secrètes d'algorithmes cryptographiques publics. Rarement ce type de cryptanalyse a été appliqué pour rétro-concevoir une partie non triviale des spécifications d'un algorithme propriétaire. La cible ici n'est plus une clé secrète mais les spécifications non révélées de l'algorithme cryptographique même.

Dans [Nov03], Roman Novak a décrit comment retrouver le contenu d'une (parmi deux) table de substitution d'une instance secrète d'un algorithme A3/A8, l'algorithme d'authentification et de génération de clés de session pour les réseaux GSM. Son attaque présente néanmoins deux inconvénients d'un point de vue pratique. D'abord, pour retrouver une des tables de sub-

stitution (T_2), l'attaquant doit connaître le contenu de l'autre (T_1). Ensuite, l'attaquant doit également connaître la valeur de la clé secrète K.

Dans ce chapitre, nous améliorons la cryptanalyse de Roman Novak et montrons comment retrouver *les deux* tables de substitution (T_1 et T_2) *sans aucune connaissance préalable de la clé secrète*. Qui plus est, notre attaque permet également de retrouver la clé secrète. Avec cette contribution, nous voulons présenter une attaque pratique par *analyse des canaux auxiliaires pour la rétro-conception* (SCARE). Nous anticipons également un intérêt croissant pour ce nouveau type d'exploitation du signal de canal auxiliaire, et rappelons, si besoin était, que la sécurité ne peut être obtenue par l'obscurité seulement.

11.1 Introduction

Les implémentations sûres d'algorithmes cryptographiques sur des composants de sécurité comme les cartes à puce ont été étudiées avec attention, particulièrement depuis que les attaques par canaux auxiliaires ont été initialement proposées par Paul Kocher [Koc96]. Ce type d'attaques dérive de l'information sur l'exécution d'un algorithme sensible, soit à partir de la durée d'exécution, soit à partir de la mesure de la consommation de courant ou de l'émanation électromagnétique. L'exploitation du signal peut aller des simples observations à des analyses statistiques plus avancées. Une simple observation (SPA, SEMA) permet de distinguer la structure grossière de l'algorithme (par exemple, le nombre de tours) ou de détecter la présence/absence d'instructions ou de blocs d'instructions spécifiques. Les analyses statistiques ressemblent plus à un test d'hypothèse, ou bien par réduction de bruit grâce au moyennage et rehaussement d'une petite contribution du signal dans le cas des techniques différentielles (DPA, DEMA) [KJJ99, QS00], ou bien par un ajustement de modèle plus global et robuste dans le cas des analyses basées sur la corrélation (CPA, CEMA) [BCO04]. Bien que de nombreuses variantes de ces techniques ont été proposées ces dernières années, l'objectif était invariablement le dévoilement d'une donnée sensible liée à l'utilisateur comme par exemple une clé privée ou un code personnel d'identification (PIN).

Dans [Nov03], Roman Novak exploite la fuite des canaux auxiliaires pour obtenir des détails non triviaux concernant les spécifications secrètes d'un algorithme de chiffrement par bloc. L'algorithme cible est une des nombreuses instances propriétaires de l'A3/A8, l'algorithme GSM d'authentification et de génération de clés de session. Cela a ouvert une brèche dans ce nouveau type d'attaques cryptanalytiques : les attaques SCARE par *analyses des canaux auxiliaires pour la rétro-conception*. Par la suite, Rémy Daudigny, Hervé Ledig, Frédéric Muller et Frédéric Valette [DLMV05] ont également appliqué ce type de technique pour "retrouver" les détails de l'algorithme standard DES.

Sans dévoiler plus de détails sur l'algorithme cible que ceux que l'on peut trouver dans [Nov03], nous présentons deux améliorations de l'attaque de Novak. L'attaque décrite dans son article retrouve les entrées d'une table de substitution secrète T_2 à partir de la connaissance de l'autre table de substitution T_1 et de la clé secrète K. Les attaques SCARE que nous présentons ici permettent de retrouver tout à la fois les entrées des deux tables de substitution secrète et la clé secrète. Nous rendons ainsi plus pratique cette attaque par rétro-conception en élargissant son applicabilité. Notre attaque confirme que la sécurité ne peut pas être atteinte par l'obscurité seulement. Cela est encore plus vrai à cause des attaques SCARE. Le niveau de sécurité des algorithmes cryptographiques propriétaires (c'est-à-dire, à spécifications secrètes) est parfois plus faible que celui des algorithmes publiquement examinables. Les attaques SCARE étant à même de dévoiler leurs spécifications secrètes (les tables de substitution dans notre cas de figure), ces

algorithmes ont alors plus de chance de succomber aux attaques cryptanalytiques classiques.

La suite de ce chapitre est organisée de la manière suivante. Dans la Section 11.2, nous rappelons les principes de l'attaque de Novak ainsi que ses hypothèses sous-jacentes. Nous en proposons alors une interprétation en termes de graphes, et discutons de sa faisabilité théorique. Les deux sections suivantes décrivent l'essentiel de notre contribution : la Section 11.3 explique comment retrouver la table de substitution T_1 à l'aide de la clé secrète K seulement, et la Section 11.4 explique comment le faire sans même la connaissance de la clé secrète. Dans la Section 11.5 nous discutons de la menace des attaques SCARE et suggérons des contre-mesures pratiques. Enfin, nous concluons ce chapitre à la Section 11.6.

11.2 Retrouver la table T_2 connaissant la clé K et la table T_1

11.2.1 Description de l'attaque de Novak

Comme pour toute instance de l'A3/A8 GSM, l'algorithme cryptographique attaqué par Roman Novak dans [Nov03] prend en entrée un challenge de 16 octets, $M = (m_0, \ldots, m_{15})$, ainsi qu'une clé secrète de 16 octets, $K = (k_0, \ldots, k_{15})$, et produit en sortie un code d'authentification de message de 32 bits, S_{RES}, ainsi qu'une clé de session de 64 bits, K_C, pour le chiffrement de la voix.

L'attaque de Novak repose sur trois hypothèses :

Hypothèse 1 (Observation)**.** *L'attaquant est supposé être capable de détecter, par analyse des canaux auxiliaires, si des valeurs intermédiaires à deux instants donnés de l'exécution de l'algorithme sont les mêmes.*

Bien qu'elle ne soit pas irréaliste, nous pensons qu'il est utile de préciser ici pourquoi, et dans quelles circonstances, cette hypothèse pourrait être vérifiée en pratique. Par exemple, cette hypothèse n'est pas vérifiée si le composant fuit parfaitement selon le modèle linéaire en le poids de Hamming de la donnée manipulée. Dans ce cas, deux données différentes ayant des poids de Hamming identiques ne peuvent être distinguées l'une de l'autre par l'observation des canaux auxiliaires. Néanmoins, nous avons vérifié et confirmé par la pratique que certaines conditions expérimentales sont compatibles avec l'Hypothèse 1. Il est en effet possible de détecter l'égalité de résultats intermédiaires dans le modèle de la distance de Hamming. Cela peut être réalisé en consolidant les informations d'égalité de distance de Hamming par rapport à différents états de référence.

Notons que cette hypothèse a déjà été utilisée par le passé dans [SWP03] et [SLFP04] pour imaginer des attaques par canaux auxiliaires basées sur les collisions dans le contexte classique de *révélation de la clé*.

Hypothèse 2 (Connaissance structurelle *a priori*)**.** *L'attaquant est supposé connaître la structure du tout début de l'algorithme cible proposé. Exactement, il doit savoir que les toutes premières opérations agissant sur les données d'entrées consistent en 16 applications de la fonction f décrite à la Figure 11.1. L'attaquant sait également que les deux fonctions T_1 et T_2 sont des tables de substitution d'octets, mais il en ignore les valeurs. Chaque paire d'octets (m_i, k_i) est initialement transformée en la paire d'octets $f(m_i, k_i) = (c_i, d_i)$ avec :*

$$\begin{aligned} c_i &= T_2\big((m_i \oplus k_i) \oplus T_1(d_i)\big) \\ d_i &= T_2\big(T_1(m_i \oplus k_i) \oplus m_i\big) \ . \end{aligned}$$

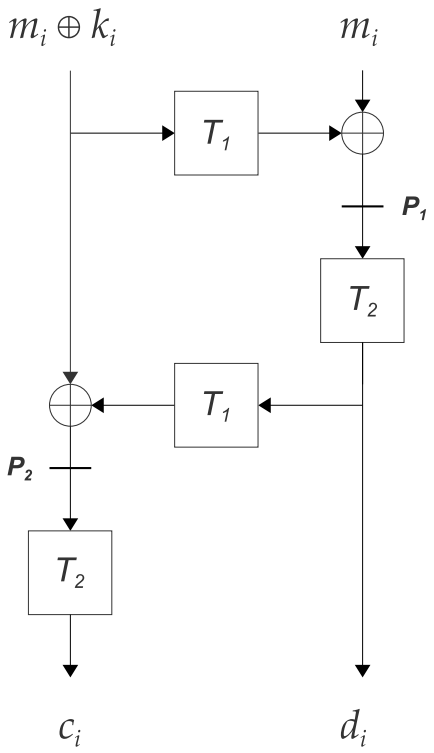


FIG. 11.1 – Synopsis du début de l'algorithme inconnu.

Notons que dans son but de retrouver la valeur de la (des) table(s) de substitution, l'attaquant n'a aucun besoin d'en savoir davantage sur la structure de l'algorithme. En particulier, il n'a pas besoin de connaître le nombre de tours et, le cas échéant, le nombre de sous-tours par tour[45], ni comment se réalise le processus de diffusion dans l'algorithme.

Hypothèse 3 (Connaissance *a priori* sur les données)**.** *Afin de retrouver la table T_2, l'attaquant est supposé connaître le contenu complet de la table T_1, ainsi que la clé secrète K.*

Cette dernière hypothèse est de loin la limitation principale de l'attaque de Novak. En particulier, il est difficile d'imaginer qu'un attaquant ait pu se débrouiller pour obtenir le contenu de T_1 sans connaître T_2 également. De plus, la connaissance nécessaire de la clé secrète constitue une barrière additionnelle.

Dans ce qui suit, nous appelons *itération i* l'application de f à une paire (m_i, k_i) au tout début de l'algorithme. Une observation de deux valeurs intermédiaires à deux itérations différentes (respectivement, à la même itération) est appelée observation *inter-itérations* (respectivement, observation *intra-itération*).

Définition 7. *Nous notons $\mathcal{R}^{(2)}_{\alpha,\beta}$ l'ensemble des paires d'entrées (m_α, m'_β) produisant deux valeurs identiques au point P_2 (voir la Figure 11.1) pour les itérations α et β, c'est-à-dire :*

$$\begin{aligned}\mathcal{R}^{(2)}_{\alpha,\beta} = \Big\{(m_\alpha, m'_\beta) \quad : \quad & T_1\Big(T_2\big(T_1(m_\alpha \oplus k_\alpha) \oplus m_\alpha\big)\Big) \oplus (m_\alpha \oplus k_\alpha) \\ & = T_1\Big(T_2\big(T_1(m'_\beta \oplus k_\beta) \oplus m'_\beta\big)\Big) \oplus (m'_\beta \oplus k_\beta)\Big\} \ .\end{aligned}$$

D'après l'Hypothèse 1, il est possible de collecter l'ensemble des paires

$$\mathcal{R}^{(2)} = \bigcup_{\alpha,\beta} \mathcal{R}^{(2)}_{\alpha,\beta}$$

pour lesquelles une égalité intermédiaire se produit au point P_2 au début de l'algorithme.

Notons que, en incrémentant tous les octets du message en parallèle, seulement 256 invocations de l'algorithme sont nécessaires pour collecter l'ensemble de paires de $\mathcal{R}^{(2)}$.

Chacune de ces paires relie entre elles deux entrées différentes de T_2 par l'équation :

$$T_1\big(T_2(x_\alpha)\big) \oplus T_1\big(T_2(x'_\beta)\big) = d \ , \tag{11.1}$$

avec

$$\begin{cases} x_\alpha = T_1(m_\alpha \oplus k_\alpha) \oplus m_\alpha \\ x'_\beta = T_1(m'_\beta \oplus k_\beta) \oplus m'_\beta \\ d = (m_\alpha \oplus k_\alpha) \oplus (m'_\beta \oplus k_\beta) \end{cases} , \tag{11.2}$$

où x_α, x'_β et d sont connus d'après l'Hypothèse 3.

Chaque relation conforme à l'Equation (11.1) donne l'opportunité de relier entre elles, si cela n'est pas déjà fait, deux entrées différentes de T_2. Cela décrémente d'une unité le degré de liberté de la table, c'est-à-dire le nombre restant d'entrées indépendantes de T_2.

Lorsque le degré de liberté de T_2 est égal à 1, toutes les entrées de T_2 peuvent être déduites de n'importe laquelle d'entre elles, par exemple $T_2(0)$. Il reste alors 256 candidats possibles pour

[45] Un autre algorithme A3/A8 GSM largement utilisé et maintenant rendu public, le COMP128, possède une structure en tours, chacun divisé en 5 sous-tours similaires.

le contenu complet de la table. L'attaquant peut alors identifier la valeur correcte de T_2 par des techniques de type DPA [KJJ99] ou de type CPA (basées sur la corrélation) [BCO04].

Alternativement, et seulement s'il connaît les autres détails de l'algorithme, l'attaquant peut aussi identifier la valeur correcte de T_2 par comparaison clair/chiffré classique. Nous insistons sur le fait que cette connaissance n'est pas du tout nécessaire pour aucune des attaques décrites dans ce chapitre. Seule la connaissance de la structure du début de l'algorithme (Hypothèse 2) est requise.

11.2.2 Interprétation par graphe

Dans cette section, nous proposons une interprétation des principes de l'attaque sous forme de graphe, nous discutons de certains aspects liés à son implémentation, et présentons des arguments théoriques en faveur de sa faisabilité.

Nous avons remarqué qu'observer des valeurs intermédiaires égales au point P_2 relie entre elles, par l'intermédiaire du paramètre d, les deux entrées de T_2 pour les indices x et x'. Cette relation de base suggère une interprétation en terme de graphe de la connaissance courante que l'attaquant a alors obtenue au sujet des contraintes sur T_2.

Le graphe des contraintes sur T_2 est un graphe non orienté $\mathcal{G}^{(2)}$ dont les sommets sont les indices des différentes entrées de T_2 et pour lequel une arête étiquetée d entre deux sommets x et x' (notée $x \overset{d}{\frown} x'$) signifie que les valeurs des entrées $T_2(x)$ et $T_2(x')$ sont liées par la relation :

$$T_1\big(T_2(x)\big) \oplus T_1\big(T_2(x')\big) = d \ . \tag{11.3}$$

Au début de l'attaque, le graphe $\mathcal{G}^{(2)}$ ne contient aucune arête, chaque sommet entre 0 et 255 étant isolé de chaque autre. Cela signifie que chaque entrée de la table est *a priori* indépendante de toutes les autres (à ceci près qu'elles sont toutes différentes puisque T_2 est une permutation). Il y a alors 256 composantes connexes différentes, chacune ne comprenant qu'un sommet. Le degré de liberté de T_2 est alors aussi égal à 256.

Chaque fois qu'une relation comme l'Equation (11.3) doit être exploitée, l'attaquant connecte les sommets x et x' (s'ils ne l'étaient pas déjà) par une arête étiquetée d. Il en résulte un graphe contenant une composante connexe de moins. On note que le nombre d'entrées indépendantes de T_2, son degré de liberté, reste donc toujours égal au nombre de composantes connexes de $\mathcal{G}^{(2)}$.

Proposition 3 (Transitivité sur les arêtes)**.** *Pour tous sommets x, x', x'', et toutes étiquettes d'arêtes d_1 et d_2,*

$$x \overset{d_1}{\frown} x' \ \textit{et}\ x' \overset{d_2}{\frown} x'' \implies x \overset{d_1 \oplus d_2}{\frown} x'' \ .$$

Démonstration. Cela provient du fait que

$$\Big(T_1\big(T_2(x)\big) \oplus T_1\big(T_2(x')\big)\Big) \oplus \Big(T_1\big(T_2(x')\big) \oplus T_1\big(T_2(x'')\big)\Big) = T_1\big(T_2(x)\big) \oplus T_1\big(T_2(x'')\big) \ .$$

□

Cette proposition montre qu'il est possible d'assurer que toute composante connexe de $\mathcal{G}^{(2)}$ soit complètement connexe.

Exploitation pratique des observations

D'un point de vue pratique, il existe un moyen, économe en mémoire, de gérer et de maintenir l'information contenue dans $\mathcal{G}^{(2)}$. Il suffit pour cela de définir deux tableaux de 256 octets, que nous notons `comprep` et `delta`, tels que :

- `comprep`$[x]$ représente un identifiant de la composante connexe $comp(x)$ de x. Par convention, il est défini dynamiquement comme le plus petit sommet appartenant à $comp(x)$. Ce sommet peut être vu comme un représentant de $comp(x)$.
- `delta`$[x]$ représente le paramètre d de la relation liant le sommet x et le représentant de $comp(x)$ (`comprep`$[x]$). En particulier,

$$\forall\, x \in \mathcal{G}^{(2)},\ \texttt{delta}\big[\texttt{comprep}[x]\big] = 0 \ .$$

Le processus d'exploitation des relations démarre avec `comprep` $= \{0, 1, \ldots, 255\}$ et `delta` $= \{0, 0, \ldots, 0\}$, ce qui signifie que chaque sommet forme une composante connexe par lui-même, dont il est évidemment le représentant.

Chaque fois qu'une relation est exploitée, la fonction `AddRelationToGraph`(x, x', d) définie à la Figure 11.2 est appelée et modifie éventuellement la structure du graphe en réunissant, si elles étaient distinctes, les composantes connexes de x et de x'.

Cette opération préserve tout à la fois la convention que `comprep`$[x]$ est minimal dans $comp(x)$, et la propriété induite par la Proposition 3. Elle assure également que toutes les composantes connexes sont complètement connexes.

Le processus s'arrête ou bien lorsqu'il n'y a plus de relation à exploiter, ou bien lorsque le graphe entier est complètement connexe, ce qui se détecte par le fait que `comprep` $= \{0, 0, \ldots, 0\}$. Dans ce dernier cas, `delta` contient toute l'information nécessaire pour inférer un candidat possible pour T_2 à partir de chaque valeur de son premier élément $T_2(0)$. L'attaque a alors réussi.

Notons que dans le cas d'une attaque non aboutie, le nombre de candidats possibles pour T_2 croît rapidement avec le nombre de composantes connexes qu'il reste dans `comprep` (degré de liberté). Cela peut devenir prohibitif si le nombre de composantes connexes restant n'est pas suffisamment petit. Cela motive la discussion qui suit, sur la connectivité de $\mathcal{G}^{(2)}$ qui résulte de l'exploitation de toutes les relations.

Connectivité résultante de $\mathcal{G}^{(2)}$

Nous évaluons d'abord le nombre de relations qui peuvent être collectées durant l'attaque quand toutes les valeurs possibles d'octets de message sont prises à toutes les itérations possibles.

Pour toute clé secrète $K = (k_0, \ldots, k_{15})$, soit

$$l_K = \mathrm{Card}(\{k_i\}_{i=0\ldots15})$$

le nombre d'octets distincts de K. Notons également $g(m, k)$ la valeur au point P_2 lorsque m et k sont les octets d'entrée de la fonction f.

Pour toute valeur intermédiaire possible $z \in \{0, \ldots, 255\}$ au point P_2, nous définissons également $\mathcal{S}_K(z)$ comme étant l'ensemble de toutes les paires (m, k) qui provoquent la valeur intermédiaire z à une quelconque itération :

$$\mathcal{S}_K(z) = \{(m, k) : k = k_i \text{ pour un certain } i, \text{ et } g(m, k) = z\} \ .$$

Enfin, posons $s_K(z) = \mathrm{Card}(\mathcal{S}_K(z))$.

Algorithme 11.1 La fonction `AddRelationToGraph`(x, x', d)

Entrée : $\mathcal{G}$ donné par les tableaux `comprep` et `delta`
(x, x', d) une relation $x \overset{d}{\frown} x'$ observée

Sortie : $\mathcal{G}$ avec $comp(x)$ et $comp(x')$ réunies

1. **si** (`comprep`$[x] =$ `comprep`$[x']$) **alors**
2. **retourne** $\mathcal{G}$
3. **fin si**
4. **si** (`comprep`$[x] >$ `comprep`$[x']$) **alors**
5. échanger x et x'
6. **fin si**
7. **pour tout** $y \in comp(x') \setminus \{x'\}$
8. `AddPointToGraph`$(x, y, d \oplus$ `delta`$[x'] \oplus$ `delta`$[y])$
9. **fin pour**
10. `AddPointToGraph`(x, x', d)
11. **retourne** $\mathcal{G}$

FIG. 11.2 – La fonction `AddRelationToGraph`(x, x', d).

Algorithme 11.2 La fonction `AddPointToGraph`(x, y, d)

Entrée : $\mathcal{G}$ donné par les tableaux `comprep` et `delta`
(x, y, d) une relation $x \overset{d}{\frown} y$

Sortie : $\mathcal{G}$ avec y ajouté à $comp(x)$

1. **si** (`comprep`$[x] \neq$ `comprep`$[y]$) **alors**
2. `comprep`$[y] \leftarrow$ `comprep`$[x]$
3. `delta`$[y] \leftarrow$ `delta`$[x] \oplus d$
4. **fin si**
5. **retourne** $\mathcal{G}$

FIG. 11.3 – La fonction `AddPointToGraph`(x, y, d).

Chaque paire $\big((m_\alpha, k_\alpha), (m'_\beta, k_\beta)\big)$ d'éléments de $\mathcal{S}_K(z)$ induit une collision locale au point P_2. D'après l'Equation (11.2), la paire correspondante (x_α, x'_β) d'entrées de T_2 forme une arête qui doit être ajoutée au graphe de contraintes sur T_2. Le nombre de telles arêtes pour un z donné est *a priori* $\binom{s_K(z)}{2}$, mais nombre d'entre elles peuvent être déduites à partir d'autres d'après la propriété de transitivité (Proposition 3). Comme elles ne doivent pas être comptées comme de nouvelles relations, le nombre d'arêtes indépendantes apportées au graphe $\mathcal{G}^{(2)}$ par $\mathcal{S}_K(z)$ n'est que de $s_K(z) - 1$.[46] Le nombre total de telles arêtes s'élève à

$$n_K = \sum_{z=0}^{255} (s_K(z) - 1) = 256 \cdot l_K - 256 = 256 \cdot (l_K - 1) \ .$$

En supposant que $g(m, k)$ se comporte comme une fonction aléatoire à valeur dans $\{0, \ldots, 255\}$, les ensembles :

$$\{x = T_1(m \oplus k) \oplus m : (m, k) \in \mathcal{S}_K(z)\}_z$$

se comportent comme des échantillons aléatoires de sommets, et l'évolution de $\mathcal{G}^{(2)}$ peut être modélisée comme un processus de graphe aléatoire.

Ce type de structure et l'évolution de leurs composantes sont étudiés en détails par la théorie des graphes. Un résultat asymptotique dû à Paul Erdős et Alfréd Rényi [ER60] établit que pour un nombre de sommets n, et lorsque $n \to \infty$, un graphe devient presque certainement complètement connecté dès lors que l'on a placé sur celui-ci $m \sim \frac{1}{2} n \ln n$ arêtes aléatoires.

Pour $n = 256$, le graphe est connecté dès lors qu'il contient $m \approx 710$ arêtes. D'après [MOV97], la distribution de probabilité du nombre de valeurs distinctes d'octets de clé est définie par :

$$\mathcal{P}\,(\,l_K = t\,) = \left\{ {16 \atop t} \right\} \cdot \frac{\prod_{k=0}^{t-1}(256 - k)}{256^{16}} \ ,$$

où $\left\{ {16 \atop t} \right\}$ représente le nombre de Stirling de seconde espèce. Nous en déduisons que :

$$\mathcal{P}\,(\,l_K \geqslant 13\,) = \mathcal{P}\,(\,n_K \geqslant 3072\,) \geqslant 0.999 \ .$$

Ceci établit qu'il y a beaucoup plus de relations qu'il en est nécessaire pour que $\mathcal{G}^{(2)}$ soit connecté. Nous avons confirmé ce résultat par de nombreuses simulations avec des permutation T_1 et T_2 aléatoires, dans lesquelles il apparaît que l'exploitation des relations intra-itération seulement est déjà, à elle seule, toujours suffisante pour mener à un graphe complètement connexe.

11.3 Retrouver la table T_1 connaissant la clé K

L'attaque présenté dans [Nov03] suppose la possibilité de détecter des égalités de résultats intermédiaires au point P_2 ; l'exploitation de telles collisions locales permettant de retrouver T_2.

Dans cette section, nous présentons une méthode similaire qui permet de retrouver T_1 par observation d'égalités de résultats intermédiaires au point P_1 (voir la Figure 11.1).

Comparée à l'attaque de Novak, la nôtre repose sur les mêmes hypothèses d'observation et de connaissance structurelle (Hypothèses 1 and 2), mais l'hypothèse de connaissance *a priori* sur les données (Hypothèses 3) est allégée et remplacée par la suivante :

Hypothèse 3' (Connaissance *a priori* [allégée] sur les données)**.** *Afin de retrouver la table T_1, l'attaquant est supposé connaître la clé secrète K. (La connaissance de T_2 n'est pas nécessaire.)*

[46] Nous négligeons le cas très improbable où $s(z) = 0$.

Définition 8. *Nous notons* $\mathcal{R}^{(1)}_{\alpha,\beta}$ *l'ensemble des paires d'entrées* (m_α, m'_β) *produisant deux valeurs identiques au point* P_1 *(voir la Figure 11.1) pour les itérations* α *and* β*, c'est-à-dire :*

$$\mathcal{R}^{(1)}_{\alpha,\beta} = \left\{(m_\alpha, m'_\beta) \,:\, T_1(m_\alpha \oplus k_\alpha) \oplus m_\alpha = T_1(m'_\beta \oplus k_\beta) \oplus m'_\beta\right\} \ .$$

De manière similaire à la Section 11.2.1, chaque paire $(m_\alpha, m'_\beta) \in \mathcal{R}^{(1)}_{\alpha,\beta}$ relie entre elles deux entrées différentes de T_1 par l'équation :

$$T_1(x_\alpha) \oplus T_1(x'_\beta) = d \ , \tag{11.4}$$

avec

$$\begin{cases} x_\alpha = m_\alpha \oplus k_\alpha \\ x'_\beta = m'_\beta \oplus k_\beta \\ d = m_\alpha \oplus m'_\beta \end{cases} \ , \tag{11.5}$$

où ici aussi, d'après l'Hypothèse 3', les paramètres x_α, x'_β et d sont connus de l'attaquant.

Nous faisons une remarque spécifique dans ce cas :

Proposition 4. $\forall \alpha, \beta \in \{0, \ldots, 15\}$*, nous avons*[47] $\mathcal{R}^{(1)}_{\beta,\beta} = \mathcal{R}^{(1)}_{\alpha,\alpha} \oplus (k_\alpha \oplus k_\beta)$.

Démonstration. $\forall m, m' \in \{0, \ldots, 255\}$, nous avons

$$\begin{aligned}
(m, m') \in \mathcal{R}^{(1)}_{\alpha,\alpha} \quad &\iff \quad T_1(m \oplus k_\alpha) \oplus m = T_1(m' \oplus k_\alpha) \oplus m' \\
&\iff \quad T_1\Big(\big(m \oplus (k_\alpha \oplus k_\beta)\big) \oplus k_\beta\Big) \oplus \big(m \oplus (k_\alpha \oplus k_\beta)\big) \\
&\qquad\quad = T_1\Big(\big(m' \oplus (k_\alpha \oplus k_\beta)\big) \oplus k_\beta\Big) \oplus \big(m' \oplus (k_\alpha \oplus k_\beta)\big) \\
&\iff \quad \big(m \oplus (k_\alpha \oplus k_\beta), m' \oplus (k_\alpha \oplus k_\beta)\big) \in \mathcal{R}^{(1)}_{\beta,\beta} \\
&\iff \quad (m, m') \oplus (k_\alpha \oplus k_\beta) \in \mathcal{R}^{(1)}_{\beta,\beta} \ .
\end{aligned}$$

□

Cela implique pour l'attaquant que l'information sur T_1 apportée par l'ensemble de relations $\mathcal{R}^{(1)}_{\alpha,\alpha}$ est la même que celle apportée par tout autre $\mathcal{R}^{(1)}_{\beta,\beta}$. Il n'est donc utile d'exploiter qu'un seul des 16 ensembles de relations intra-itération. Heureusement, cette remarque ne s'applique pas aux ensembles de relations inter-itérations. Chacun des ensembles de relations inter-itérations est *a priori* informatif. Comparé au cas où l'attaquant retrouve T_2 en observant au point P_2, le nombre d'ensembles de relations $\mathcal{R}^{(1)}_{\alpha,\beta}$ à exploiter est réduit de $16 + \binom{16}{2} = 136$ à $1 + \binom{16}{2} = 121$. Cela ne représente pas une pénalité importante pour mener l'attaque.

Le processus d'exploitation des relations est le même que celui décrit à la Section 11.2.2

Le graphe $\mathcal{G}^{(1)}$ de contraintes sur T_1 rassemble toutes les arêtes $x \overset{d}{\frown} x'$ pour lesquelles les entrées $T_1(x)$ et $T_1(x')$ sont liées entre elles par la relation :

$$T_1(x) \oplus T_1(x') = d \ . \tag{11.6}$$

La discussion au sujet de la connectivité de $\mathcal{G}^{(1)}$ est essentiellement la même que pour celle de $\mathcal{G}^{(2)}$ — $g(m, k)$ étant défini comme la valeur au point P_1, et le sommet x étant égal à $m \oplus k$ plutôt qu'à $T_1(m \oplus k) \oplus m$.

[47] Par abus de notation, pour un vecteur $\vec{x}$ et un scalaire δ, $\vec{x} \oplus \delta$ signifie que chaque composante de $\vec{x}$ est XOR-ée avec δ, c'est-à-dire : si $\vec{x} = (x_0, \ldots, x_t)$ alors $\vec{x} \oplus \delta = (y_0, \ldots, y_t)$ avec $y_i = x_i \oplus \delta$.

$\mathcal{G}^{(1)}$ est lui aussi modélisé comme un graphe aléatoire mais il y a légèrement moins d'arêtes aléatoires disponibles du fait de la Proposition 4. Néanmoins, des simulations ont confirmé que ce nombre de relations observées est encore largement suffisant pour mener l'attaque avec succès.

11.4 Retrouver la table T_1 sans connaître la clé K

Dans le cas où il ne connaît pas la clé secrète $K = (k_0, \ldots, k_{15})$, l'attaquant peut faire des suppositions sur ses octets successifs.

En faisant une supposition g_0 sur k_0, l'attaquant est déjà capable d'exploiter les relations appartenant à $\mathcal{R}^{(1)}_{0,0}$. Plus généralement, en faisant une supposition $\vec{g}_t = (g_0, \ldots, g_{t-1})$ sur les t premiers octets $\vec{k}_t = (k_0, \ldots, k_{t-1})$ de la clé, l'attaquant est capable d'exploiter toutes les relations dans

$$\mathcal{R}^{(1)}_t \triangleq \bigcup_{0 \leqslant \alpha, \beta < t} \mathcal{R}^{(1)}_{\alpha,\beta} \ .$$

Pour chaque supposition $\vec{g}_t$, notons $\mathcal{G}^{(1)}(\vec{g}_t)$ le graphe de contraintes sur T_1 après avoir exploité toutes les relations dans $\mathcal{R}^{(1)}_t$, et en supposant que $\vec{k}_t = \vec{g}_t$.

Le graphe $\mathcal{G}^{(1)}(\vec{g}_t)$ est dit *inconsistant* dès lors que la proposition de transitivité sur les arêtes (Proposition 3) n'est pas vérifiée ; dans le cas contraire il est dit *consistant*. Pour toute supposition incorrecte $\vec{g}_t$, la probabilité qu'a le graphe $\mathcal{G}^{(1)}(\vec{g}_t)$ d'être inconsistant augmente avec t. Cela suggère un algorithme de recherche en largeur d'abord pour retrouver T_1.

Notons $\mathcal{C}_t$ l'ensemble de tous les couples $\big(\vec{g}_t, \mathcal{G}^{(1)}(\vec{g}_t)\big)$, pour lesquels $\mathcal{G}^{(1)}(\vec{g}_t)$ est consistant. Au niveau de profondeur $t+1$, chaque supposition g_t sur l'octet de clé k_t est combinée avec chaque $\vec{g}_t$ de $\mathcal{C}_t$. Concernant l'exploitation des relations, il en résulte que chaque graphe $\mathcal{G}^{(1)}(\vec{g}_t)$ de $\mathcal{C}_t$ devient davantage contraint, par les nouvelles relations de $\mathcal{R}^{(1)}_{t+1} \setminus \mathcal{R}^{(1)}_t$. Pourvu que le graphe actualisé $\mathcal{G}^{(1)}(\vec{g}_{t+1})$ demeure consistant, chaque couple $\big(\vec{g}_{t+1}, \mathcal{G}^{(1)}(\vec{g}_{t+1})\big)$ est alors conservé dans $\mathcal{C}_{t+1}$. Les graphes devenus inconsistant par adjonction des contraintes nouvellement exploitées ne sont plus considérés.

Avant d'aller plus loin, nous donnons une légère généralisation de la Définition 8 :

Définition 8'. *Pour tout $\vec{k}$, on définit $\mathcal{R}^{(1)}_{\alpha,\beta}(\vec{k})$ comme étant l'ensemble des paires d'entrées (m_α, m'_β) produisant deux valeurs identiques au point P_1 pour les itérations α and β, quand la clé secrète est $\vec{k}$.*

Proposition 5. *$\forall \alpha, \beta \in \{0, \ldots, 15\}, \forall \delta \in \{0, \ldots, 255\}$, nous avons*

$$\mathcal{R}^{(1)}_{\alpha,\beta}(\vec{k} \oplus \delta) = \mathcal{R}^{(1)}_{\alpha,\beta}(\vec{k}) \oplus \delta \ .$$

Démonstration. $\forall m, m' \in \{0, \ldots, 255\}$, nous avons

$$\begin{aligned}
(m, m') \in \mathcal{R}^{(1)}_{\alpha,\beta}(\vec{k}) \quad &\iff \quad T_1(m \oplus k_\alpha) \oplus m = T_1(m' \oplus k_\beta) \oplus m' \\
&\iff \quad T_1\big((m \oplus \delta) \oplus (k_\alpha \oplus \delta)\big) \oplus (m \oplus \delta) \\
&\qquad\quad = T_1\big((m' \oplus \delta) \oplus (k_\beta \oplus \delta)\big) \oplus (m' \oplus \delta) \\
&\iff \quad (m \oplus \delta, m' \oplus \delta) \in \mathcal{R}^{(1)}_{\alpha,\beta}(\vec{k} \oplus \delta) \\
&\iff \quad (m, m') \oplus \delta \in \mathcal{R}^{(1)}_{\alpha,\beta}(\vec{k} \oplus \delta) \ .
\end{aligned}$$

□

Étant donné que chaque entrée $x = m \oplus k$ de T_1 dépends linéairement de m, la Proposition 5 implique l'existence de classes d'équivalence de paires (table, clé). Pour tout δ, le même ensemble de relations observées peut tout aussi bien suggérer une valeur donnée de la table T_1 si la clé secrète est $\vec{k}$, qu'une table déduite de celle-ci en XOR-ant ses indices avec δ si la clé secrète est $\vec{k} \oplus \delta$.

La conséquence de ceci est double. D'abord, exploiter les égalités de valeurs intermédiaires au point P_1 ne pourra au mieux révéler T_1 qu'à la valeur près de son premier élément (tout comme dans les attaques précédemment décrites), mais également qu'au XOR près de ses indices avec une constante δ. Ensuite, si une clé secrète $\vec{k}$ est compatible avec les observations alors toute clé secrète $\vec{k} \oplus \delta$ est aussi compatible.

Tenant compte de ces propriétés, l'algorithme décrit ci-dessus est modifié pour ne faire qu'une seule supposition sur k_0 (disons $g_0 = 0$). Le reste de l'algorithme reste inchangé.

Notons que sans les invalidations des suppositions qui révèlent leur graphe comme étant inconsistant, le nombre de suppositions à considérer augmenterait exponentiellement avec le niveau de profondeur t. On doit donc se préoccuper de savoir si ce processus de recherche en largeur entraîne effectivement un nombre prohibitif de suppositions $\vec{g}_t$ à considérer, ou bien si au contraire les suppositions incorrectes se révèlent suffisamment rapidement inconsistantes pour que l'attaque soit pratiquement réalisable.

Ici aussi, des simulations de ce processus de suppositions en largeur ont montré que la révélation de la valeur relative de T_1 (à un XOR près de ses entrées avec $T_1(0)$, et à un XOR près de ses indices avec $\delta = k_0$), est effectivement possible. Au niveau de profondeur $t = 2$, seulement quelques suppositions incorrectes (souvent moins de 20) restent compatibles, et pour $t = 3$ ou 4 seul le graphe de la supposition relative correcte (à k_0 près) reste généralement consistant. À ce point, la valeur relative de T_1 est déjà connue, mais l'attaquant peut vouloir continuer ce processus et exploiter les relations impliquant les itérations successives restantes afin de retrouver le reste des valeurs (relatives) des octets de la clé secrète.

À la fin, les valeurs relatives de T_1 et de K sont retrouvées, et l'attaquant doit seulement identifier par des techniques de type DPA ou CPA (2^{16} candidats pour T_1) quel couple $(T_1(0), k_0)$ définit leurs valeurs absolues correctes.

Nous avons expliqué comment un attaquant peut retrouver T_1 et K sans aucune connaissance *a priori* sur les données. Cette étape peut alors être suivie par l'attaque initiale de Novak afin de retrouver également T_2.

11.5 Contre-mesures

En améliorant le résultat du travail de Roman Novak, les attaques présentées dans les sections précédentes permettent de retrouver les deux tables de substitutions d'un algorithme secret. La révélation de tels détails de conception représente une menace au niveau *système*, là où une révélation de clé classique n'était une menace qu'au niveau *utilisateur*. Étant donné que l'attaque n'a besoin d'être menée qu'une fois pour toutes, le secret des spécifications de l'algorithme est directement dépendant de la protection offerte par le *plus faible produit disponible* implémentant cet algorithme GSM A3/A8.

Heureusement, il existe des contre-mesures empêchant nos attaques. La fuite des canaux auxiliaires peut être réduite par des dispositifs matériels (incluant les brouilleurs de courant ou la logique double-rail). Des modifications aléatoires dans le déroulement temporel du processus peuvent être introduites au niveau matériel (par exemple, par des interruptions aléatoire du processus, RPI) ou au niveau logiciel (par exemple, par des délais aléatoires), rendant plus difficile

la comparaison des courbes en des points spécifiés. Enfin, masquer toutes les données intermédiaires, contre-mesure classique contre les analyses statistiques, doit efficacement empêcher nos attaques, pourvu que ce masquage soit renouvelé à chaque itération. Nous attirons l'attention sur la synergie que peut fournir la combinaison de ces protections, chacune rendant plus difficile la tâche de contourner l'autre. Pourvu que de telles contre-mesures soient correctement implémentées, l'hypothèse d'observation (Hypothèse 1) n'est plus vérifiée, et l'attaque échoue.

11.6 Conclusion

Une attaque SCARE, présentée dans [Nov03], permet à un attaquant de retrouver la valeur d'une table de substitution T_2 qui fait partie des spécifications secrètes d'un algorithme GSM A3/A8 d'authentification et de génération de clés de session.

Nous avons proposé ici une interprétation de cette attaque en termes de graphes, et avons prouvé, en utilisant une modélisation par graphes aléatoires, que l'ensemble des relations qui peuvent être collectées par observation du canal auxiliaire est suffisant pour inférer la valuation complète de la table inconnue, à la valeur près de son premier élément.

Remarquant que cette première attaque nécessite la connaissance d'une autre table de substitution T_1 utilisée dans l'algorithme, ainsi que celle de la clé secrète K (Hypothèse 3), nous avons présenté une façon similaire de retrouver T_1 à partir de la seule connaissance de la clé secrète K. Nous avons alors encore amélioré cette dernière attaque afin de retrouver T_1 sans même connaître la clé secrète K, qui est également retrouvée comme sous-produit de l'attaque.

Les attaques que nous avons proposées dans ce chapitre ont été validées par simulations. Pourvu que l'hypothèse d'observation (Hypothèse 1) discutée dans [Nov03] ainsi qu'à la Section 11.2.1, et une hypothèse faible de connaissance structurelle préalable (Hypothèse 2) soient satisfaites, notre dernière attaque permet de retrouver les tables de substitution T_1 et T_2 (ainsi que la clé secrète K), sans aucune connaissance additionnelle préalable sur les données.

Nous insistons sur le fait que, contrairement aux scénarios classiques d'attaques dans lesquelles l'objectif est généralement la clé cryptographique secrète d'un utilisateur, les attaques SCARE sont des attaques *à un coup* dans le sens où elles mettent en danger les spécifications de l'algorithme une fois pour toutes. Si elles étaient publiées, une analyse poussée de ces spécifications menée par des chercheurs en cryptologie pourrait alors révéler de potentielles faiblesses de conception, qui à leur tour menaceraient tous les utilisateurs du système. La sécurité d'un système étant celle de son élément le plus faible, ce type d'attaque démontre la nécessité d'une *généralisation* des implémentations conçues avec soin. Par dessus tout, il illustre le besoin de se départir du paradigme *sécurité par l'obscurité*.

Nous espérons que cette contribution, ainsi que [Nov03] et [DLMV05], ouvriront de nouvelles perspectives pour l'analyse des canaux auxiliaires appliquée à la rétro-conception.

Chapitre 12

Généralisation d'une analyse de fautes par collisions sur l'AES

Sommaire

Nous décrivons dans ce chapitre et le suivant deux attaques par fautes qui peuvent s'appliquer à des algorithmes cryptographiques dont les spécifications ne sont pas connues. Malgré cette ignorance, ces attaques permettent de retrouver la clé secrète utilisée. Néanmoins, et contrairement au chapitre précédent, ces attaques n'ont pas pour objectif, et ne permettent pas, de retrouver de l'information sur les détails fonctionnels de l'algorithme. Après avoir retrouvé la clé, l'attaquant n'en sait pas plus au sujet des spécifications de l'algorithme qu'auparavant.

12.1 Deux techniques d'analyses de fautes

Les premières attaques par fautes, c'est-à-dire exploitant des erreurs lors d'un calcul cryptographique, furent inventées et publiées il y a une dizaine d'année par Dan Boneh, Richard DeMillo et Richard Lipton [BDL97] dans le cas du RSA, et par Eli Biham et Adi Shamir [BS97] dans le cas du DES. Ces attaques sont appelées *analyses différentielles de fautes* (DFA), car elles exploitent une différence observée sur les sorties de l'algorithme. Pour un même message d'entrée M, la fonction est évaluée une première fois sans perturber le déroulement du calcul pour donner le chiffré de référence C, puis exécutée à nouveau en injectant cette fois-ci une faute dont il résulte le chiffré fauté C'. C'est de la différence entre C et C' que l'attaquant est capable de déduire de l'information sur la clé K. Le principe de la DFA est illustré à la Figure 12.1.

Plutôt que d'obtenir de l'information sur la clé à partir de la différence produite en sortie par le chiffrement de la même entrée, il est possible de considérer le cas inverse dans lequel on chiffre deux entrées différentes M et M^*, et on apprend de l'information sur la clé par l'observation "rare" que les chiffrés de M et M^*, l'un fauté, l'autre non, coïncident. C'est parce que l'attaquant essaie de provoquer une collision entre ces deux chiffrés que cette technique est

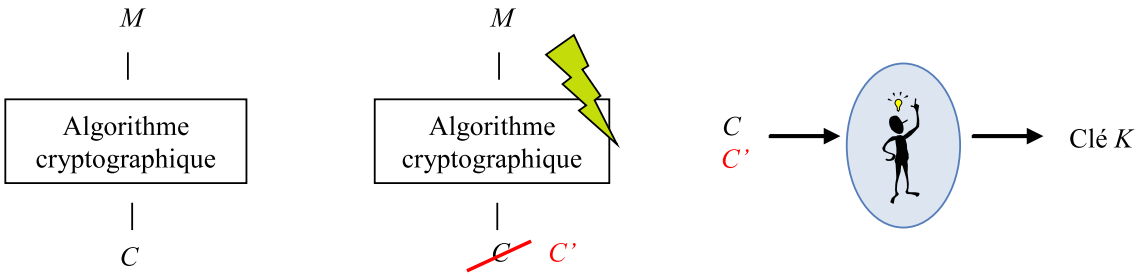


FIG. 12.1 – Principe de l'analyse différentielle de fautes.

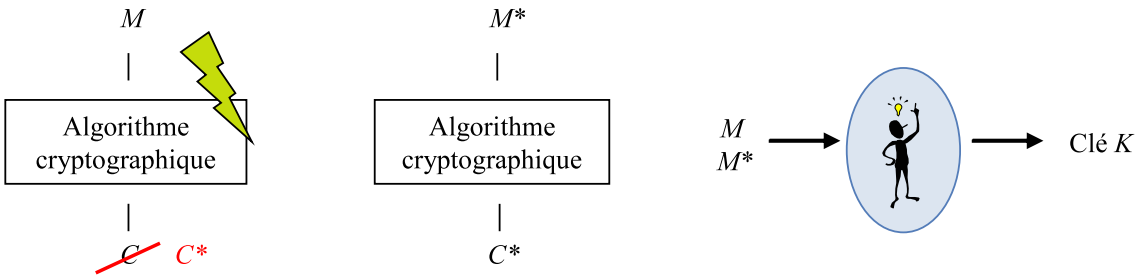


FIG. 12.2 – Principe de l'analyse de fautes par collisions.

appelée *analyse de fautes par collisions* (CFA). Parmi les premières attaques par CFA publiées, on peut citer celle de LUDGER HEMME sur les premiers tours de DES [Hem04]. On peut également mentionner l'attaque de JOHANNES BLÖMER et JEAN-PIERRE SEIFERT sur l'`AddRoundKey` initial de l'AES [BS03], quoique celle-ci puisse aussi être considérée comme une *analyse de fautes sans effet* (IFA).

Dans la pratique, un analyse de fautes par collisions est souvent menée en commençant par obtenir le chiffré fauté C^* correspondant à une entrée M, puis en recherchant un message M^* qui donne naturellement le même chiffré C^*. Le principe de la CFA est illustré à la Figure 12.2.

Notons une certaine dualité entre l'analyse différentielle de fautes et l'analyse de fautes par collisions. La DFA utilise des entrées identiques et produit des sorties différentes, alors que la CFA observe au contraire une égalité des chiffrés pour des messages différents. Également, la DFA exploite les sorties sans se préoccuper de la valeur de l'entrée, alors que pour la CFA, l'information retirée sur la clé est généralement liée à la valeur du message M^* provoquant la collision. Enfin, pour les algorithmes symétriques, la faute doit être injectée vers la fin de la fonction de chiffrement dans le cas de la DFA, alors qu'une collision ne pourra guère être obtenue qu'en visant le ou les premiers tours.

De manière moins systématique, cette dualité peut se prolonger jusqu'au modèle de faute. L'analyse différentielle pourra souvent s'accommoder d'une faute localisée de manière imprécise, et dont l'effet peut être aléatoire, alors que pour provoquer une collision, il est généralement nécessaire de cibler précisément un type d'instruction spécifique, tout en requérant un modèle de faute plus exigeant.

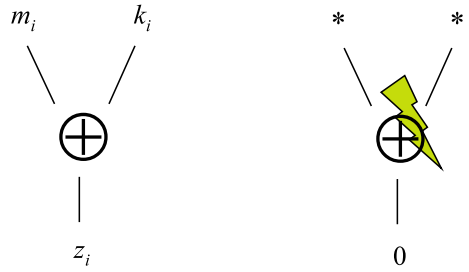


FIG. 12.3 – Le modèle de faute sur l'instruction XOR.

12.2 Une analyse de fautes par collisions sur l'AES

Nous présentons dans cette section une analyse de fautes par collisions sur l'AES. Cette attaque a déjà été considérée à la Section 10.2 du Chapitre 10, et nous allons la décrire ici plus en détails.

Nous supposons qu'un attaquant dispose d'une carte à puce contenant une clé secrète $K = (k_0, \ldots, k_{15})$ et une implémentation logicielle de l'AES. Cet attaquant est capable d'obtenir les chiffrés $C = \mathsf{AES}_K(M)$ pour autant de messages $M = (m_0, \ldots, m_{15})$ de son choix qu'il le désire. Nous supposons également qu'il est capable de provoquer des erreurs de calcul de l'AES. Le modèle de faute que nous considérons prévoit que lorsqu'un stress est appliqué pendant que le micro-processeur exécute une instruction XOR, une faute est produite dont l'effet est de rendre nul le résultat de cette opération, et cela, quelles que soient les valeurs des opérandes d'entrée. Une illustration de ce modèle de faute est proposée à la Figure 12.3.

Une façon simple d'exploiter ce modèle de faute consiste à obtenir le chiffré d'un message M quelconque tout en provoquant une faute sur l'une des 16 instructions XOR de la fonction initiale `AddRoundKey` de l'AES. Pour l'octet d'indice i correspondant à la position du XOR qui a été visé, l'exécution de l'AES se poursuit à la suite du XOR, avec la valeur 0 plutôt qu'avec la donnée normale $m_i \oplus k_i$. C'est précisément cette valeur nulle qui sera utilisée en entrée de la fonction `SubBytes` du premier tour, plutôt que la valeur normale.

La faute n'ayant d'effet que sur l'instruction XOR ciblée, il sera possible d'obtenir la même valeur du chiffré sans faire de faute, en modifiant le message uniquement à la position i, et en y recherchant la valeur m_i^* induisant une sortie naturelle du XOR égale à 0. Cette valeur m_i^* est à rechercher exhaustivement parmi les 256 possibilités, et est précisément égale à k_i puisqu'elle doit vérifier $m_i^* \oplus k_i = 0$. Un octet de la clé est ainsi révélé, et il suffit de reconduire cette attaque successivement aux 16 positions, pour obtenir la valeur complète de K. Dans le cas d'une implémentation non protégée, cette attaque est donc de faible complexité puisqu'elle ne nécessite que 16 injections de fautes, et 2^{12} chiffrements normaux à effectuer sur le dispositif attaqué.

Remarquons que dans l'AES, chacun des octets de message n'est impliqué que dans le `AddRoundKey` initial. C'est ce qui permet de reproduire un chiffré égal au chiffré fauté en reproduisant simplement une valeur identique à la sortie du XOR. Si le message avait également été utilisé ailleurs dans l'algorithme, alors la collision locale au niveau du XOR ne se serait pas propagée jusqu'à la sortie de la fonction.

Examinons maintenant le cas où l'AES attaqué implémente la contre-mesure simple qui consiste à rendre aléatoire l'ordre dans lequel sont considérés les couples (m_i, k_i) dans la fonc-

TAB. 12.1 – Complexité d'une CFA sur l'AES dans le cas d'une implémentation en ordre aléatoire.

Nombre de fautes	Nombre moyen d'octets connus	Probabilité d'une complexité au plus 2^{24}	2^{32}	2^{40}	Taille moyenne de l'espace de clé
5	4,4	0,	0,	0,	$2^{105,0}$
10	7,6	0,	0,	0,	$2^{90,0}$
15	9,9	0,015	0,095	0,319	$2^{78,1}$
20	11,6	0,239	0,535	0,809	$2^{68,0}$
25	12,8	0,609	0,858	0,968	$2^{59,3}$
30	13,7	0,851	0,968	0,996	$2^{51,7}$
35	14,3	0,953	0,994	1,000	$2^{45,1}$
40	14,8	0,987	0,999	1,000	$2^{39,2}$

tion `AddRoundKey`. Cette fois-ci, lorsqu'il provoque une faute sur l'un des XOR, l'attaquant ne connaît pas la valeur de i qui lui correspond. Il lui faudra donc rechercher le message parmi l'ensemble de tous ceux qui ne diffèrent du message initial que d'octet dont la position n'est pas spécifiée. L'attaquant peut donc commencer par établir un dictionnaire contenant cette liste de 2^{12} chiffrés. À chaque fois qu'il injecte une faute sur l'un des XOR, il recherche quelle valeur du dictionnaire est égale à celle du chiffré fauté observé. Le message qui correspond à cette entrée dans le dictionnaire permet à l'attaquant d'apprendre quelle valeur de i était utilisée lors de la faute, et de retrouver l'octet de clé k_i correspondant. Pour obtenir la clé complète, il suffit alors de répéter cette opération autant de fois qu'il est nécessaire pour que le nombre d'octets de clé connus, c'est-à-dire le nombre d'indices i couverts, soit suffisant pour pouvoir entreprendre une recherche exhaustive des octets inconnus de la clé. La Table 12.1 présente, en fonction du nombre de fautes injectées, le nombre moyen d'octets de clé révélés, les probabilités pour que la complexité de la recherche exhaustive finale soit respectivement de 2^{24}, 2^{32} ou 2^{40} clés, ainsi que la taille moyenne de cet espace de recherche. Pour 40 fautes injectées, l'espace de recherche est de taille moyenne $2^{39,2}$ clés, mais sera plus petit que 2^{24} dans presque tous les cas. Avec seulement 20 fautes, la recherche exhaustive sera de complexité moyenne égale à 2^{68}, mais inférieure ou égale à 2^{32} plus d'une fois sur deux.

Il nous semble intéressant de remarquer que les données de complexité présentées ici constituent un bel exemple de ce que la complexité moyenne d'une recherche exhaustive n'est pas nécessairement un critère pertinent pour juger de la faisabilité d'une attaque. En effet, dans le cas de 20 fautes injectées, est-il vraiment important de savoir que la complexité moyenne est de 2^{68}, c'est-à-dire que l'attaque est *a priori* infaisable, alors qu'en réalité une recherche exhaustive modérée parmi 2^{40} candidats (environ 10 jours de calcul sur un ordinateur standard) permettra de retrouver effectivement la clé dans plus de 80% des cas.

Pour résumer, la complexité de la CFA sur l'AES dans le cas d'une exécution en ordre aléatoire n'est pas significativement plus élevée que celle d'une implémentation en ordre fixe. Le nombre d'injections de fautes nécessaires est légèrement plus élevé (20 à 30 fautes au lieu de 16) et le nombre de chiffrements non fautés sur la carte attaquée est le même. Il est simplement rajouté une recherche exhaustive sur ordinateur, tout à fait abordable le plus souvent. Notons enfin que d'un point de vue expérimental, l'attaque est légèrement simplifiée dans le cas d'un `AddRoundKey` en ordre aléatoire car l'attaquant n'a pas besoin de faire varier l'instant où il injecte la faute.

12.3 Une observation triviale

Concernant l'attaque décrite ci-dessus, nous pouvons faire la remarque suivante, certes triviale, mais tout à fait importante :

Remarque 10. L'attaque CFA présentée à la Section 12.2 dans le cas de l'AES ne nécessite pas de connaître autre chose sur l'algorithme attaqué que le simple fait qu'il commence par un XOR entre le message M et la clé K.

La conséquence de cette remarque est simple. L'attaque CFA de la Section 12.2, y compris la version dans laquelle le XOR se fait en ordre aléatoire, peut s'appliquer à *tout algorithme à spécifications secrètes* dont l'attaquant ignore tout, excepté que cet algorithme commence par un XOR entre le message et la clé[48].

L'importance de cette remarque tient au fait qu'il est classique de concevoir un algorithme de chiffrement par bloc de cette manière. Cette opération linéaire initiale est un moyen simple de rendre dépendant de la clé, dès le début, le traitement qui suit appliqué au message. Ce traitement est souvent une substitution non linéaire comme dans le cas de la fonction `SubBytes` de l'AES.

Une large classe d'algorithmes est donc concernée. Parmi les algorithmes propriétaires utilisés par les opérateurs de téléphonie mobile ou de télévision à péage, ceux vérifiant cette propriété ne sont pas rares et sont donc vulnérables à cette attaque permettant de dévoiler la clé malgré le secret de (l'essentiel de) la partie cryptographique.

12.3.1 Mise en œuvre expérimentale

Nous avons réalisé cette attaque, dans des conditions de laboratoire, dès l'automne 2001 dans le département des technologies de la sécurité de *Gemplus*.

La mise en œuvre avait été facilitée par une caractérisation préalable d'un certain microprocesseur utilisé couramment à cette époque dans les cartes à puce. Cette caractérisation visait à étudier de manière systématique la vulnérabilité de ce composant à l'injection de faute par lumière blanche, lors de l'exécution de diverses instructions arithmétiques ou logiques. Entre autres résultats, cela a permis d'établir la validité du modèle de faute présenté à la Figure 12.3.

Suite à cette caractérisation, nous avons demandé à un développeur innocent, de nous fournir une carte de test sur laquelle était implémenté un des algorithmes GSM commençant par un XOR entre le message et la clé. Nous ne connaissions, ni l'algorithme, ni la valeur de la clé.

Dans notre expérience, la lumière blanche était émise par un flash. Elle illuminait sur la surface du composant, préalablement mise au jour par abrasion chimique, un cercle d'environ 1 mm de diamètre. Il était donc possible de choisir assez précisément la zone que nous voulions perturber. Le déclenchement du flash était synchronisé avec le programme exécuté dans la carte et nous pouvions choisir l'instant d'illumination avec une précision d'un cycle d'horloge.

Une expérimentation similaire sur une carte implémentant l'AES avec le même composant nous avait permis, quelques jours plus tôt, de nous familiariser avec certains aspects pratiques liés à l'expérience. L'attaque a essentiellement consisté en un balayage, cycle d'horloge par cycle d'horloge, d'une zone que nous soupçonnions contenir la boucle des 16 instructions XOR entre les octets du messages et de la clé. Nous avons recueilli toute une liste de valeurs de sorties différentes du chiffré de référence. Les valeurs communes à cette liste et à un dictionnaire préalablement établi nous ont permis de retrouver facilement la clé.

[48] À l'exception toutefois de ceux pour lesquels le message est impliqué également à un autre endroit que dans le XOR initial. Ceci exclut donc, notamment, toutes les conceptions à base de réseau de Feistel.

Chapitre 13

Les encodages externes secrets ne protègent pas des analyses de fautes transitoires

Sommaire

Tout comme celle du chapitre précédent, l'attaque que nous décrivons dans ce chapitre s'applique à une large classe d'algorithmes, et permet de retrouver une clé malgré l'ignorance de la fonction cryptographique qui l'utilise. Cette contribution a fait l'objet d'une publication à CHES '07 [Cla07a].

Contrairement au principe de KERCKHOFFS, de nombreuses applications de la cryptographie actuelle continuent d'adopter le paradigme de la *sécurité par l'obscurité*. Néanmoins, et afin de profiter de sa sécurité empirique ou prouvée, certaines réalisations se basent sur un algorithme cryptographique bien connu et largement utilisé. En particulier, une conception possible consiste à obfusquer un chiffrement par bloc standard $\mathtt{E}$ en l'encadrant par deux encodages externes *secrets* $\mathtt{P}_1$ et $\mathtt{P}_2$ (bijections), d'où résulte l'algorithme propriétaire $\mathtt{E'} = \mathtt{P}_2 \circ \mathtt{E} \circ \mathtt{P}_1$.

Puisque les entrées et les sorties de la fonction sous-jacente $\mathtt{E}$ ne sont pas connues d'un potentiel attaquant, une telle construction revendique généralement l'avantage d'empêcher tout type d'analyse de fautes transitoires qui pourrait s'appliquer sur la fonction interne $\mathtt{E}$. Dans ce chapitre, nous montrons que ce dernier argument n'est pas correct, en exhibant une attaque révélant la clé qui s'applique à la classe entière des DES ou Triple-DES munis d'un encodage externe. De plus, notre attaque reste applicable même en présence de la contre-mesure anti-fautes classique qui consiste à exécuter l'algorithme deux fois et à ne retourner le chiffré que si les deux résultats coïncident.

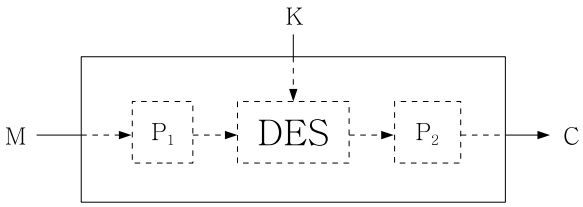


FIG. 13.1 – Un DES obfusqué par des encodages externes $\mathtt{P}_1$ et $\mathtt{P}_2$.

13.1 Introduction

Contrairement au principe de KERCKHOFFS, de nombreuses applications de la cryptographie actuelle continuent d'adopter le paradigme de la *sécurité par l'obscurité*. Pour les applications publiques et civiles, cela est notamment le cas dans les domaines du GSM et de la télévision à péage où les spécifications des fonction d'authentification et de chiffrement sont souvent gardées secrètes. Un avantage couramment revendiqué de dissimuler ainsi les détails des fonctions cryptographiques est de se protéger contre les attaques physiques connues telles que les analyses de canaux auxiliaires (SPA, DPA, CPA,...) [KJJ99, BCO04], ou les analyses de fautes (DFA, CFA,...) [BS97, BDL97, Hem04, ACT06], qui sinon seraient utilisées pour révéler la clé secrète de l'utilisateur. Le résultat principal de notre contribution est de réfuter partiellement cette croyance. Plus précisément, nous nous concentrons sur une manière particulière de concevoir un tel algorithme propriétaire, qui consiste à encadrer un chiffrement par bloc $\mathtt{E}$ bien connu et largement utilisé (comme par exemple le DES ou l'AES), par deux encodages externes *secrets* $\mathtt{P}_1$ et $\mathtt{P}_2$ (bijections sur les espaces des entrées et des sorties), d'où résulte un nouveau chiffrement par bloc obfusqué et secret $\mathtt{E'} = \mathtt{P}_2 \circ \mathtt{E} \circ \mathtt{P}_1$. La motivation pour une telle stratégie de conception est double. D'abord, il semble raisonnable de baser la construction sur un chiffrement par bloc $\mathtt{E}$ bien connu, afin d'hériter sa robustesse cryptographique prouvée ou empirique. Ensuite, les deux encodages secrets $\mathtt{P}_1$ et $\mathtt{P}_2$ assurent que les entrées et les sorties de $\mathtt{E}$ ne peuvent pas être connues de l'attaquant, de sorte que les attaques physiques nécessitant cette connaissance ne devraient pas être faisables.

Dans ce chapitre, nous présentons une attaque par fautes, permettant la révélation de la clé, et qui s'applique à ce type de construction lorsque le chiffrement par bloc interne $\mathtt{E}$ est le DES ou le Triple-DES. Notre attaque fonctionne pour tous $\mathtt{P}_1$ et $\mathtt{P}_2$, si bien qu'elle menace la classe entière des DES (ou Triple-DES) munis d'un encodage externe[49] (voir la Figure 13.1).

Nous présentons à la Section 13.2 l'état de l'art des travaux liés aux analyses par fautes des fonctions cryptographiques secrètes, ainsi que notre modèle de menace et les conditions nécessaires au succès de notre attaque. La Section 13.3 constitue le cœur de notre contribution et propose deux variantes de notre attaque : d'abord, nous présentons et expliquons une version basique qui en illustre les principes majeurs ; ensuite, nous décrivons une version améliorée de notre attaque et en présentons des résultats de simulation. Des contre-mesures possibles sont alors discutées dans la section suivante, et la Section 13.5 conclut ce travail tout en proposant de possibles axes de recherche future.

[49] Comme cas particulier où $\mathtt{P}_1$ et $\mathtt{P}_2$ sont des masquages par XOR avec des clés externes, notre attaque s'applique notamment à la construction DESX [KR96] et permet de retrouver sa clé interne secrète.

13.2 Préliminaires

Dan Boneh, Richard DeMillo et Richard Lipton [BDL97] ont les premiers introduit l'utilisation d'erreurs de calcul transitoires comme moyen d'extraire les clés secrètes d'algorithmes cryptographiques. Leur attaque s'applique au RSA en mode CRT et a été suivie dans la foulée par des résultats similaires s'appliquant à l'algorithme DES [BS97] et à d'autres fonctions. Ces méthodes nécessitent la connaissance de l'algorithme cryptographique attaqué.

Dans [BS97], Eli Biham et Adi Shamir présentent également trois autres méthodes d'attaques par fautes s'appliquant à des *crypto-systèmes inconnus*. L'une d'elles suppose que l'attaquant est capable de sectionner une piste ou de détruire une cellule mémoire de sorte qu'un bit d'un certain registre du DES soit constamment fixé à la valeur 0. Une deuxième méthode repose sur l'hypothèse qu'il est possible de baisser, progressivement et de manière permanente, les bits de la clé stockée en mémoire non volatile. De plus, pour permettre de retrouver la clé lorsque l'algorithme est inconnu, l'attaquant doit avoir la possibilité de changer la valeur de la clé secrète stockée dans le dispositif attaqué. Plus tard, cette technique a été améliorée et étendue par Pascal Paillier [Pai99]. Étant donné qu'elles requièrent des fautes permanentes, ces deux méthodes nous semblent assez difficilement réalisables en pratique. De plus, détruire définitivement le dispositif attaqué peut être indésirable. La troisième méthode proposée dans [BS97] pour retrouver la clé d'un crypto-système inconnu utilise le modèle de faute classique de la DFA, c'est-à-dire des fautes transitoires et aléatoires, pour retrouver très astucieusement les détails d'une fonction cryptographique "de type DES" inconnue. Cette attaque permet également de révéler la clé utilisée pourvu que l'attaquant puisse avoir accès à plusieurs dispositifs du même type contenant des clés différentes.

À notre connaissance, il n'a jamais été publié d'attaque par fautes transitoires sur des crypto-systèmes inconnus qui permet de retrouver la clé secrète sans avoir besoin d'un dispositif contenant d'autres valeurs de clé et sans rien révéler sur le crypto-système considéré. L'attaque que nous proposons atteint précisément cet objectif sous les hypothèses suivantes :

La cible est une implémentation logicielle classique du DES sur une architecture à 8 bits. Nous supposons également que l'attaquant est capable de contrôler précisément quelle instruction est exécutée à l'instant où il injecte la faute.

Concernant le modèle de faute, nous supposons qu'une faute injectée pendant l'exécution d'un XOR entre deux opérandes de 8 bits provoque un résultat égal à zéro[50] quelles que soient les valeurs des opérandes d'entrée. Notons que ce modèle de faute est réaliste car nous avons identifié des composants vulnérables à ce type de fautes et sur lesquels nous avons réalisé concrètement une attaque reposant sur ce modèle (voir l'attaque décrite au Chapitre 12).

Enfin, l'attaquant est supposé contrôler l'entrée donnée à la fonction de chiffrement `E'`, et connaître sa sortie.[51]

Comparée aux analyses de fautes sur les crypto-systèmes publics, notre attaque nécessite un grand nombre (plusieurs milliers) d'injections de fautes. Nous voyons cet inconvénient comme un juste prix à payer pour obtenir cette faculté "magique" de pouvoir retrouver la clé sans considération des deux encodages externes secrets P_1 and P_2.

[50] Notons que notre attaque fonctionne aussi bien si la sortie du XOR fauté est supposée valoir n'importe quelle constante connue arbitraire à la place de zéro.

[51] Ces hypothèses peuvent être assouplies. Il est seulement nécessaire de pouvoir rejouer plusieurs fois différentes entrées arbitraires, et de pouvoir détecter lorsque deux sorties sont identiques.

13.3 Analyse de fautes sans effet

13.3.1 L'injection de fautes comme outil de sondage

Le principe de notre attaque, décrite dans les Sections 13.3.2 et 13.3.3, considère la capacité à injecter des fautes comme un outil de sondage de n'importe quel XOR à n'importe quel stade de l'algorithme. Plus précisément, si un attaquant cible une instruction XOR particulière, alors il est capable de détecter si la sortie de cette instruction est égale à zéro ou non. Pour une entrée arbitraire, si la sortie de l'algorithme lorsqu'une faute est injectée durant le XOR ciblé est la même que celle d'une exécution normale, alors cela indique que la valeur naturelle du résultat du XOR est zéro[52]. Une identité de sorties survient lorsque la faute injectée n'a *aucun effet* sur l'instruction ciblée et son résultat. Une certaine information sur une valeur intermédiaire est ainsi obtenue en observant deux sorties identiques de l'algorithme. Cet évènement étant le plus informatif exploité dans notre attaque, nous appelons ce type d'attaque une *analyse de fautes sans effet* (IFA). Bien qu'elle soit assez similaire aux *analyses des erreurs sûres* (en anglais, *Safe-Error Analysis*) [JQYY02, YJ00, YKLM02], l'IFA est légèrement différente en ce que la faute vise une véritable instruction, dont la sortie est éventuellement non modifiée, plutôt qu'une instruction fictive comme c'est le cas par exemple des multiplications arithmétiquement neutres (qu'elles soient perturbées ou non) de l'implémentation *élever au carré et multiplier systématiquement* du RSA. Dans le cas de l'IFA, une sortie inchangée de l'algorithme résulte d'une condition liée à la *donnée*, alors que ce même évènement est spécifique à l'*algorithme* dans le cas de l'analyse des erreurs sûres.

Dans le contexte de l'attaque présentée dans ce chapitre, pour tout texte clair, l'IFA permet à un attaquant de détecter si la sortie de n'importe quel XOR arbitraire du DES interne est zéro.

13.3.2 L'attaque basique

Nous renvoyons le lecteur à la Section 1.2.1 ou à [NBS77] pour une description complète de l'algorithme DES. Néanmoins, et avant de décrire l'attaque en détail, nous rappelons au lecteur que la dérivation des clés de tour du DES est structurée de telle sorte que la clé peut être considérée comme partitionnée en deux moitiés de clé de 28 bits, que nous notons respectivement K^A et K^B. À chaque tour, les 24 bits de clé impliqués dans les S-Box 1 à 4 sont un sous-ensemble de K^A, alors que les 24 bits impliqués dans les S-Box 5 à 8 appartiennent à K^B. Bien que notre attaque ne soit *pas* dépendante de cette propriété, nous l'utiliserons avec avantage pour simplifier le calcul d'exploitation des fautes en considérant séparément les deux espaces de moitiés de clés.

Nous supposons une implémentation naturelle du DES sur une architecture à 8 bits. Dans cette implémentation typique, il y a 12 opérations XOR par tour. Comme présenté sur la Figure 13.2, pour chaque tour $h = 2, \ldots, 16$, nous considérons deux groupes d'instructions XOR. Le premier groupe est constitué de quatre instructions, que nous appelons `xor_left`, exécutées à la fin du tour $(h-1)$, et qui calculent les quatre octets $(r_1, \ldots, r_4)$ de la valeur de 32 bits R_h qui entre dans le tour h. Alors, $(r_1, \ldots, r_4)$ est étendu en huit valeurs de 6 bits $(s_1, \ldots, s_8)$ qui sont XOR-ées avec la clé de tour $K_h = (k_1, \ldots, k_8)$, à travers huit instructions que nous appelons `xor_key`, pour produire les entrées $(x_1, \ldots, x_8)$ des S-Box. Chaque sortie de S-Box de 4 bits est calculée comme $y_j = S_j(x_j)$.

L'idée centrale de cette version basique de l'attaque est d'inférer de l'information sur la clé à partir de couples de fautes sans effet sur deux exécutions différentes avec la même entrée. Bien

[52] Ou au moins que cette valeur est équivalente à zéro à travers le restant de l'algorithme. Ce commentaire sera clarifié dans l'exemple donné à la Section 13.3.2.

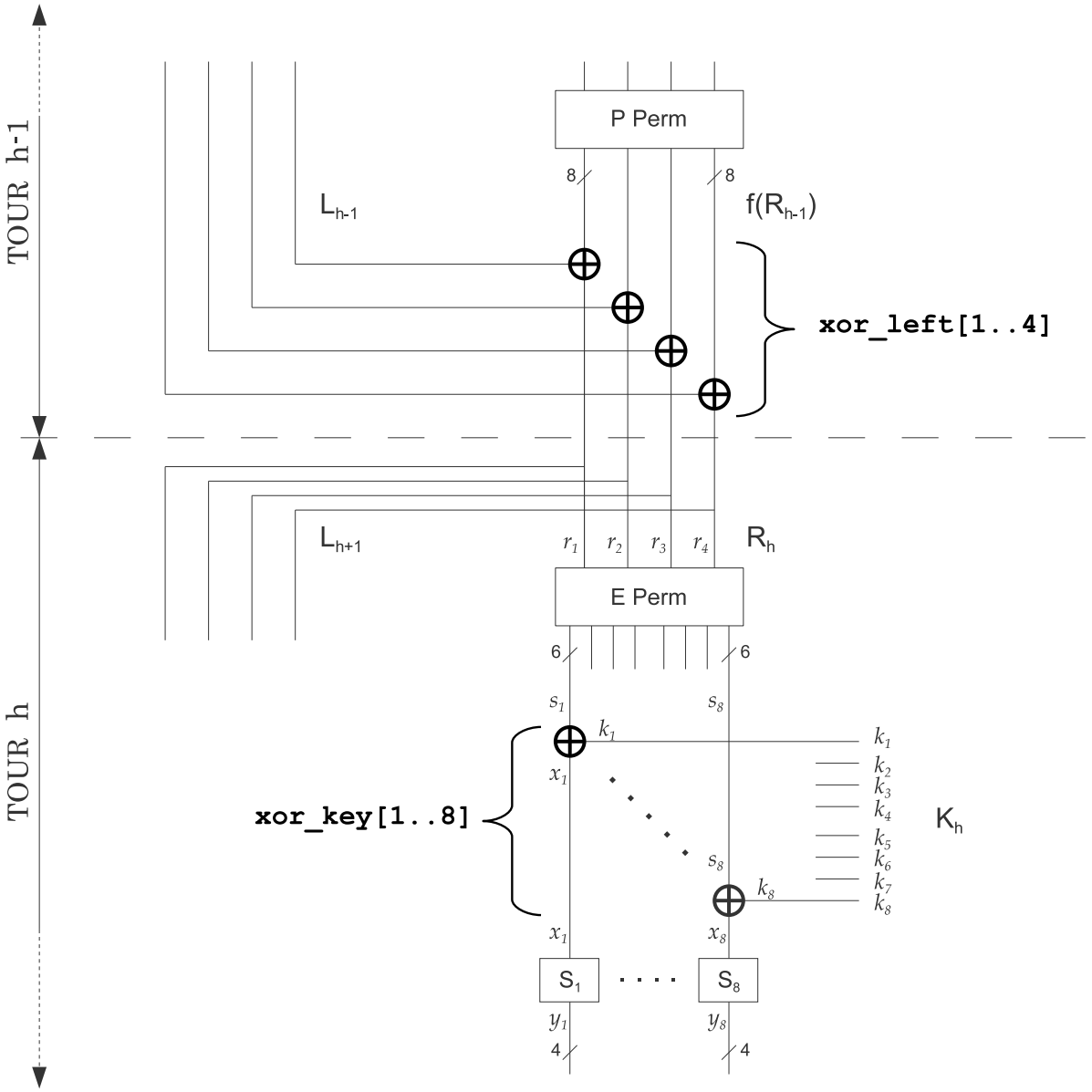


FIG. 13.2 – Les 12 instructions XOR de chaque tour ciblées par l'attaque.

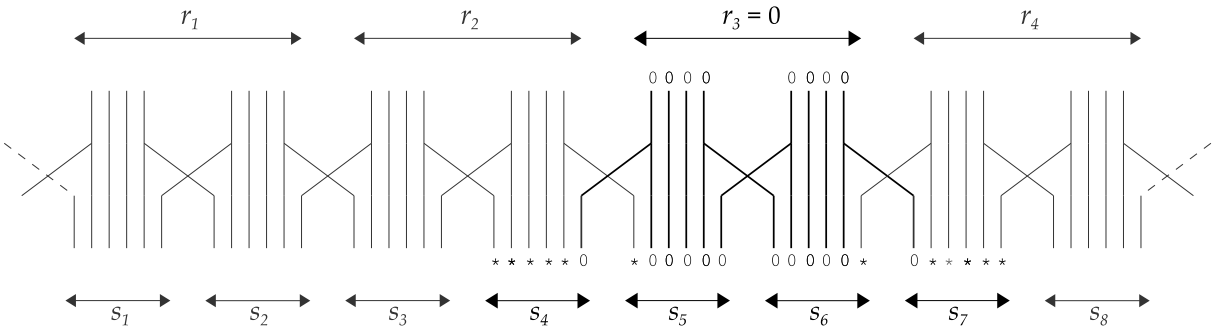


FIG. 13.3 – Un octet égal à zéro à travers la permutation expansive.

que de l'information sur deux valeurs intermédiaires d'un calcul soit ainsi obtenue, nous attirons l'attention sur le fait que notre attaque ne requiert pas la capacité d'injecter des fautes multiples sur la même exécution.

Supposons d'abord que pour un certain message M, une faute injectée durant `xor_left[`i`]` (pour $i \in \{1, \ldots, 4\}$) au tour $(h-1)$ s'avère être sans effet. Cela implique que l'octet de sortie correspondant r_i est égal à zéro. Donc, 8 des 32 bits de R_h sont connus pour valoir 0. La permutation qui suit les étend en 12 bits qui sont impliqués dans quatre S-Box adjacentes[53] au tour h. Supposons maintenant que pour une autre exécution avec le même clair M, une faute sur `xor_key[`j`]` (pour $j \in \{2i-1, 2i\}$) au tour h se montre sans effet elle aussi. Dans ce contexte, nous montrons que de l'information intéressante peut être inférée sur la valeur de 6 bits k_j. Ceci est le principe de base de notre attaque. Nous donnons maintenant un exemple pour illustrer ce raisonnement :

Exemple : Pour un certain M, l'observation d'une faute sans effet sur `xor_left[3]` au tour $(h-1)$ nous apprend que $r_3 = 0$. La Figure 13.3 montre que lorsque $r_3 = 0$, les entrées de 6 bits s_4 à s_7 de `xor_key[4]` à `xor_key[7]` au tour suivant se conforment à $(*,*,*,*,*,0)$, $(*,0,0,0,0,0)$, $(0,0,0,0,0,*)$ et $(0,*,*,*,*,*)$ respectivement. Supposons que, pour le même M, une faute sur `xor_key[5]` au tour h apparaisse également être sans effet. On pourrait rapidement en déduire que la sortie x_5 de `xor_key[5]` est égale à 0. Mais en réalité, cela implique plutôt que x_5 appartient à l'ensemble $\mathcal{A}_5 = \{(0,0,0,0,0,0),(0,0,0,1,0,1),(1,0,0,0,1,0),(1,0,1,1,0,1)\}$ formé des quatre antécédents de $S_5(0)$ par S_5. Cela est dû à la propriété de non-injectivité des S-Box, qui fait que chaque sortie de 4 bits possède exactement quatre antécédents[54]. Nous pouvons maintenant en déduire que $k_5 = x_5 \oplus s_5 \in \mathcal{A}_5 \oplus (*,0,0,0,0,0)$, ce qui conduit à 8 valeurs possibles pour k_5, correspondant à 3 bits d'information retrouvée sur la clé K.

Avec le même raisonnement, une égalité des sorties lorsque l'on faute `xor_key[6]` impliquerait que $k_6 \in \mathcal{A}_6 \oplus (0,0,0,0,0,*)$, révélant 3 autres bits d'information sur la clé. Notons que pour les deux autres S-Box (S_4 et S_7), il n'est pas possible de déterminer les valeurs, ni du bit le plus à droite de k_4, ni du bit le plus à gauche de k_7.[55]

[53] On considère les huit S-Box comme formant un anneau. Par exemple, les S-Box 8, 1, 2 et 3 sont adjacentes.

[54] Remarquons que MEHDI-LAURENT AKKAR a également présenté dans sa thèse [Akk04] une analyse de faute par collisions qui exploite astucieusement la non-injectivité des S-Box. Cette attaque s'applique à un DES standard (non muni d'un encodage externe), et est applicable, tout comme la nôtre, en présence de la contre-mesure qui consiste à vérifier le résultat du calcul.

[55] Cela est dû au fait que, par conception de chaque S-Box S_j, chaque bit latéral est systématiquement représenté avec les deux valeurs 0 et 1 parmi $\mathcal{A}_j$.

Cet exemple nous montre le type d'évènement dont nous tirons parti pour obtenir de l'information sur une sous-clé de tour. Nous le formalisons par la définition suivante :

Définition 9 (Évènement gagnant)**.** *Nous appelons* évènement gagnant *au locus* (h, i, j) *une paire d'observations, pour le même clair, de deux fautes sans effet : une sur* `xor_left[i]` *au tour* $(h-1)$*, et une autre sur* `xor_key[j]` *au tour* h*, où* $j \in \{2i-1, 2i\}$*.*

Les évènements gagnants tels que celui au locus $(h, 3, 5)$ décrit dans l'exemple précédent sont les évènements clés exploités dans cette attaque.

Obtenir un évènement gagnant à un certain locus dépend évidemment du clair. En effet, les valeurs de $L_{h-1}[i]$ et $R_{h-1}[i]$ qui gouvernent le caractère "avec" ou "sans" effet d'une faute sur `xor_left[`i`]`, ainsi que les valeurs des bits "$*$" de s_j qui influencent le caractère "avec" ou "sans" effet d'une faute sur `xor_key[`j`]`, dépendent tous du clair. Ainsi, si un évènement gagnant à un certain locus n'est pas obtenu pour un clair donné, il peut très bien être obtenu pour un autre. Cependant, pour qu'un évènement gagnant soit obtenu, les bits de clé correspondant aux cinq bits "0" de s_j doivent être égaux à leur contrepartie dans un des éléments de $\mathcal{A}_j$. En conséquence, étant donné une clé K, il existe certains locus auxquels aucun évènement gagnant ne peut être obtenu quel que soit le clair, et d'autres auxquels des évènements gagnants peuvent être obtenus pour certains clairs.

Définition 10 (Locus gagnable)**.** *Étant donnée une clé* K*, nous disons que* (h, i, j) *est un* locus gagnable *si les cinq bits de* k_j *aux positions "0" (celles où une faute sans effet sur* `xor_left[i]` *au tour* $(h-1)$ *implique une valeur de bit de* s_j *égale à 0), sont égaux à leur contrepartie dans un des quatre éléments de* $\mathcal{A}_j$*.*

Exemple : Pour $K =$ `CD3ABC5876AC062B`, le locus $(7, 3, 5)$ est gagnable car la sous-clé k_5 entrant dans la S-Box 5 au tour 7 est égale à $(1, 0, 0, 1, 0, 1)$ dont les cinq bits les plus à droite sont égaux à ceux de l'élément $(0, 0, 0, 1, 0, 1)$ de $\mathcal{A}_5$.

La probabilité (sur toutes les clés) pour un locus donné d'être gagnable est de $4 \times 2^{-5} = 0,125$. Il existe $8*(16-1) = 120$ locus intéressants le long du DES (le premier tour n'est pas exploitable), donc, dans le modèle simplifié où tous les K_h sont considérés indépendants, on s'attend à 15 locus par clé en moyenne. Un comptage par simulations sur 27 000 clés générées aléatoirement donne un nombre de locus gagnables distribué comme montré sur la Figure 13.4, avec une moyenne de 14,986.

Pour chaque locus gagnable, et dès qu'un évènement gagnant est obtenu, l'espace des clés peut être réduit en tenant compte des contraintes expliquées précédemment. L'entropie résiduelle optimale, c'est-à-dire qui résulte de l'exploitation de tous les locus gagnables, est distribuée comme indiqué sur la Figure 13.5. Les percentiles de cette distribution pour les fréquences (0,10 ; 0,50 ; 0,90) sont (14,17 ; 21,32 ; 29,98), signifiant que pour une clé sur deux, l'exploitation complète de tous les locus gagnables réduit l'espace des clés de 2^{56} à moins de $2^{21,32}$ clés.

Nous présentons à la Figure 13.6 une procédure possible pour mener l'attaque décrite dans cette section. La condition d'arrêt est laissée au choix de l'attaquant : elle peut impliquer le nombre de fautes déjà injectées, l'entropie résiduelle courante de l'espace des clés, ou toute autre considération.

Une simulation de cette attaque d'après cette procédure nous a permis de quantifier le nombre de fautes nécessaires. À partir de 27 000 simulations, le nombre de fautes nécessaires pour gagner tous les locus gagnables est distribué comme indiqué à la Figure 13.7. Avec 100 000 fautes, tous les locus gagnables sont gagnés dans 54,5 % des cas, et tous sauf éventuellement un sont gagnés dans 87,3% des cas. Lorsque le nombre de locus gagnés augmente, la probabilité d'en gagner un

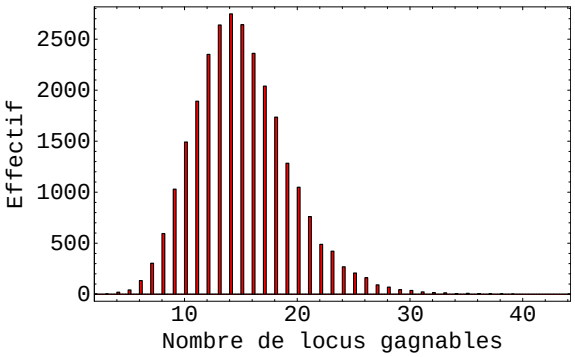


FIG. 13.4 – Nombre de locus gagnables par clé.

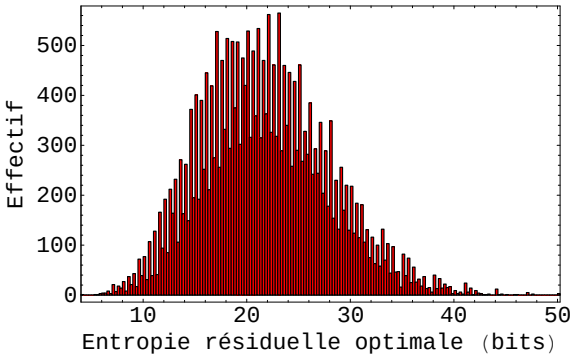


FIG. 13.5 – Entropie résiduelle optimale par clé après exploitation de tous les locus gagnables.

Algorithme 13.1 L'attaque basique

tant que la condition d'arrêt n'est pas vérifiée
    choisir aléatoirement un clair M
    obtenir le chiffré $C \leftarrow \mathtt{E}(M, K)$
    pour $h \leftarrow 2$ à 16
        pour $i \leftarrow 1$ à 4
            obtenir $C^* \leftarrow \mathtt{E}(M, K)$ avec faute sur `xor_left[`i`]` au tour $h - 1$
            si $(C^* = C)$ **alors**
                pour $j \leftarrow 2i - 1$ à $2i$
                    obtenir $C^* \leftarrow \mathtt{E}(M, K)$ avec faute sur `xor_key[`j`]` au tour h
                    si $(C^* = C)$ **alors**
                        utiliser l'évènement gagnant (h, i, j) pour réduire l'espace de demi-clés pertinent (K^A ou K^B)
                    fin si
                fin pour
            fin si
        fin pour
    fin pour
fin tant que
à l'aide d'un dispositif ouvert permettant de simuler l'algorithme, rechercher exhaustivement $K = (K^A, K^B)$ parmi les candidats restant

FIG. 13.6 – Version de base de l'attaque.

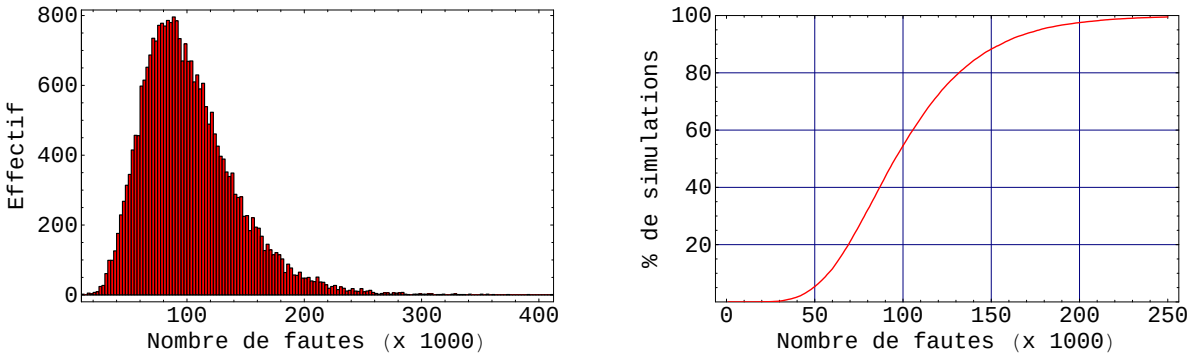


FIG. 13.7 – Nombre de fautes nécessaires pour gagner tous les locus gagnables.

nouveau décroît assez rapidement. Cela suggère qu'il n'y a aucun intérêt à continuer à produire des fautes pendant trop longtemps. Les entropies résiduelles médianes après 50 000 et 100 000 fautes sont respectivement de 26,49 et 22,32 bits.

13.3.3 Une version améliorée de l'attaque

Essentiellement, l'attaque décrite à la Section 13.3.2 élimine les clés qui sont incompatibles avec un quelconque évènement gagnant observé. Dans cette section, nous en présentons une évolution qui apporte deux améliorations.

Tout d'abord, nous étendons les types d'évènements qui sont exploités. Par exemple, quand un évènement gagnant au locus $(h, 3, 5)$ est observé, et que pour le même M l'attaquant sait que $r_2 = 0$ (par exemple, dans le cas d'une faute sans effet sur `xor_left[2]` au tour $(h-1)$), alors les 6 bits de k_5 sont contraints au lieu de seulement 5. Cela fournit une information supplémentaire sur la clé qui n'était pas exploitée dans l'attaque basique. Comme autre exemple, supposons qu'une faute sans effet soit observée sur `xor_left[3]` au tour $(h-1)$, mais pas sur `xor_key[5]` au tour h. Même si cela ne constitue pas un évènement gagnant au locus $(h, 3, 5)$, on peut malgré tout en inférer de l'information sur k_5, précisément le fait que k_5 *n'appartient pas* à $\mathcal{A}_5 \oplus (*, 0, 0, 0, 0, 0)$. Ici aussi, cet évènement informatif n'était pas considéré dans l'attaque basique.

La seconde amélioration consiste à calculer pour chaque clé sa probabilité *a posteriori* conditionnée par les observations.

L'effet conjugué de ces deux améliorations est que, non seulement l'espace des clés compatibles est encore plus réduit, mais également que sa recherche exhaustive est accélérée en testant les clés par ordre décroissant de leur probabilité.

Définition 11 (Vecteur de positions sans effet)**.** *Nous appelons* vecteur de positions sans effet *au tour h, noté $\mathbf{e} = (\mathbf{e}_{left}, \mathbf{e}_{key})$, le vecteur booléen $\mathbf{e}_{left}$ du caractère sans effet de fautes injectées sur `xor_left[1]` à `xor_left[4]` au tour $(h-1)$, ainsi que le vecteur booléen $\mathbf{e}_{key}$ du caractère sans effet de fautes injectées sur `xor_key[1]` à `xor_key[8]` au tour h.*

(Les fautes produites pour l'observation de leur caractère "avec" ou "sans" effet s'entendent à texte clair M constant.)

Par exemple, l'évènement gagnant au locus $(h, 3, 5)$ décrit à la Section 13.3.2 peut avoir été produit par le vecteur de positions sans effet $(\mathbf{e}_{left}, \mathbf{e}_{key})$, où $\mathbf{e}_{left} = (0, 0, 1, 0)$, et $\mathbf{e}_{key} = (0, 0, 0, 0, 1, 0, 0, 0)$.

Algorithme 13.2 L'attaque améliorée

1. initialiser à 1 les probabilités de chaque K^A et chaque K^B
2. **tant que** la condition d'arrêt n'est pas vérifiée
3. choisir aléatoirement un clair M
4. obtenir le chiffré $C \leftarrow \mathtt{E}(M, K)$
5. **pour** $h \leftarrow 2$ à 16
6. **pour** $i \leftarrow 1$ à 4
7. obtenir $C^* \leftarrow \mathtt{E}(M, K)$ avec faute sur `xor_left[`i`]` au tour $h-1$
8. $\mathbf{e}_{left}[i] \leftarrow (C^* \stackrel{?}{=} C)$
9. **fin pour**
10. **si** ($\mathbf{e}_{left}[1] = \mathtt{vrai}$) ou ($\mathbf{e}_{left}[2] = \mathtt{vrai}$) **alors**
11. **pour** $j \leftarrow 1$ à 4
12. obtenir $C^* \leftarrow \mathtt{E}(M, K)$ avec faute sur `xor_key[`j`]` au tour h
13. $\mathbf{e}^A_{key}[j] \leftarrow (C^* \stackrel{?}{=} C)$
14. **fin pour**
15. **pour** toutes les K^A telles que proba$(K^A) > 0$
16. proba$(K^A) \leftarrow$ proba$(K^A) \cdot \mathcal{P}\left((\mathbf{e}_{left}, \mathbf{e}^A_{key}) \mid K^A \right)$
17. **fin pour**
18. **fin si**
19. **si** ($\mathbf{e}_{left}[3] = \mathtt{vrai}$) ou ($\mathbf{e}_{left}[4] = \mathtt{vrai}$) **alors**
20. **pour** $j \leftarrow 1$ à 4
21. obtenir $C^* \leftarrow \mathtt{E}(M, K)$ avec faute sur `xor_key[`$j+4$`]` au tour h
22. $\mathbf{e}^B_{key}[j] \leftarrow (C^* \stackrel{?}{=} C)$
23. **fin pour**
24. **pour** toutes les K^B telles que proba$(K^B) > 0$
25. proba$(K^B) \leftarrow$ proba$(K^B) \cdot \mathcal{P}\left((\mathbf{e}_{left}, \mathbf{e}^B_{key}) \mid K^B \right)$
26. **fin pour**
27. **fin si**
28. **fin pour**
29. **fin tant que**
30. à l'aide d'un dispositif ouvert permettant de simuler l'algorithme, rechercher exhaustivement $K = (K^A, K^B)$ par ordre de probabilité décroissante

FIG. 13.8 – Version améliorée de l'attaque.

Pour tout $\sigma \in \{A, B\}$, notons $\mathbf{e}_{key}^{\sigma}$ la partie de $\mathbf{e}_{key}$ liée aux quatre S-Box impliquant K^{σ} (de sorte que $\mathbf{e}_{key} = (\mathbf{e}_{key}^{A}, \mathbf{e}_{key}^{B})$), et $\mathbf{e}^{\sigma}$ le couple $(\mathbf{e}_{left}, \mathbf{e}_{key}^{\sigma})$.

Tout vecteur observé de positions sans effet peut être utilisé pour calculer la probabilité *a posteriori* de chaque demi-clé K^{σ} grâce à la formule de Bayes :

$$\mathcal{P}\left(K^{\sigma} \mid \mathbf{e}^{\sigma}\right) = \mathcal{P}\left(\mathbf{e}^{\sigma} \mid K^{\sigma}\right) \cdot \frac{\mathcal{P}\left(K^{\sigma}\right)}{\mathcal{P}\left(\mathbf{e}^{\sigma}\right)} \,. \tag{13.1}$$

De l'Equation (13.1), nous dérivons une forme récursive permettant de mettre à jour la probabilité *a posteriori* d'une clé, à partir d'une nouvelle observation d'un vecteur de positions sans effet :

$$\mathcal{P}\left(K^{\sigma} \mid (\mathbf{e}_1^{\sigma}, \ldots, \mathbf{e}_n^{\sigma})\right) = \frac{\mathcal{P}\left(\mathbf{e}_n^{\sigma} \mid K^{\sigma}\right)}{\mathcal{P}\left(\mathbf{e}_n^{\sigma}\right)} \cdot \mathcal{P}\left(K^{\sigma} \mid (\mathbf{e}_1^{\sigma}, \ldots, \mathbf{e}_{n-1}^{\sigma})\right) \,. \tag{13.2}$$

Notons qu'évaluer le dénominateur $\mathcal{P}\left(\mathbf{e}_n^{\sigma}\right)$ n'est pas nécessaire puisqu'il est indépendant de la clé. Dans le but de comparer les probabilités des clés les unes aux autres, omettre ce dénominateur ne fera qu'affecter ces probabilités par un même facteur multiplicatif. Ainsi, quand on considère une nouvelle observation $\mathbf{e}^{\sigma}$, le processus de mise à jour des probabilités des demi-clés se réduit à multiplier la probabilité (non normalisée) de chaque demi-clé K^{σ} par $\mathcal{P}\left(\mathbf{e}^{\sigma} \mid K^{\sigma}\right)$.

En supposant un comportement aléatoire de R_h, évaluer $\mathcal{P}\left(\mathbf{e}^{\sigma} \mid K^{\sigma}\right)$ se fait en comptant le nombre d'entrées de tour compatibles avec les observations. En effet, nous avons :

$$\mathcal{P}\left(\mathbf{e}^{\sigma} \mid K^{\sigma}\right) = \frac{\#\{R_h : \mathbf{e}_{left} \text{ et } \mathbf{e}_{key}^{\sigma} \text{ sont satisfaits lorsque } K^{\sigma} \text{ est utilisée}\}}{2^{32}} \,. \tag{13.3}$$

Notons que cette opération de comptage peut être optimisée car $\mathbf{e}_{key}^{\sigma}$ ne dépend que de 18 bits de R_h.

La procédure décrite à la Figure 13.8 présente une manière d'implémenter cette attaque améliorée. Nous avons décidé de n'exploiter un vecteur de positions sans effet $\mathbf{e}^{\sigma} = (\mathbf{e}_{left}, \mathbf{e}_{key}^{\sigma})$ que lorsqu'au moins une de ses deux instructions `xor_left` les plus influentes s'est montrée sans effet. Quatre injections de fautes sur des `xor_key` sont ainsi économisées quand le gain d'information attendu est négligeable.

Nous avons effectué des simulations extensives de cette attaque avec différents nombres de fautes allant de 15 000 à 100 000. Dans chaque cas, 10 000 simulations ont été effectuées. Pour chaque nombre de fautes considéré, la Table 13.1 donne l'entropie résiduelle en fonction de différents niveaux de percentiles. Les entropies résiduelles médianes pour 50 000 et 100 000 fautes sont de 13,95 et 6,68 bits. Comparé aux chiffres correspondants de la Section 13.3.2, cela montre un gain important pour cette méthode par rapport à l'attaque basique. La Figure 13.9 fournit une visualisation graphique de la décroissance de l'entropie résiduelle de l'espace des clés, en fonction du nombre de fautes.

13.4 Contre-mesures

Nous analysons maintenant les conditions pour que cette attaque soit réalisable, et les contre-mesures qui peuvent l'empêcher.

Comme nous l'avons déjà mentionné, le DES embarqué que nous attaquons doit être implémenté de manière logicielle. Nous pensons que l'attaque proposée n'est pas applicable lorsqu'on utilise un crypto-processeur DES. Nous nous basons également sur une architecture à 8 bits.

TAB. 13.1 – Percentiles de l'entropie résiduelle (en bits) en fonction du nombre de fautes

Nombre de fautes	Niveau de percentile						
	5 %	10 %	25 %	50 %	75 %	90 %	95 %
15 000	23.59	26.33	30.98	36.10	40.46	43.92	46.37
25 000	14.35	16.92	21.51	26.62	31.63	35.86	38.31
35 000	9.17	11.27	15.38	20.23	25.2	29.60	32.31
50 000	5.13	6.80	9.85	**13.95**	18.65	22.96	25.64
70 000	2.81	3.93	6.23	9.57	13.57	17.44	19.95
100 000	1.40	2.26	4.03	**6.68**	10.07	13.59	15.87

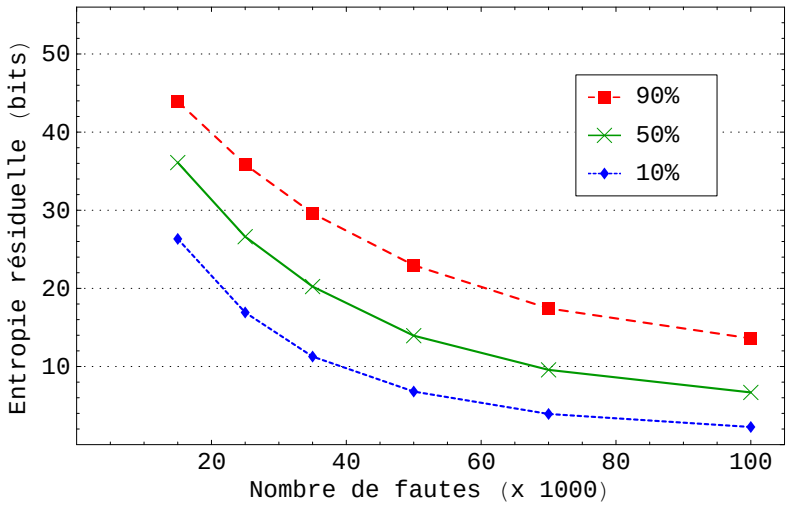


FIG. 13.9 – Percentiles de l'entropie résiduelle en fonction du nombre de fautes.

Cette condition n'est pas strictement nécessaire mais elle a un fort impact sur la complexité de l'attaque. Par exemple, sur une architecture à 16 bits, le nombre moyen de fautes nécessaires pour en obtenir une sans effet (en ciblant une instruction `xor_left`) est de 2^{16} au lieu de 2^8. Les chiffres de complexité que nous avons mentionnés dans le cas 8 bits deviendraient donc prohibitifs pour une réalisation concrète sur des architectures avec des chemins de données plus larges.

Étant donné que l'attaquant a besoin de savoir quelle instruction est corrompue lorsqu'une faute est injectée, nous pensons que la contre-mesure classique d'*insertion de délais aléatoires* (pour voie logicielle ou matérielle) devrait empêcher l'attaque, ou au moins rendre sa réalisation très difficile. En effet, une condition importante est d'être capable d'interpréter des sorties identiques comme résultant d'une sortie naturelle égale à zéro pour le XOR ciblé. En cas de délais aléatoires, cette condition est difficilement vérifiée car :

- on peut avoir une collision en n'impactant rien ou une instruction neutre,
- on risque d'observer des non-collisions en perturbant des opérations autres que le XOR.

Pour des raisons similaires, la contre-mesure d'*exécution en ordre aléatoire* perturbera également l'attaquant. Néanmoins, et bien que nous n'ayons pas étudié cette idée de manière approfondie, nous entrevoyons un moyen d'adapter l'attaque à ce contexte. Quand cette contre-mesure est implémentée seule, et en injectant des fautes de manière répétée sur la même instruction `xor_left` (respectivement, `xor_key`) pour la même entrée, l'attaquant est capable d'inférer le nombre de `xor_left[`i`]` (respectivement, `xor_key[`j`]`) pour lesquels une faute est sans effet. L'observation obtenue par l'attaquant n'est plus le vecteur complet des positions sans effet $(\mathbf{e}_{left}, \mathbf{e}_{key})$, mais plutôt les poids de Hamming de $\mathbf{e}_{left}$ et $\mathbf{e}_{key}$. Probablement au prix d'un plus grand nombre de fautes nécessaires, nous pensons qu'il devrait tout de même être possible d'assigner des probabilités aux clés, sur la base de cette information partielle sur le vecteur des positions sans effet.

Une contre-mesure classique contre les attaques par canaux auxiliaires comme les SPA, DPA, CPA est le *masquage de données* (en anglais, *data blinding*) [GP99] qui conduit, quand il est implémenté correctement, à une imprédictibilité parfaite au premier ordre des valeurs intermédiaires. Une conséquence directe de cette propriété est que l'attaque que nous avons décrite n'est plus possible : toute faute sans effet, conséquence d'une valeur *physique* de sortie du XOR égale à zéro sur l'exécution fautée, est compatible avec n'importe quelle valeur *logique* masquée et ne donne aucune information utile à l'attaquant. Notons que cette contre-mesure contre les attaques par canaux auxiliaires de premier ordre n'est pas efficace contre une variante de notre attaque dans laquelle l'attaquant serait capable d'injecter des fautes multiples à des instants choisis de la même exécution.

Nous considérons enfin la contre-mesure classique contre la DFA et la CFA qui consiste à calculer la fonction cryptographique deux fois, à comparer les deux résultats, et à ne retourner sa valeur que s'ils sont identiques. Comme déjà mentionné dans [YJ00] pour le cas général des attaques par erreurs sûres, nous attirons l'attention sur le fait que cette contre-mesure *n'empêche pas* notre attaque. Une sortie valide indique que la faute a été sans effet, alors qu'aucune sortie signifie qu'elle ne l'était pas. L'attaque est même légèrement simplifiée par le fait que l'attaquant n'a pas besoin de demander les calculs sans faute. La contre-mesure devrait néanmoins retrouver son efficacité si une limite est imposée sur le nombre de fautes détectées permises.

13.5 Conclusion

Nous avons présenté une attaque par révélation de la clé basée sur les fautes et s'appliquant à une implémentation du DES sous forme logicielle. Cette attaque s'appuie sur le modèle de faute suivant : quand une faute est injectée pendant l'exécution d'une instruction XOR, la sortie de ce XOR est forcée à zéro quelles que soient les valeurs des opérandes. Une grande quantité d'information sur la clé secrète K est retrouvée sans connaître les entrées ni les sorties du DES. Seule est nécessaire la possibilité de détecter que deux sorties du DES sont égales. Une conséquence importante est que notre attaque s'applique à la classe entière des DES ou Triple-DES[56] munis d'un encodage externe, c'est-à-dire définis comme des chiffrements par bloc secrets construits en encadrant un DES ou un Triple-DES entre deux permutations secrètes arbitraires. Ceci menace potentiellement des algorithmes cryptographiques propriétaires basés sur ce concept d'obfuscation, et invalide l'immunité supposée de ces fonctions secrètes contre les analyses de fautes. À notre connaissance, notre attaque est le premier exemple publié d'une analyse de fautes transitoires contre cette classe de fonctions cryptographiques secrètes.

Problème ouvert : Nous suggérons enfin certaines directions possibles pour prolonger notre contribution. Des investigations plus poussées peuvent viser à concevoir des variantes de cette attaque qui s'appuieraient sur d'autres modèles de faute réalistes, ou qui s'appliqueraient à d'autres chiffrements par bloc munis d'un encodage externe. Par exemple, un résultat similaire applicable à un AES muni d'un encodage externe menacerait l'utilisation la plus répandue du schéma MILENAGE [3GPP] pour les fonctions d'authentification mutuelle et de génération de clés sur les réseaux de communication mobile de troisième génération.

[56] Le cas du Triple-DES est traité en appliquant successivement l'attaque sur K_1 et K_2.

Cinquième partie

Annexe

Annexe A

Cryptanalyse de signatures RSA avec padding à structure fixe

Sommaire

Nous reportons dans cette annexe un travail n'ayant pas précisément trait aux attaques physiques. Cette cryptanalyse, qui permet de falsifier des signatures RSA utilisant un padding fixe, a été publiée à Crypto '01 avec Éric Brier, Jean-Sébastien Coron et David Naccache [BCCN01].

Un padding à structure fixe consiste à concaténer au message m une structure fixe P. La signature RSA est alors obtenue en calculant $(P\|m)^d \mod N$ où d est l'exposant privé et N le module. À Eurocrypt '97, Marc Girault et Jean-François Misarsky ont montré que la taille de P doit être au moins égale à la moitié de la taille de N (en d'autres termes les configurations de paramètres $|P| < |N|/2$ ne sont pas sûres), mais la sécurité du RSA avec padding à structure fixe demeurait inconnue pour $|P| > |N|/2$. Dans ce chapitre, nous montrons que la taille de P doit être au moins égale aux deux tiers de la taille de N, c'est-à-dire nous montrons que $|P| < 2|N|/3$ n'est pas sûr.

A.1 Introduction

Le crypto-système RSA a été inventé en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman [RSA78], et est maintenant le crypto-système à clé publique le plus largement répandu. RSA est communément utilisé pour assurer le respect de la sphère privée, l'authenticité de données numériques et la sécurisation du trafic sur internet entre serveurs et navigateurs.

Un mode d'utilisation très courant pour signer avec RSA est de commencer par hacher le message, ajouter un padding, puis élever le résultat à la puissance l'exposant de signature. Ce paradigme est à la base de nombreux standards comme PKCS #1 v2.0 [PKCS-1].

Dans ce chapitre, nous considérons les signatures RSA avec padding à structure fixe, sans utilisation d'une fonction de hachage. Pour signer un message m, le signataire concatène un padding fixe P au message, et la signature est obtenue en calculant :

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

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

Plus généralement, nous considérons les signatures RSA dans lesquelles une redondance affine simple est utilisée. Pour signer un message m, le signataire commence par calculer

$$R(m) = \omega \cdot m + a \qquad \text{où} \quad \begin{cases} \omega & \text{est la redondance multiplicative,} \\ a & \text{est la redondance additive.} \end{cases} \tag{A.1}$$

La signature de m est alors :

$$s = R(m)^d \mod N .$$

Un schéma de redondance avec padding à gauche $(P\|m)$ est obtenu en prenant $\omega = 1$ et $a = P \cdot 2^\ell$, alors qu'un schéma de redondance avec padding à droite $(m\|P)$ est obtenu en prenant $\omega = 2^\ell$ et $a = P$.

On ne connaît aucune preuve de sécurité pour les signatures RSA avec redondance affine, et plusieurs attaques sur de tels formats ont vu jour (voir [Mis98] pour un état des lieux de ces attaques). À Crypto '85, Wiebren de Jonge et David Chaum [JC86] ont exhibé une attaque multiplicative contre les signatures RSA avec redondance affine, basée sur l'algorithme d'Euclide étendu. Leur attaque s'applique lorsque la redondance multiplicative ω est égale à 1 et la taille du message est au moins égale aux deux tiers de la taille du module RSA N :

$$|\text{message}| \succ \frac{2}{3}|N| .$$

Par exemple, une signature peut être falsifiée si l'on utilise la redondance affine de la Figure A.1.

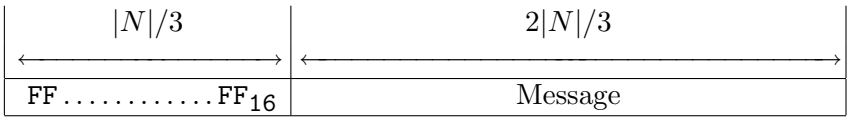


Fig. A.1 – Exemple de padding RSA falsifiable par la méthode de Wiebren de Jonge et David Chaum où $\omega = 1$ et $a = \texttt{FF}\ldots\texttt{FF}\ \texttt{00}\ldots\texttt{00}_{16}$

L'attaque de de Jonge et Chaum a été étendue par Marc Girault et Jean-François Misarsky [GM97] à Eurocrypt '97, en utilisant l'algorithme de Okamoto-Shiraishi [OS85], qui est une extension de l'algorithme d'Euclide étendu. Ils ont augmenté le champ d'application des attaques multiplicatives sur les signatures RSA avec redondance affine puisque leur attaque s'applique à toutes valeurs de ω et de a, lorsque la taille du message est au moins égale à la moitié de la taille du module (voir la Figure A.2 pour une illustration) :

$$|\text{message}| \succ \frac{1}{2}|N| .$$

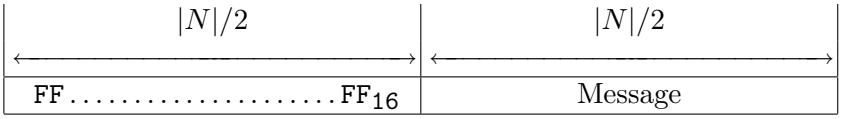


FIG. A.2 – Exemple de padding RSA falsifiable par la méthode de MARC GIRAULT et JEAN-FRANÇOIS MISARSKY où $\omega = 1$ et $a = \texttt{FF} \ldots \texttt{FF}\ \ \texttt{00} \ldots \texttt{00}_{16}$

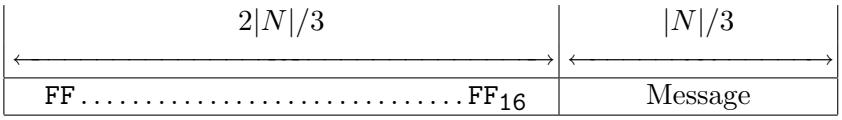


FIG. A.3 – Exemple de padding RSA falsifiable par notre méthode lorsque ω est égal à 1 et $a = \texttt{FF} \ldots \texttt{FF}\ \ \texttt{00} \ldots \texttt{00}_{16}$

MARC GIRAULT et JEAN-FRANÇOIS MISARSKY ont également étendu les attaques multiplicatives aux signatures RSA avec redondance modulaire :

$$R(m) = \omega_1 \cdot m + \omega_2 \cdot (m \bmod b) + a \ , \tag{A.2}$$

$$\text{où} \quad \begin{cases} \omega_1, \omega_2 & \text{est la redondance multiplicative,} \\ a & \text{est la redondance additive,} \\ b & \text{est la redondance modulaire.} \end{cases}$$

Dans ce cas, la taille du message doit être au moins égale à la moitié de la taille du module plus la taille de la redondance modulaire.

Enfin, l'attaque de GIRAULT et MISARSKY a été étendue par MISARSKY [Mis97] à CRYPTO '97 à une fonction de redondance dans laquelle le message m et la redondance modulaire m mod b peuvent être scindés en plusieurs parties, en utilisant l'algorithme LLL [LLL82]. L'attaque s'applique lorsque la taille du message est au moins la moitié de la taille du module plus la taille de la redondance modulaire.

Dans ce chapitre, nous étendons l'attaque de GIRAULT et MISARSKY contre les signatures RSA à redondance affine, à des messages de taille aussi petite qu'un tiers de la taille du module, comme illustré sur la Figure A.3.

$$|\text{message}| \succ \frac{1}{3}|N| \ .$$

Comme dans l'attaque de GIRAULT et MISARSKY, nous attaque s'applique pour toutes valeurs de ω et a et s'exécute en temps polynomial. Cependant notre attaque n'est qu'existentielle puisque nous ne pouvons pas choisir le message dont on souhaite falsifier la signature, alors que l'attaque de GIRAULT et MISARSKY est sélective : ils peuvent choisir le message dont la signature est falsifiée.

A.2 Notre nouvelle attaque

Dans cette section nous étendons l'attaque multiplicative de GIRAULT et MISARSKY sur les signatures RSA avec redondance affine, aux messages de taille aussi petite qu'un tiers de

la taille du module N. Une attaque multiplicative est une attaque dans laquelle la fonction de redondance d'un message peut être exprimée comme une combinaison multiplicative des fonctions de redondance d'autres messages. Nous cherchons ainsi quatre messages distincts m_1, m_2, m_3 et m_4, chacun de taille égale à un tiers de celle du module, tels que :

$$R(m_1) \cdot R(m_2) = R(m_3) \cdot R(m_4) \mod N \quad . \tag{A.3}$$

On peut alors, en utilisant les signatures de m_2, m_3 et m_4, falsifier la signature de m_1 par :

$$R(m_1)^d = \frac{R(m_3)^d \cdot R(m_4)^d}{R(m_2)^d} \mod N \quad .$$

De (A.3) on obtient

$$(\omega \cdot m_1 + a) \cdot (\omega \cdot m_2 + a) = (\omega \cdot m_3 + a) \cdot (\omega \cdot m_4 + a) \mod N \quad .$$

En notant $P = a/\omega \mod N$, nous obtenons

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

et en posant

$$\begin{array}{rclcrcl} t & = & m_3 & & y & = & m_2 - m_3 \\ x & = & m_1 - m_3 & & z & = & m_4 - m_1 - m_2 + m_3 \end{array} \tag{A.4}$$

on obtient

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

qui se simplifie en

$$x \cdot y = (P + t) \cdot z \mod N \quad . \tag{A.5}$$

Notre but est donc de trouver quatre entiers x, y, z et t, chacun aussi petit qu'un tiers de la taille de N, vérifiant l'Equation (A.5).

Nous obtenons tout d'abord deux entiers z et u tels que

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

Comme il est remarqué dans [GTV90], ceci est équivalent à trouver une bonne approximation de la fraction P/N, et peut être réalisé efficacement en la développant en fractions continues, par exemple en appliquant l'algorithme d'Euclide étendu à P et N. On trouve une solution telle que $|z| < Z$ et $0 < u < U$ si $Z \cdot U > N$, ce qui est le cas ici avec $Z = N^{\frac{1}{3}}$ et $U = 2 \cdot N^{\frac{2}{3}}$.

Nous sélectionnons alors un entier y tel que $N^{\frac{1}{3}} \leqslant y \leqslant 2 \cdot N^{\frac{1}{3}}$ et $\text{pgcd}(y, z) = 1$. Nous trouvons l'entier positif $t < y$ tel que :

$$t \cdot z = -u \mod y \ ,$$

ce qui est possible puisque $\text{pgcd}(y, z) = 1$. Nous prenons alors

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

et obtenons

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

ce qui donne l'Equation (A.5), avec x, y, z et t aussi petits que $4 \cdot N^{\frac{1}{3}}$. De x, y, z, t nous en déduisons, en utilisant (A.4), quatre messages m_1, m_2, m_3 et m_4, chacun de taille un tiers de la taille de N :

$$\begin{array}{lcllcl} m_1 & = & x+t & m_2 & = & y+t \\ m_3 & = & t & m_4 & = & x+y+z+t \end{array} \quad . \tag{A.6}$$

Puisque $-N^{1/3} < z < N^{1/3}$ et $y \geqslant N^{1/3}$, nous avons $y+z > 0$, ce qui donne en utilisant $u \geqslant 0$:

$$x + t = \frac{u + t \cdot (y+z)}{y} \geqslant 0 \ ,$$

ce qui montre que les quatre entiers m_1, m_2, m_3 et m_3 sont positifs, et nous avons

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

La complexité de notre attaque est polynomiale en la taille de N car bornée par celle de l'algorithme d'Euclide étendu.

Exemple : Nous donnons ici un exemple d'une telle falsification avec $\omega = 1$ et $a = 2^{1023} - 2^{352}$, le module N étant le challenge officiel RSA-309 de RSA Laboratories, dont la factorisation est actuellement inconnue.

$N =$ RSA-309

```
=  bdd14965 645e9e42 e7f658c6 fc3e4c73 c69dc246 451c714e b182305b 0fd6ed47
   d84bc9a6 10172fb5 6dae2f89 fa40e7c9 521ec3f9 7ea12ff7 c3248181 ceba33b5
   5212378b 579ae662 7bcc0821 30955234 e5b26a3e 425bc125 4326173d 5f4e25a6
   d2e172fe 62d81ced 2c9f362b 982f3065 0881ce46 b7d52f14 885eecf9 03076ca5

R(m1) = 7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff 00415df4 ca4219b6 ea5fa8e4
        e2eabcfc 61348b80 e7ccbac7 3d1f5cc7 249e1519 9412886a f76220c6 d1409cd6

R(m2) = 7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff 00127f44 f753253a a0348be7
        826e893f 693032db c2194dbb 3b81e1c2 630b66d3 1448a3f4 7fd2d34f b28aefd6

R(m3) = 7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff 00781bd4 e0c918a7 308fcff7
        8f64044c a35b4937 36cd37d7 93f281b5 fdd0a951 52a0479b 57dd73b2 25b6df85

R(m4) = 7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
        ffffffff ffffffff ffffffff ffffffff ffffffff 000919fd 86e5afce 7fc11c94
        0e0827c8 03be05bb 71f8de48 c61d6d5f 0feb036d a1ff2f8b 5f596108 3d142538
```

Nous obtenons :

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

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

A.3 Extension à une falsification sélective

L'attaque de la section précédente n'est qu'existentielle : nous ne pouvons pas choisir le message dont la signature est falsifiée. Dans cette section, nous montrons comment nous pouvons réaliser une falsification sélective. Soit m_3 le message dont nous voulons falsifier la signature.

Posant x, y, z et t comme dans (A.4), nous trouvons, en appliquant l'algorithme d'Euclide étendu, deux couples d'entiers (z_1, u_1) et (z_2, u_2) vérifiant :

$$\begin{cases} (P+t)\cdot z_1 = u_1 \mod N \\ (P+t)\cdot z_2 = u_2 \mod N \end{cases},$$

et tels que $|z_1|$ et $|z_2|$ sont proches de $N^{1/3}$, et u_1 et u_2 sont proches de $N^{2/3}$.

Comme combinaisons linéaires de (z_1, u_1) et de (z_2, u_2), nous pouvons générer des couples (z, u) d'entiers positifs respectivement proches de $N^{1/3}$ et de $N^{2/3}$, définis par :

$$\begin{cases} z = \lambda z_1 + \mu z_2 \\ u = \lambda u_1 + \mu u_2 \end{cases},$$

et tels que :

$$(P+t)\cdot z = u \mod N\,.$$

Lorsque l'entier u peut être exprimé comme le produit $x \cdot y$ de deux entiers de tailles à peu près égales, nous obtenons quatre entiers x, y, z, t de tailles environ un tiers de la taille du module, avec :

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

ce qui donne :

$$R(m_1)\cdot R(m_2) = R(m_3)\cdot R(m_4) \mod N\,.$$

La signature de m_3 peut maintenant être falsifiée en utilisant les signatures de m_1, m_2 et m_4.

La liberté que l'on se donne de pouvoir générer plusieurs candidats u permet d'exploiter le fait que certains sont plus facilement factorisables que d'autres. Malgré cela, et contrairement à l'attaque de la section précédente, la complexité de celle-ci n'est pas polynomiale. Pour un module de 512 bits l'attaque par falsification sélective est facilement réalisable. Pour un module de 1024 bits, l'attaque est plus difficile mais tout de même faisable.

Exemple : Voici un exemple de falsification sélective avec $\omega = 1$ et $a = 2^{1023} - 2^{352}$, et le module RSA-309 de factorisation inconnue. Le message m_3 est choisi comme le codage ASCII du texte : "`Une falsification sélective sur 1024 bits.`".

$R(m_1) =$
```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff 007b9599 bef4bf6b 71c0cb0e
11f06631 f6289f2b 27666382 401ac93b 1f16e24e e4ab6ec3 189b6ff8 b970bffd
```

$R(m_2) =$
```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff 019cd1e8 9bced9d2 41d6c11d
4d89ba0b 5253112c 6b66534e 85e6ae8c 320c2b00 6d5feb29 0b0c9676 1b1dd52f
```

$R(m_3) =$
```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff 0000556e 65206661 6c736966
69636174 696f6e20 73e96c65 63746976 65207375 72203130 32342062 6974732e
```

$R(m_4) =$
```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff 02bbea36 9924302b e222f150
54771ae9 8f2e4e80 cccef8f6 dc942b21 20ffb55d 7973e8e6 2d4fd8a1 9bbe4760
```

A.4 Conclusion

Nous avons étendu l'attaque de Marc Girault et Jean-François Misarsky sur les signatures RSA avec redondance affine : nous avons décrit une attaque à messages choisis contre les signatures RSA avec redondance affine pour des messages aussi petits qu'un tiers de la taille du module. En conséquence, lorsqu'on utilise un padding fixe $(P\|m)$ ou $(m\|P)$, la taille de P doit être au moins égale aux deux tiers de la taille de N. Notre attaque est polynomiale en la longueur du module. Nous ne savons pas s'il existe une attaque en temps polynomial contre les signatures RSA avec redondance affine pour des messages plus petits qu'un tiers de la taille du module. Néanmoins, nous pensons que rechercher dans quelle mesure le padding affine est malléable accroît notre compréhension des propriétés du RSA et de ses limitations.

Problème ouvert : Est-il possible d'étendre cette attaque à des taille relatives de messages encore plus petites (ou, de manière équivalente, à des constantes de structure fixe plus grandes) ?

Conclusion

Nous espérons, à travers les quelques travaux présentés dans cette thèse, avoir pu donner au lecteur un aperçu, certes très partiel, de l'évolution de la recherche sur la sécurité physique des crypto-systèmes embarqués durant ces dix dernières années. Une nouvelle méthode générique d'analyse des canaux auxiliaires a été proposée. Des attaques inédites s'appliquant spécifiquement à certaines implémentations ou à certains algorithmes ont été décrites. Des contre-mesures ont été proposées, et d'autres ont été étudiées pour en préciser les limitations.

Nous constatons qu'une majorité des sujets abordés dans ce document se positionnent dans un champ généralement encore peu étudié : l'analyse du courant sur une génération de clé, l'exploitation de l'altération d'un élément public de clé RSA, l'étude de la sécurité de certains crypto-processeurs asynchrones, une proposition d'exponentiation modulaire par chaîne d'additions "prouvée sûre" contre une certaine classe d'attaques et d'adversaires, et enfin plusieurs attaques sur des algorithmes inconnus de l'adversaire.

Au delà de cette évolution passée, nous entrevoyons plusieurs directions vers lesquelles pourrait s'orienter, à l'avenir, la recherche sur les attaques physiques de crypto-systèmes embarqués.

Tout d'abord, il est intéressant de constater une volonté forte de la part des concepteurs de circuits de sécuriser leurs produits à l'aide de contre-mesures au niveau porte visant à annihiler toute fuite de courant au niveau le plus élémentaire. Bien que les publications actuelles sur ce sujet montrent que les propositions de contre-mesures sont souvent mises à mal après avoir été publiées, nous pensons qu'il faut s'attendre à une stabilisation progressive dans ce domaine et, souhaitons le, à de prochains succès de cette approche, tout à la fois séduisante et fondamentale car s'attaquant à la source même de la fuite d'information.

Bien que nous ne les ayons qu'effleurées dans cette thèse, les analyses statistiques d'ordre supérieur constituent aujourd'hui une menace qui doit être prise en considération avec le même sérieux qui a été consacré à celles de premier ordre par le passé. La conception de contre-mesures génériques et efficaces vis-à-vis de ces attaques constitue ainsi un axe de recherche intéressant et important pour cette discipline.

La plupart des attaques physiques que nous avons présentées permettent de retrouver un secret à partir de sa manipulation par l'algorithme qui l'utilise. Ces attaques opèrent donc au niveau cryptographique et sont très liées aux propriétés mathématiques de la fonction sous-jacente. Il existe cependant un champ d'applicabilité des attaques physiques se situant à un niveau logiciel supérieur. Cela inclut toutes les routines sensibles du logiciel embarqué (vérification d'un PIN, lecture de fichier,...), ainsi que les plate-formes ouvertes, éventuellement multi-applicatives. L'exploitation des attaques physiques à ce niveau peut donner jour à des menaces réelles dont certaines sont déjà prises en compte dans la conception des cartes à puces. Nous observons néanmoins que ce champ d'investigation souffre injustement d'une faible considération au niveau de la recherche académique. Nous espérons que la recherche sur la sécurité physique à un niveau logiciel plus éloigné de la fonction cryptographique suscitera à l'avenir un

intérêt plus grand de la part de la communauté scientifique.

Dans la dernière partie de ce mémoire, nous avons commencé à étudier l'applicabilité des attaques physiques lorsque la fonction cryptographique analysée n'est pas connue de l'adversaire. Nous avons montré que certaines attaques spécifiques peuvent permettre soit une rétro-conception d'une partie des détails de l'algorithme, soit, plus classiquement, une révélation de la clé. Considérant le nombre important d'applications intégrant des algorithmes propriétaires non publics, nous pensons que l'étude de la sécurité physique des crypto-systèmes entièrement ou partiellement inconnus constitue un domaine de recherche intéressant tout autant qu'important. Cet axe de recherche pourrait inclure dans ses objectifs une définition de modèles d'attaquant dans le but de formaliser la portée des attaques physiques dans ce contexte.

Enfin, il existe tout un espace d'applicabilité des analyses de canaux auxiliaires qui nous semble très important même s'il ne relève pas *a priori* du sujet de cette thèse. Il s'agit des attaques sur des crypto-systèmes dont l'exécution est réalisée sur un dispositif inaccessible à l'attaquant. Il n'est pas question ici d'attaquer une carte à puce par analyse du courant ou par exploitation de fautes, mais plutôt un serveur internet ou une borne WI-FI, à l'aide d'un canal auxiliaire différent comme par exemple, un temps d'exécution dépendant d'un nombre plus ou moins grand de lectures hors de la mémoire cache, ou bien la réponse à un contrôle de la conformité d'une requête. Libérer l'adversaire de la contrainte d'avoir un accès physique au dispositif attaqué induit une extension considérable du nombre de menaces possibles, et justifie selon nous que cette classe d'attaques par analyse de canaux auxiliaires soit considérée à l'avenir avec une grande attention.

Publications

[Cla07b] Christophe Clavier. An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. Dans *International Conference on Information Systems Security – ICISS '07*, Lecture Notes in Computer Science. Springer-Verlag, 2007. À paraître.

[Cla07a] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. Dans Pascal Paillier et Ingrid Verbauwhede, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '07*, volume 4727 de *Lecture Notes in Computer Science*, pages 181–194. Springer-Verlag, 2007.

[CC07] Christophe Clavier et Jean-Sébastien Coron. On the implementation of a fast prime generation algorithm. Dans Pascal Paillier et Ingrid Verbauwhede, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '07*, volume 4727 de *Lecture Notes in Computer Science*, pages 443–449. Springer-Verlag, 2007.

[BCCC06] Éric Brier, Benoît Chevallier-Mames, Mathieu Ciet et Christophe Clavier. Why one should also secure RSA public key elements. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 324–338. Springer-Verlag, 2006.

[MRL+06b] Yannick Monnet, Marc Renaudin, Régis Leveugle, Christophe Clavier et Pascal Moitrel. Case study of a fault attack on asynchronous DES crypto-processors. Dans Luca Breveglieri, Israel Koren, David Naccache et Jean-Pierre Seifert, éditeurs, *Fault Diagnosis and Tolerance in Cryptography – FDTC '06*, volume 4236 de *Lecture Notes in Computer Science*, pages 88–97. Springer-Verlag, 2006.

[ACT06] Frédéric Amiel, Christophe Clavier et Michael Tunstall. Fault analysis of DPA-resistant algorithms. Dans Luca Breveglieri, Israel Koren, David Naccache et Jean-Pierre Seifert, éditeurs, *Fault Diagnosis and Tolerance in Cryptography – FDTC '06*, volume 4236 de *Lecture Notes in Computer Science*, pages 223–236. Springer-Verlag, 2006.

[BCO04] Éric Brier, Christophe Clavier et Francis Olivier. Correlation power analysis with a leakage model. Dans Marc Joye et Jean-Jacques Quisquater, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '04*, volume 3156 de *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag, 2004.

[BCCN01] Éric Brier, Christophe Clavier, Jean-Sébastien Coron et David Naccache. Cryptanalysis of RSA signatures with fixed-pattern padding. Dans Joe Kilian, éditeur, *Advances in Cryptology – CRYPTO '01*, volume 2139 de *Lecture Notes in Computer Science*, pages 433–439. Springer-Verlag, 2001.

[CJ01] Christophe Clavier et Marc Joye. Universal exponentiation algorithm. Dans Çetin Kaya Koç, David Naccache et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '01*, volume 2162 de *Lecture Notes in Computer Science*, pages 300–308. Springer-Verlag, 2001.

[CCD00] Christophe Clavier, Jean-Sébastien Coron et Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 de *Lecture Notes in Computer Science*, pages 252–263. Springer-Verlag, 2000.

Brevets déposés

[1] Éric Brier et Christophe Clavier. Procédé de génération de clés sécurisées pour un algorithme cryptographique. Publié le 22 août 2003. Numéro de publication : FR2836312.

[2] Christophe Clavier, Nathalie Feyt, Pascal Guterman, Sébastien Petit et Philippe Proust. Procédé de mise en œuvre sécurisée d'un module fonctionnel, dans un composant électronique et composant correspondant. Publié le 2 mai 2003. Numéro de publication : FR2831739.

[3] Christophe Clavier et Marc Joye. Dispositif destiné à réaliser des calculs d'exponentiation, et procédé de programmation et d'utilisation du dispositif. Délivré le 8 août 2003.

[4] Christophe Clavier et Nathalie Feyt. Procédé de génération de clés électroniques pour la mise en œuvre d'un algorithme cryptographique, carte à puce mettant en œuvre le procédé. Publié le 24 janvier 2003. Numéro de publication : FR2827722.

[5] Christophe Clavier et Jean-Sébastien Coron. Procédé de contrôle d'utilisation d'une carte à puce. Délivré le 29 décembre 2000.

[6] Olivier Benoit et Christophe Clavier. Procédé de contre-mesure dans un composant électronique mettant en œuvre un algorithme de cryptographie à clé secrète. Délivré le 15 décembre 2000.

[7] Olivier Benoit et Christophe Clavier. Procédé de contre-mesure dans un composant électronique mettant en œuvre un algorithme de cryptographie à clé secrète. Délivré le 29 décembre 2000.

Bibliographie

[3GPP] 3GPP Project. Specification of the MILENAGE algorithm set : An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5* ; document 2 : Algorithm specification. 3GPP TS 35.206. http://www.3gpp.org/ftp/Specs/html-info/35206.htm.

[AARR03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao et Pankaj Rohatgi. The EM side-channel(s). Dans Burton S. Kaliski, Jr, Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 de *Lecture Notes in Computer Science*, pages 29–45. Springer-Verlag, 2003.

[ABDM00] Mehdi-Laurent Akkar, Régis Bévan, Paul Dischamp et Didier Moyart. Power analysis, what is now possible.... Dans Tatsuaki Okamoto, éditeur, *Advances in Cryptology – ASIACRYPT '00*, volume 1976 de *Lecture Notes in Computer Science*, pages 489–502. Springer-Verlag, 2000.

[ABF+02] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter et Jean-Pierre Seifert. Fault attacks on RSA with CRT : Concrete results and practical countermeasures. Dans Burton S. Kaliski, Jr, Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 de *Lecture Notes in Computer Science*, pages 260–275. Springer-Verlag, 2003.

[ACT06] Frédéric Amiel, Christophe Clavier et Michael Tunstall. Fault analysis of DPA-resistant algorithms. Dans Luca Breveglieri, Israel Koren, David Naccache et Jean-Pierre Seifert, éditeurs, *Fault Diagnosis and Tolerance in Cryptography – FDTC '06*, volume 4236 de *Lecture Notes in Computer Science*, pages 223–236. Springer-Verlag, 2006.

[AG01] Mehdi-Laurent Akkar et Christophe Giraud. An implementation of DES and AES, secure against some attacks. Dans Çetin Kaya Koç, David Naccache et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '01*, volume 2162 de *Lecture Notes in Computer Science*, pages 309–318. Springer-Verlag, 2001.

[AK96] Ross J. Anderson et Markus G. Kuhn. Tamper resistance — a cautionary note. Dans *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–11, novembre 1996.

[AK98] Ross J. Anderson et Markus G. Kuhn. Low cost attacks on tamper resistant devices. Dans Bruce Christianson, Bruno Crispo, T. Mark A. Lomas et Michael Roe, éditeurs, *Security Protocols Workshop*, volume 1361 de *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 1997.

[Akk04] Mehdi-Laurent Akkar. *Attaques et méthodes de protections de systèmes cryptographiques embarqués*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, France, octobre 2004.

[APSQ06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert et Jean-Jacques Quisquater. Template attacks in principal subspaces. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2006.

[BBBD89] François Bergeron, Jean Berstel, Srecko Brlek et Christine Duboc. Addition chains using continued fractions. *Journal of Algorithms*, 10(3):403–412, 1989.

[BCCC06] Éric Brier, Benoît Chevallier-Mames, Mathieu Ciet et Christophe Clavier. Why one should also secure RSA public key elements. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 324–338. Springer-Verlag, 2006.

[BCCN01] Éric Brier, Christophe Clavier, Jean-Sébastien Coron et David Naccache. Cryptanalysis of RSA signatures with fixed-pattern padding. Dans Joe Kilian, éditeur, *Advances in Cryptology – CRYPTO '01*, volume 2139 de *Lecture Notes in Computer Science*, pages 433–439. Springer-Verlag, 2001.

[BCN+06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, et Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE, Special Issue on Cryptography and Security*, 94(2):370–382, février 2006.

[BCO03] Éric Brier, Christophe Clavier et Francis Olivier. Optimal statistical power analysis. *Cryptology ePrint Archive*, Report 2003/152, 2003.

[BCO04] Éric Brier, Christophe Clavier et Francis Olivier. Correlation power analysis with a leakage model. Dans Marc Joye et Jean-Jacques Quisquater, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '04*, volume 3156 de *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag, 2004.

[BDF98a] Dan Boneh, Glenn Durfee et Yair Frankel. An attack on RSA given a small fraction of the private key bits. Dans Kazuo Ohta et Dingyi Pei, éditeurs, *Advances in Cryptology – ASIACRYPT '98*, volume 1514 de *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag, 1998.

[BDF98b] Dan Boneh, Glenn Durfee et Yair Frankel. Exposing an RSA private key given a small fraction of its bits. Version complète du travail paru à ASIACRYPT '98, 1998. http://citeseer.ist.psu.edu/113533.html.

[BDK07] Eli Biham, Orr Dunkelman et Nathan Keller. A new attack on 6-round IDEA. Dans Alex Biryukov, éditeur, *Fast Software Encryption – FSE '07*, volume 4593 de *Lecture Notes in Computer Science*, pages 211–224. Springer-Verlag, 2007.

[BDL01] Dan Boneh, Richard A. DeMillo et Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, 2001.

[BDL97] Dan Boneh, Richard A. DeMillo et Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). Dans Walter Fumy, éditeur, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 de *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval et Phillip Rogaway. Relations among notions of security for public-key encryption schemes. Dans Hugo Krawczyk, éditeur, *Advances in Cryptology – CRYPTO '98*, volume 1462 de *Lecture Notes in Computer Science*, pages 26–45. Springer-Verlag, 1998.

[BGLT06] Marco Bucci, Luca Giancane, Raimondo Luzzi et Alessandro Trifiletti. Three-phase dual-rail pre-charge logic. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 232–241. Springer-Verlag, 2006.

[BGW98] Marc Briceno, Ian Goldberg et David Wagner. Smartcard developer association clones digital GSM cellphones, avril 1998. http://www.efc.ca/pages/crypto/smartcard.13apr98.html.

[BK03] Régis Bévan et Erik Knudsen. Ways to enhance differential power analysis. Dans Pil Joong Lee et Chae Hoon Lim, éditeurs, *Information Security and Cryptology – ICISC '02*, volume 2587 de *Lecture Notes in Computer Science*, pages 327–342. Springer-Verlag, 2003.

[BMM00] Ingrid Biehl, Bernd Meyer et Volker Müller. Differential fault attacks on elliptic curve cryptosystems. Dans Mihir Bellare, éditeur, *Advances in Cryptology – CRYPTO '00*, volume 1880 de *Lecture Notes in Computer Science*, pages 131–146. Springer-Verlag, 2000.

[Bor06] Michele Boreale. Attacking right-to-left modular exponentiation with timely random faults. Dans Luca Breveglieri, Israel Koren, David Naccache et Jean-Pierre Seifert, éditeurs, *Fault Diagnosis and Tolerance in Cryptography – FDTC '06*, volume 4236 de *Lecture Notes in Computer Science*, pages 24–35. Springer-Verlag, 2006.

[BR93] Mihir Bellare et Phillip Rogaway. Random oracles are practical : A paradigm for designing efficient protocols. Dans *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR95] Mihir Bellare et Phillip Rogaway. Optimal asymmetric encryption. Dans Alfredo de Santis, éditeur, *Advances in Cryptology – EUROCRYPT '94*, volume 950 de *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995.

[BR96] Mihir Bellare et Phillip Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. Dans Ueli M. Maurer, éditeur, *Advances in Cryptology – EUROCRYPT '96*, volume 1070 de *Lecture Notes in Computer Science*, pages 399–416. Springer-Verlag, 1996.

[BS03] Johannes Blömer et Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). Dans Rebecca N. Wright, éditeur, *Financial Cryptography – FC '03*, volume 2742 de *Lecture Notes in Computer Science*, pages 162–181. Springer-Verlag, 2003.

[BS91] Eli Biham et Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. Dans Alfred Menezes et Scott A. Vanstone, éditeurs, *Advances in Cryptology*

– *CRYPTO '90*, volume 537 de *Lecture Notes in Computer Science*, pages 2–21. Springer-Verlag, 1991.

[BS93] Eli Biham et Adi Shamir. Differential cryptanalysis of the full 16-round DES. Dans Ernest F. Brickell, éditeur, *Advances in Cryptology – CRYPTO '92*, volume 740 de *Lecture Notes in Computer Science*, pages 487–496. Springer-Verlag, 1993.

[BS96a] Eli Biham et Adi Shamir. The next stage of differential fault analysis : How to break completely unknown cryptosystems. Draft, octobre 1996. www.fit.vutbr.cz/~cvrcek/cards/nextstage.ps.

[BS96b] Eli Biham et Adi Shamir. Differential fault analysis : Identifying the structure of unknown ciphers sealed in tamper-proof devices. Draft, novembre 1996. www.fit.vutbr.cz/~cvrcek/cards/sealed.ps.

[BS97] Eli Biham et Adi Shamir. Differential fault analysis of secret key cryptosystems. Dans Burton S. Kaliski, Jr, éditeur, *Advances in Cryptology – CRYPTO '97*, volume 1294 de *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag, 1997.

[CC05] Cécile Canovas et Jessy Clédière. What do S-boxes say in differential side channel attacks ? *Cryptology ePrint Archive*, Report 2005/311, 2005.

[CC07] Christophe Clavier et Jean-Sébastien Coron. On the implementation of a fast prime generation algorithm. Dans Pascal Paillier et Ingrid Verbauwhede, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '07*, volume 4727 de *Lecture Notes in Computer Science*, pages 443–449. Springer-Verlag, 2007.

[CCD00] Christophe Clavier, Jean-Sébastien Coron et Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 de *Lecture Notes in Computer Science*, pages 252–263. Springer-Verlag, 2000.

[CCJ04] Benoît Chevallier-Mames, Mathieu Ciet et Marc Joye. Low-cost solutions for preventing simple side-channel analysis : Side-channel atomicity. *IEEE Transanctions on Computers*, 53(6):760–768, 2004.

[CG00] Jean-Sébastien Coron et Louis Goubin. On boolean and arithmetic masking against differential power analysis. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 de *Lecture Notes in Computer Science*, pages 231–237. Springer-Verlag, 2000.

[CJ01] Christophe Clavier et Marc Joye. Universal exponentiation algorithm. Dans Çetin Kaya Koç, David Naccache et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '01*, volume 2162 de *Lecture Notes in Computer Science*, pages 300–308. Springer-Verlag, 2001.

[CJ05] Mathieu Ciet et Marc Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, Codes and Cryptography*, 36(1):33–43, 2005.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao et Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. Dans Michael J.

Wiener, éditeur, *Advances in Cryptology – CRYPTO '99*, volume 1666 de *Lecture Notes in Computer Science*, pages 398–412. Springer-Verlag, 1999.

[CKN01] Jean-Sébastien Coron, Paul C. Kocher et David Naccache. Statistics and secret leakage. Dans Yair Frankel, éditeur, *Financial Cryptography – FC '00*, volume 1962 de *Lecture Notes in Computer Science*, pages 157–173. Springer-Verlag, 2001.

[Cla04] Christophe Clavier. Side channel analysis for reverse engineering (SCARE) – an improved attack against a secret A3/A8 GSM algorithm. *Cryptology ePrint Archive*, Report 2004/049, 2004.

[Cla07a] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. Dans Pascal Paillier et Ingrid Verbauwhede, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '07*, volume 4727 de *Lecture Notes in Computer Science*, pages 181–194. Springer-Verlag, 2007.

[Cla07b] Christophe Clavier. An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. Dans *International Conference on Information Systems Security – ICISS '07*, Lecture Notes in Computer Science. Springer-Verlag, 2007. À paraître.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997. Version révisée de deux articles d'Eurocrypt '96.

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. Dans Lars R. Knudsen, éditeur, *Advances in Cryptology – EUROCRYPT '02*, volume 2332 de *Lecture Notes in Computer Science*, pages 272–287. Springer-Verlag, 2002.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 de *Lecture Notes in Computer Science*, pages 292–302. Springer-Verlag, 1999.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff et Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. Dans Pascal Paillier et Ingrid Verbauwhede, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '07*, volume 4727 de *Lecture Notes in Computer Science*, pages 28–44. Springer-Verlag, 2007.

[CRR03] Suresh Chari, Josyula R. Rao et Pankaj Rohatgi. Template attacks. Dans Burton S. Kaliski, Jr, Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 de *Lecture Notes in Computer Science*, pages 13–28. Springer-Verlag, 2003.

[CT05] Hamid Choukri et Michael Tunstall. Round reduction using faults. Dans *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC '05*, pages 13–24, Edinburgh, Scotland, 2005.

[CY03] Chien-Ning Chen et Sung-Ming Yen. Differential fault analysis on AES key schedule and some coutnermeasures. Dans Reihaneh Safavi-Naini et Jennifer Seberry, éditeurs, *Information Security and Privacy, 8th Australasian Conference, ACISP '03*, volume 2727 de *Lecture Notes in Computer Science*, pages 118–129. Springer-Verlag, 2003.

[CZ06] Zhimin Chen et Yujie Zhou. Dual-rail random switching logic : A countermeasure to reduce side channel leakage. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 242–254. Springer-Verlag, 2006.

[DH76] Whitfield Diffie et Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, novembre 1976.

[DK05] William Dupuy et Sébastien Kunz-Jacques. Resistance of randomized projective coordinates against power analysis. Dans Josyula R. Rao et Berk Sunar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '05*, volume 3659 de *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2005.

[DKL+00] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater et Jean-Louis Willems. A practical implementation of the timing attack. Dans Jean-Jacques Quisquater et Bruce Schneier, éditeurs, *Smart Card Research and Advanced Application – CARDIS '98*, volume 1820 de *Lecture Notes in Computer Science*, pages 167–182. Springer-Verlag, 2000.

[DJRV00] Jean Donio, Jean Leroux les Jardins, Edouard de Rocca et Malika Verstrepen. *La carte à puce*, volume 3492 de *Que sais-je ?* Presses Universitaires de France, mars 2000.

[DLMV05] Rémy Daudigny, Hervé Ledig, Frédéric Muller et Frédéric Valette. SCARE of the DES. Dans John Ioannidis, Angelos D. Keromytis et Moti Yung, éditeurs, *Applied Cryptography and Network Security – ACNS '05*, volume 3531 de *Lecture Notes in Computer Science*, pages 393–406. Springer-Verlag, 2003.

[DLV03a] Pierre Dusart, Gilles Letourneux et Olivier Vivolo. Differential fault analysis on AES. *Cryptology ePrint Archive*, Report 2003/010, 2003.

[DLV03b] Pierre Dusart, Gilles Letourneux et Olivier Vivolo. Differential fault analysis on AES. Dans Jianying Zhou, Moti Yung et Yongfei Han, éditeurs, *Applied Cryptography and Network Security – ACNS '03*, volume 2846 de *Lecture Notes in Computer Science*, pages 293–306. Springer-Verlag, 2003.

[DR00] Joan Daemen et Vincent Rijmen. Rijndael for AES. Dans *AES Candidate Conference*, pages 343–348, 2000.

[DR02] Joan Daemen et Vincent Rijmen. *The design of Rijndael : AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[DR99] Joan Daemen et Vincent Rijmen. AES proposal : Rijndael, sep 1999. http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. Dans G. R. Blakley et David Chaum, éditeurs, *Advances in Cryptology – CRYPTO '84*, volume 196 de *Lecture Notes in Computer Science*, pages 10–18. Springer-Verlag, 1985.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[ER60] Paul Erdős et Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960.

[FKM+06] Pierre-Alain Fouque, Sébastien Kunz-Jacques, Gwenaëlle Martinet, Frédéric Muller et Frédéric Valette. Power attack on small RSA public exponent. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 339–353. Springer-Verlag, 2006.

[FOPS01] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval et Jacques Stern. RSA-OAEP is secure under the RSA assumption. Dans Joe Kilian, éditeur, *Advances in Cryptology – CRYPTO '01*, volume 2139 de *Lecture Notes in Computer Science*, pages 260–274. Springer-Verlag, 2001.

[FP99] Paul N. Fahn et Peter K. Pearson. IPA : A new class of power attacks. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 de *Lecture Notes in Computer Science*, pages 173–186. Springer-Verlag, 1999.

[Gir03] Christophe Giraud. DFA on AES. *Cryptology ePrint Archive*, Report 2003/008, 2003.

[Gir04] Christophe Giraud. DFA on AES. Dans Hans Dobbertin, Vincent Rijmen et Aleksandra Sowa, éditeurs, *AES 4 Conference*, volume 3373 de *Lecture Notes in Computer Science*, pages 27–41. Springer-Verlag, 2004.

[Gir07] Christophe Giraud. *Attaques de cryptosystèmes embarqués et contre-mesures associées*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, France, octobre 2007.

[GLP06] Benedikt Gierlichs, Kerstin Lemke-Rust et Christof Paar. Templates vs. stochastic methods. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 15–29. Springer-Verlag, 2006.

[GM84] Shafi Goldwasser et Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GM97] Marc Girault et Jean-François Misarsky. Selective forgery of RSA signatures using redundancy. Dans Walter Fumy, éditeur, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 de *Lecture Notes in Computer Science*, pages 495–507. Springer-Verlag, 1997.

[GMO01] Karine Gandolfi, Christophe Mourtel et Francis Olivier. Electromagnetic analysis : Concrete results. Dans Çetin Kaya Koç, David Naccache et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '01*, volume 2162 de *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2001.

[Gol97] Oded Goldreich. On the foundations of modern cryptography. Dans Burton S. Kaliski, Jr, éditeur, *Advances in Cryptology – CRYPTO '97*, volume 1294 de *Lecture Notes in Computer Science*, pages 46–74. Springer-Verlag, 1997.

[GP99] Louis Goubin et Jacques Patarin. DES and differential power analysis (the "duplication" method). Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 de *Lecture Notes in Computer Science*, pages 158–172. Springer-Verlag, 1999.

[GT03] Jovan Dj. Golic et Christophe Tymen. Multiplicative masking and power analysis of AES. Dans Burton S. Kaliski, Jr, Çetin Kaya Koç et Christof Paar,

éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 de *Lecture Notes in Computer Science*, pages 198–212. Springer-Verlag, 2003.

[GT04] Christophe Giraud et Hugues Thiebeauld. A survey on fault attacks. Dans Jean-Jacques Quisquater, Pierre Paradinas, Yves Deswarte et Anas Abou El Kalam, éditeurs, *Smart Card Research and Advanced Application – CARDIS '04*, pages 159–176. Kluwer, 2004.

[GTV90] Marc Girault, Philippe Toffin et Brigitte Vallée. Computation of approximate L-th roots modulo n and application to cryptography. Dans Shafi Goldwasser, éditeur, *Advances in Cryptology – CRYPTO '88*, volume 403 de *Lecture Notes in Computer Science*, pages 100–117. Springer-Verlag, 1990.

[Hem04] Ludger Hemme. A differential fault attack against early rounds of (triple-)DES. Dans Marc Joye et Jean-Jacques Quisquater, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '04*, volume 3156 de *Lecture Notes in Computer Science*, pages 254–267. Springer-Verlag, 2004.

[HH99] Helena Handschuh et Howard M. Heys. A timing attack on RC5. Dans Stafford E. Tavares et Henk Meijer, éditeurs, *Selected Areas in Cryptography – SAC '98*, volume 1556 de *Lecture Notes in Computer Science*, pages 306–318. Springer-Verlag, 1999.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai et David Wagner. Private circuits II : Keeping secrets in tamperable circuits. Dans Serge Vaudenay, éditeur, *Advances in Cryptology – EUROCRYPT '06*, volume 4004 de *Lecture Notes in Computer Science*, pages 308–327. Springer-Verlag, 2006.

[JC86] Wiebren de Jonge et David Chaum. Attacks on some RSA signatures. Dans Hugh C. Williams, éditeur, *Advances in Cryptology – CRYPTO '85*, volume 218 de *Lecture Notes in Computer Science*, pages 18–27. Springer-Verlag, 1986.

[JLQ99] Marc Joye, Arjen K. Lenstra et Jean-Jacques Quisquater. Chinese remaindering based cryptosystems in the presence of faults. *Journal of Cryptology*, 12(4):241–245, 1999.

[JP06] Marc Joye et Pascal Paillier. Fast generation of prime numbers on portable devices : An update. Dans Louis Goubin et Mitsuru Matsui, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '06*, volume 4249 de *Lecture Notes in Computer Science*, pages 160–173. Springer-Verlag, 2006.

[JPV00] Marc Joye, Pascal Paillier et Serge Vaudenay. Efficient generation of prime numbers. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 de *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 2000.

[JQYY02] Marc Joye, Jean-Jacques Quisquater, Sung-Ming Yen et Moti Yung. Observability analysis – detecting when improved cryptosystems fail. Dans Bart Preneel, éditeur, *Topics in Cryptology – CT-RSA '02*, volume 2271 de *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, 2002.

[KJJ98] Paul C. Kocher, Joshua Jaffe et Benjamin Jun. Introduction to differential power analysis and related attacks. Rapport Technique, Cryptography Research, 1998. http://www.cryptography.com/resources/whitepapers/DPATechInfo.pdf.

[KJJ99] Paul C. Kocher, Joshua Jaffe et Benjamin Jun. Differential power analysis. Dans Michael J. Wiener, éditeur, *Advances in Cryptology – CRYPTO '99*, volume 1666 de *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[Knu81] Donald E. Knuth. *The Art of Computer Programming, Volume II : Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Dans Neal Koblitz, éditeur, *Advances in Cryptology – CRYPTO '96*, volume 1109 de *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[KR96] Joe Kilian et Phillip Rogaway. How to protect DES against exhaustive key search. Dans Neal Koblitz, éditeur, *Advances in Cryptology – CRYPTO '96*, volume 1109 de *Lecture Notes in Computer Science*, pages 252–267. Springer-Verlag, 1996.

[LH94] Susan K. Langford et Martin E. Hellman. Differential-linear cryptanalysis. Dans Yvo Desmedt, éditeur, *Advances in Cryptology – CRYPTO '94*, volume 839 de *Lecture Notes in Computer Science*, pages 17–25. Springer-Verlag, 1994.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra et László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LM04] Christopher LaFrieda et Rajit Manohar. Fault detection and isolation techniques for quasi delay-insensitive circuits. Dans *International Conference on Dependable Systems and Networks – DSN '04*, pages 41–50. IEEE Computer Society, 2004.

[LM91] Xuejia Lai et James L. Massey. Markov ciphers and differentail cryptanalysis. Dans Donald W. Davies, éditeur, *Advances in Cryptology – EUROCRYPT '91*, volume 547 de *Lecture Notes in Computer Science*, pages 17–38. Springer-Verlag, 1991.

[LSP04] Kerstin Lemke, Kai Schramm et Christof Paar. DPA on n-bit sized boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. Dans Marc Joye et Jean-Jacques Quisquater, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '04*, volume 3156 de *Lecture Notes in Computer Science*, pages 205–219. Springer-Verlag, 2004.

[Lyo96] Richard G. Lyons. *Understanding Digital Signal Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[MAM+03] Simon W. Moore, Ross J. Anderson, Robert D. Mullins, George S. Taylor et Jacques J. A. Fournier. Balanced self-checking asynchronous logic for smart card applications. *Microprocessors and Microsystems*, 27(9):421–430, 2003.

[Mat94a] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. Dans Tor Helleseth, éditeur, *Advances in Cryptology – EUROCRYPT '93*, volume 765 de *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994.

[Mat94b] Mitsuru Matsui. The first experimental cryptanalysis of the Data Encryption Standard. Dans Yvo Desmedt, éditeur, *Advances in Cryptology – CRYPTO '94*,

volume 839 de *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1994.

[May00] Rita Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 de *Lecture Notes in Computer Science*, pages 78–92. Springer-Verlag, 2000.

[MDS02] Thomas S. Messerges, Ezzy A. Dabbish et Robert H. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, 51(5):541–552, 2002.

[MDS99] Thomas S. Messerges, Ezzy A. Dabbish et Robert H. Sloan. Investigations of power analysis attacks on smartcards. Dans *WOST '99 : Proceedings of the USENIX Workshop on Smartcard Technology*, pages 151–162, Berkeley, CA, USA, 1999. USENIX Association.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 de *Lecture Notes in Computer Science*, pages 238–251. Springer-Verlag, 2000.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. Dans Hugh C. Williams, éditeur, *Advances in Cryptology – CRYPTO '85*, volume 218 de *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag, 1986.

[Mis97] Jean-François Misarsky. A multiplicative attack using LLL algorithm on RSA signatures with redundancy. Dans Burton S. Kaliski, Jr, éditeur, *Advances in Cryptology – CRYPTO '97*, volume 1294 de *Lecture Notes in Computer Science*, pages 221–234. Springer-Verlag, 1997.

[Mis98] Jean-François Misarsky. How (not) to design RSA signature schemes. Dans Hideki Imai et Yuliang Zheng, éditeurs, *Public Key Cryptography – PKC '98*, volume 1431 de *Lecture Notes in Computer Science*, pages 14–28. Springer-Verlag, 1998.

[MR04] Silvio Micali et Leonid Reyzin. Physically observable cryptography (extended abstract). Dans Moni Naor, éditeur, *Theory of Cryptography Conference – TCC '04*, volume 2951 de *Lecture Notes in Computer Science*, pages 278–296. Springer-Verlag, 2004.

[MRL05] Yannick Monnet, Marc Renaudin et Régis Leveugle. Hardening techniques against transient faults for asynchronous circuits. Dans *11th IEEE International On-Line Testing Symposium (IOLTS 2005)*, pages 129–134. IEEE Computer Society, 2005.

[MRL+05] Yannick Monnet, Marc Renaudin, Régis Leveugle, Sophie Dumont et Fraidy Bouesse. An asynchronous DES crypto-processor secured against fault attacks. Dans *15th IFIP International Conference on Very Large Scale Integration Systems (VLSI-SOC)*, pages 21–26, 2005.

[MRL+06a] Yannick Monnet, Marc Renaudin, Régis Leveugle, Nathalie Feyt, Pascal Moitrel et Félie M'Buwa Nzenguet. Practical evaluation of fault countermeasures on an asynchronous DES crypto processor. Dans *12th IEEE Internatio-*

nal On-Line Testing Symposium (IOLTS 2006), pages 125–130. IEEE Computer Society, 2006.

[MRL+06b] Yannick Monnet, Marc Renaudin, Régis Leveugle, Christophe Clavier et Pascal Moitrel. Case study of a fault attack on asynchronous DES crypto-processors. Dans Luca Breveglieri, Israel Koren, David Naccache et Jean-Pierre Seifert, éditeurs, *Fault Diagnosis and Tolerance in Cryptography – FDTC '06*, volume 4236 de *Lecture Notes in Computer Science*, pages 88–97. Springer-Verlag, 2006.

[Mui06] James A. Muir. Seifert's RSA fault attack : Simplified analysis and generalizations. Dans Peng Ning, Sihan Qing et Ninghui Li, éditeurs, *Information and Communications Security – ICICS '06*, volume 4307 de *Lecture Notes in Computer Science*, pages 420–434. Springer-Verlag, 2006.

[Mul04] Frédéric Muller. Differential attacks against the Helix stream cipher. Dans Bimal K. Roy et Willi Meier, éditeurs, *Fast Software Encryption – FSE '04*, volume 3017 de *Lecture Notes in Computer Science*, pages 94–108. Springer-Verlag, 2004.

[MOV97] Alfred Menezes, Paul C. van Oorschot et Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[MY91] Ueli M. Maurer et Yacov Yacobi. Non-interactive public-key cryptography. Dans Donald W. Davies, éditeur, *Advances in Cryptology – EUROCRYPT '91*, volume 547 de *Lecture Notes in Computer Science*, pages 498–507. Springer-Verlag, 1991.

[MY93] Ueli M. Maurer et Yacov Yacobi. A remark on a non-interactive public-key distribution system. Dans Rainer A. Rueppel, éditeur, *Advances in Cryptology – EUROCRYPT '92*, volume 658 de *Lecture Notes in Computer Science*, pages 458–460. Springer-Verlag, 1993.

[NBS77] National Bureau of Standards. Data Encryption Standard. Federal Information Processing Standard #46, 1977.

[NIST00] National Institute of Standards and Technology. Digital Signature Standard (DSS). Federal Information Processing Standard #186-2, 2000.

[NIST01] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standard #197, 2001.

[NNTW05] David Naccache, Phong Q. Nguyen, Michael Tunstall et Claire Whelan. Experimenting with faults, lattices and the DSA. Dans Serge Vaudenay, éditeur, *Public Key Cryptography – PKC '05*, volume 3386 de *Lecture Notes in Computer Science*, pages 16–28. Springer-Verlag, 2005.

[Nov03] Roman Novak. Side-channel attack on substitution blocks. Dans Jianying Zhou, Moti Yung et Yongfei Han, éditeurs, *Applied Cryptography and Network Security – ACNS '03*, volume 2846 de *Lecture Notes in Computer Science*, pages 307–318. Springer-Verlag, 2003.

[OS85] Tatsuski Okamoto et Akira Shibaishi. A fast signature scheme based on quadratic inequalities. Dans *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, pages 123–132, Oakland, CA, avril 1985.

[Osw03] Elisabeth Oswald. *On Side-Channel Attacks and the Application of Algorithmic Countermeasures*. PhD thesis, Faculty of Science of the University of Technology Graz (IAIK-TUG), Austria, juin 2003.

[Pai99] Pascal Paillier. Evaluating differential fault analysis of unknown cryptosystems. Dans Hideki Imai et Yuliang Zheng, éditeurs, *Public Key Cryptography – PKC '99*, volume 1560 de *Lecture Notes in Computer Science*, pages 235–244. Springer-Verlag, 1999.

[PKCS-1] RSA Laboratories. PKCS #1 : RSA cryptography specifications, version 2.0, septembre 1998.

[PM05] Thomas Popp et Stefan Mangard. Masked dual-rail pre-charge logic : DPA-resistance without routing constraints. Dans Josyula R. Rao et Berk Sunar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '05*, volume 3659 de *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, 2005.

[PP05] Souradyuti Paul et Bart Preneel. Solving systems of differential equations of addition. Dans Colin Boyd et Juan Manuel González Nieto, éditeurs, *Information Security and Privacy, 10th Australasian Conference, ACISP '05*, volume 3574 de *Lecture Notes in Computer Science*, pages 75–88. Springer-Verlag, 2005.

[PQ03] Gilles Piret et Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. Dans Colin D. Walter, Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '03*, volume 2779 de *Lecture Notes in Computer Science*, pages 77–88. Springer-Verlag, 2003.

[PT06] David Peacham et Byron Thomas. DFA against AES key expansion. CHES '06 Rump Session, 2006.

[QC82] Jean-Jacques Quisquater et Chantal Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, octobre 1982.

[QS00] Jean-Jacques Quisquater et David Samyde. A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions, the SEMA and DEMA methods. *Présenté à la rump session de EUROCRYPT '00*, Bruges, Belgique, mai 2000.

[Ren00] Marc Renaudin. Asynchronous circuits and systems : a promising design alternative. *Microelectronic Engineering Journal*, 54(1-2):133–149, décembre 2000.

[RO04] Christian Rechberger et Elisabeth Oswald. Practical template attacks. Dans Chae Hoon Lim et Moti Yung, éditeurs, *Workshop on Information Security Applications – WISA '04*, volume 3325 de *Lecture Notes in Computer Science*, pages 440–456. Springer-Verlag, 2004.

[RSA78] Ronald L. Rivest, Adi Shamir et Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sei05] Jean-Pierre Seifert. On authenticated computing and RSA-based authentication. Dans *ACM Conference on Computer and Communications Security*, pages 122–127, 2005.

[Sha00] Adi Shamir. Protecting smart cards from passive power analysis with detached power supplies. Dans Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '00*, volume 1965 de *Lecture Notes in Computer Science*, pages 71–77. Springer-Verlag, 2000.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. Dans G. R. Blakley et David Chaum, éditeurs, *Advances in Cryptology – CRYPTO '84*, volume 196 de *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, 1985.

[Sho01] Victor Shoup. OAEP reconsidered. Dans Joe Kilian, éditeur, *Advances in Cryptology – CRYPTO '01*, volume 2139 de *Lecture Notes in Computer Science*, pages 239–259. Springer-Verlag, 2001.

[SLFP04] Kai Schramm, Gregor Leander, Patrick Felke et Christof Paar. A collision-attack on AES : Combining side channel- and differential-attack. Dans Marc Joye et Jean-Jacques Quisquater, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '04*, volume 3156 de *Lecture Notes in Computer Science*, pages 163–175. Springer-Verlag, 2004.

[SMY06] Francois-Xavier Standaert, Tal G. Malkin et Moti Yung. A formal practice-oriented model for the analysis of side-channel attacks. *Cryptology ePrint Archive*, Report 2006/139, 2006.

[SP06] Kai Schramm et Christof Paar. Higher order masking of the AES. Dans David Pointcheval, éditeur, *Topics in Cryptology – CT-RSA '06*, volume 3860 de *Lecture Notes in Computer Science*, pages 208–225. Springer-Verlag, 2002.

[SWP03] Kai Schramm, Thomas J. Wollinger et Christof Paar. A new class of collision attacks and its application to DES. Dans Thomas Johansson, éditeur, *Fast Software Encryption – FSE '03*, volume 2887 de *Lecture Notes in Computer Science*, pages 206–222. Springer-Verlag, 2003.

[TFY07] Junko Takahashi, Toshinori Fukunaga et Kimihiro Yamakoshi. DFA mechanism on the AES key schedule. Dans Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache et Jean-Pierre Seifert, éditeurs, *Fault Diagnosis and Tolerance in Cryptography – FDTC '07*, pages 62–72. IEEE Computer Society Press, 2007.

[THH+05] Kris Tiri, David Hwang, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont et Ingrid Verbauwhede. Prototype IC with WDDL and differential routing - DPA resistance assessment. Dans Josyula R. Rao et Berk Sunar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '05*, volume 3659 de *Lecture Notes in Computer Science*, pages 354–365. Springer-Verlag, 2005.

[TSG03] Elena Trichina, Domenico de Seta et Lucia Germani. Simplified adaptive multiplicative masking for AES. Dans Burton S. Kaliski, Jr, Çetin Kaya Koç et Christof Paar, éditeurs, *Cryptographic Hardware and Embedded Systems – CHES '02*, volume 2523 de *Lecture Notes in Computer Science*, pages 187–197. Springer-Verlag, 2003.

[Tun06] Michael Tunstall. *Secure Cryptographic Algorithm Implementation on Embedded Platforms*. PhD thesis, Royal Holloway, University of London, 2006.

[Wal98] Colin D. Walter. Exponentiation using division chains. *IEEE Transactions on Computers*, 47(7):757–765, 1998.

[Yac91] Yacov YACOBI. Exponentiating faster with addition chains. Dans Ivan DAMGÅRD, éditeur, *Advances in Cryptology – EUROCRYPT '90*, volume 473 de *Lecture Notes in Computer Science*, pages 222–229. Springer-Verlag, 1991.

[YJ00] Sung-Ming YEN et Marc JOYE. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

[YKLM02] Sung-Ming YEN, Seungjoo KIM, Seongan LIM et Sang-Jae MOON. A countermeasure against one physical cryptanalysis may benefit another attack. Dans Kwangjo KIM, éditeur, *Information Security and Cryptology – ICISC '01*, volume 2288 de *Lecture Notes in Computer Science*, pages 414–427. Springer-Verlag, 2002.

Résumé

Dans un monde défiant, l'augmentation du nombre et de la diversité des applications numériques ont rendu nécessaire l'existence d'un objet pratique intégrant les fonctions cryptographiques requises pour les besoins quotidiens de sécurité des transactions, de confidentialité des échanges, d'identification du porteur ou encore d'authentification pour l'accès à un service. Parmi les dispositifs cryptographiques embarqués aptes à proposer ces fonctionnalités, la carte à puce est certainement le plus utilisé de nos jours. Sa portabilité (un porte-feuille peut en contenir une dizaine) et sa capacité à protéger les données et programmes qu'elle contient contre les attaques intrusives, lui confèrent naturellement sa fonction essentielle de "bunker" pour le stockage de clés et l'exécution d'algorithmes cryptographiques dans les usages mobiles nécessitant un haut degré de sécurité.

Évidemment nécessaire, la conception de schémas cryptographiques mathématiquement robustes, voire prouvés sûrs dans certains modèles, s'est malgré tout révélée insuffisante depuis la publication en 1996 des premières attaques physiques. Exploitant des vulnérabilités liées à la mise en œuvre concrète des routines de sécurité et à leur implémentation, ces menaces comprennent l'analyse de canaux auxiliaires permettant d'obtenir de l'information sur l'état interne d'un processus, et l'exploitation de fautes provoquées ouvrant la voie à certaines cryptanalyses autrement impossibles.

Cette thèse présente une série de travaux de recherche dans le domaine *de la sécurité physique des crypto-systèmes embarqués*. Deux parties de ce document sont consacrées à la description de certaines attaques et à l'étude de l'efficacité de possibles contre-mesures. Une troisième partie aborde le domaine particulier, et encore très peu exploré, de l'applicabilité des attaques physiques dans le cas où la fonction cryptographique considérée est en grande partie, voire totalement, inconnue de l'adversaire.

Abstract

In a world full of threats, the development of widespread digital applications has led to the need for a practical device containing cryptographic functions that provide the everyday needs for secure transactions, confidentiality of communications, identification of the subject or authentication for access to a particular service. Among the cryptographic embedded devices ensuring these functionalities, smart cards are certainly the most widely used. Their portability (a wallet may easily contain a dozen) and their ability to protect its data and programs against intruders, make it as the ideal "bunker" for key storage and the execution of cryptographic functions during mobile usage requiring a high level of security.

Whilst the design of mathematically robust (or even proven secure in some models) cryptographic schemes is an obvious requirement, it is apparently insufficient in the light of the first physical attacks that were published in 1996. Taking advantage of weaknesses related to the basic implementation of security routines, these threats include side-channel analysis which obtains information about the internal state of the process, and the exploitation of induced faults allowing certain cryptanalysis to be performed which otherwise would not have been possible.

This thesis presents a series of research works covering *the physical security of embedded cryptosystems*. Two parts of this document are dedicated to the description of some attacks and to a study of the efficiency of conceivable countermeasures. A third part deals with that particular and still mainly unexplored area which considers the applicability of physical attacks when the cryptographic function is, partly or totally, unknown by the adversary.