Dimensionnement et intégration d'un chiffre symétrique dans le contexte d'un système d'information distribué de grande taille.

Thomas Roche

UNIVERSITÉ DE GRENOBLE

N° attribué par la bibliothèque

THÈSE

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : "Mathématique et Informatique"

PRÉPARÉE AU LABORATOIRE D'INFORMATIQUE DE GRENOBLE
DANS LE CADRE DE ***l'École Doctorale "Mathématiques, Sciences et Technologies de l'Information, Informatique"***

PRÉSENTÉE ET SOUTENUE PUBLIQUEMENT

PAR

THOMAS ROCHE

LE 29 JANVIER 2010

DIMENSIONNEMENT ET INTÉGRATION D'UN CHIFFRE SYMÉTRIQUE DANS LE CONTEXTE D'UN SYSTÈME D'INFORMATION DISTRIBUÉ DE GRANDE TAILLE

Thèse dirigée par M Roland Gillard ***et co-dirigée par*** M Jean-Louis Roch

JURY

DANIEL AUGOT	INRIA, LIX	Rapporteur
JEAN-CLAUDE BAJARD	LIP6	Rapporteur
ROLAND GILLARD	Institut Fourier	Examinateur
LOUIS GOUBIN	PRiSM	Examinateur
SYLVAIN GRAVIER	Institut Fourier	Examinateur (président du jury)
JEAN-LOUIS ROCH	INRIA, INPG, LIG	Examinateur

A la mémoire de Samy.

Remerciements

C'est d'abord à ceux qui ont rendu cette thèse possible que j'adresse mes premiers remerciements : mes deux directeurs de thèse : **Roland Gillard** pour avoir garder un oeil attentif et constant sur le déroulement de mon travail, même après son départ en retraite et **Jean-Louis Roch** pour l'encadrement, jour après jour, de mes travaux et pour sa passion contagieuse de la recherche ; merci ensuite à l'entreprise Communication et Système pour le financement de cette thèse, notamment **Pierre Herchuelz** et **Jean-Michel Tenkes** qui ont soutenu mon travail et supporté une composante de recherche dans un projet industriel.

Je voudrais remercier les membres du jury de m'avoir honoré de leur présence : **Sylvain Gravier**, directeur de recherche à l'Institut Fourier, pour avoir accepté de présider le jury ; **Daniel Augot**, directeur de recherche à l'INRIA Saclay et **Jean-Claude Bajard**, professeur au LIP6, pour avoir rapporté ce manuscrit et pour les nombreuses remarques qu'ils ont pu apporter, témoignant de leur intérêt pour le sujet traité ; enfin, **Louis Goubin**, professeur à l'Université de Versailles, laboratoire PRiSM, d'être venu assister à la soutenance et d'avoir partagé son expérience dans le domaine des attaques par canaux cachés.

Ces trois années de thèse m'auront permis de croiser le chemin de nombreuses personnes qui ont eu divers influences sur mon travail :

- les membres du laboratoire LIG et en particulier la grande famille que constitue l'ex laboratoire ID-Imag. Bien que mon sujet de recherche m'ait rapidement éloigné des problématiques premières des équipes du laboratoire, je me suis senti, pendant ces trois ans, à ma place au sein de ID. Je voudrait notamment citer **Maxime Martinasso**, **Guilaume Huard**, **Pedro Velho** et **Xavier Besseron** qui sont devenus aujourd'hui des amis proches.
- ma rencontre avec **Cédric Tavernier**, expert en cryptographie chez Communication et Systèmes à l'époque, a participé grandement à la réussite de cette thèse. Son simple contact m'a appris beaucoup sur la vie du chercheur et sur la passion qu'il faut à celui-ci. C'est par son intermédiaire que j'ai rencontré **Emmanuel Prouff**, chercheur en cryptographie chez Oberthur Technologies, qui n'a jamais hésité à nous faire profiter de son savoir et ainsi nous faire avancer dans nos recherches.
- les personnes que j'ai rencontrées à l'occasion de conférences ou de séminaires, il n'est pas possible d'en faire une liste ici mais je tiens à remercier tout particulièrement : **Pascal Lafourcade**, maître de conférences à l'Université Joseph Fourier, qui m'a permis de partir 1 mois en visite à l'Université de Waterloo, Ontario, Canada, travailler sur les attaques par injections de fautes auprès du professeur **Anwar Hasan** ; **Claude Carlet**, professeur de mathématiques à l'Université Paris 8, pour nos nombreux et enrichissants échanges ; **Pierre-Louis Cayrel**, post-doc au CASED, Darmstadt, pour son dynamisme et sa connaissance de la communauté crypto française et internationale ; et bien d'autres, ...

Enfin, pour m'avoir suivi et m'avoir supporté pendant les périodes de doutes, je remercie chaleureusement :

- ma famille bien sûr, et en particulier mon père pour la relecture soigneuse du manuscrit.
- mes amis de Grenoble : Anne Lise (merci pour la relecture), Anne-Laure et Julien, Antoine et Mathilde, Céline, Éliane, Émilie, Guillaume et Valentina, Lionel, Nicolas et Maïté, Matthieu (bon courage pour les derniers mois) et Sophie, Pedro et Patricia, Sylvain, Xavier (bientôt la fin pour toi aussi) et Nia, Zeina et Alaric.

Table des matières

1 Introduction **1**

I CONTEXTE, PRÉLIMINAIRES ET OUTILS **7**

2 Contexte et Préliminaires **9**
2.1 Systèmes d'information distribués de grande taille 9
2.1.1 Premières définitions 9
2.1.2 Architectures des systèmes distribués 10
2.2 Sûreté de fonctionnement : Définition de la confiance 16
2.2.1 Attributs 16
2.2.2 Entraves 17
2.2.3 Moyens 18

3 Outils codage et cryptographie **19**
3.1 Cryptographie à clé secrète : les chiffrements par blocs 20
3.1.1 Notions de sécurité et théorie de Shannon 20
3.1.2 Schéma général d'un chiffrement par bloc : Réseau de Substitution et Permutation (SP-Network) 22
3.1.3 Techniques de cryptanalyse théorique et résistance 25
3.1.4 Attaques par canaux auxiliaires et contre-mesures 33
3.2 Théorie des codes 38
3.2.1 Préliminaires 38
3.2.2 Codes de Reed-Muller 40
3.2.3 Codes de Reed-Solomon 43
3.2.4 Codes par interpolation de polynômes univariés 47
3.2.5 Codes LDPC 47

II CRYPTOLOGIE À CLÉ SECRÈTE **49**

4 Primitives cryptographiques : Traitement parallèle **51**
4.1 Stratégie de parallélisme pour chiffrements symétriques et hachage cryptographique 52
4.1.1 Chiffrement par bloc 52

4.1.2 Fonction de Hachage 55
4.1.3 Conclusions 58
4.2 Implémentation efficace de codes parallèles 60
4.2.1 Principe du vol de travail 60
4.2.2 Ordonnancement dynamique de tâches par vol de travail 60
4.3 Application aux chiffrements symétriques avec mode opératoire Compteur 61

5 Preuve de Résistance à la cryptanalyse par différentielles impossibles 65
5.1 Préliminaires 65
5.1.1 Sécurité prouvée face aux attaques différentielles 66
5.1.2 Cryptanalyse par différentielle impossible 67
5.2 Vers la sécurité prouvée face aux attaques par différentielles impossibles 68
5.2.1 Matériel et Notations 68
5.2.2 Bornes supérieures et inférieures de EDP 72
5.2.3 Résultats pour AES et CS-Cipher 77

6 Attaques par analyse de courant et approximations linéaires 81
6.1 Préliminaires 82
6.1.1 Modèle d'exécution d'un chiffre symétrique 82
6.1.2 Approximations linéaires 84
6.2 Attaques par analyse de consommation basées sur les approximations linéaires . 86
6.2.1 Motivation 86
6.2.2 Une DPA Probabiliste 87
6.2.3 Attaques Multi-linéaires par Analyse de Courant (MLAC) 88
6.2.4 Résultats d'attaques MLAC 91
6.2.5 Pour aller plus loin 92

7 CS-Cipher 95
7.1 Présentation 95
7.1.1 CS-Cipher 95

III TOLÉRANCE AUX FAUTES, CERTIFICATION DE RÉSULTATS 103

8 Tolérance aux Fautes 105
8.1 La tolérance aux fautes : Principes de base et Taxinomie 105
8.1.1 Classes de fautes 106
8.1.2 Classes de défaillances 106
8.1.3 Techniques de tolérance aux fautes 106
8.2 Tolérance aux fautes dans les systèmes distribués 108
8.2.1 Systèmes distribués 109
8.2.2 Modèles de Fautes 109
8.2.3 Techniques de tolérance aux fautes 110
8.2.4 Cas des systèmes distribués de grande taille 111

9 Algorithmes auto-tolérants aux fautes sur système de calcul global. Application au produit de matrices **113**
9.1 Notations et premières définitions . . . 114
9.1.1 Coût de calcul sur plate-forme parallèle . . . 114
9.1.2 Fonction d'impact et capacité de tolérance aux fautes . . . 116
9.2 Tolérance aux fautes dans les systèmes de calcul globaux . . . 117
9.2.1 Description générale du système de calcul global et analyse de la problématique . . . 117
9.2.2 Schéma général du calcul tolérant aux fautes . . . 121
9.2.3 Choix de codes correcteurs et influence sur R . . . 125
9.3 Application au produit de matrices et premiers résultats . . . 130
9.3.1 Algorithmes auto-tolérants aux fautes . . . 130
9.3.2 Utilisation de codes systématiques ou creux et conséquence sur U . . . 133
9.4 Produit de matrices et codes linéaires (ABFT) . . . 134
9.4.1 ABFT : Définition et état de l'art . . . 134
9.4.2 Application au produit de matrices . . . 137
9.5 Produit de matrices et codes par résidus . . . 138
9.5.1 Schéma général et état de l'art . . . 139
9.5.2 Codes par résidus pour le produit de matrices sur des systèmes de calcul globaux . . . 142
9.6 Synthèse des résultats . . . 144
9.6.1 Récapitulation des résultats . . . 144
9.6.2 Application de l'algorithme ABFT pour le produit de matrices sur réseau pair-à-pair . . . 144

10 Conclusion **149**

Bibliographie **151**

IV ANNEXES **167**

A Preuves des théorèmes 7 et 8 **169**

B Attaque MLAC sur traces de consommation du DPA-context **173**
B.1 Mise en oeuvre de l'attaque . . . 173
B.2 Simulations . . . 178

C Coût des opérations arithmétiques **181**

Table des figures

1.1 Diagramme des dépendances entre les chapitres introductifs et chapitres de contributions 4

2.1 Le Nuage 15
2.2 L'arbre de la sûreté de fonctionnement ([9]) 17

3.1 Schéma général d'un chiffrement itératif 23
3.2 Schéma d'une ronde de type réseau de substitution-permutation 23
3.3 Consommation en fonction de la distance de Hamming 35

4.1 Mode Opératoire ECB 53
4.2 Mode Opératoire CBC 54
4.3 Mode Opératoire CTR 54
4.4 Mode Opératoire arbre de Merkle 57
4.5 Mode Opératoire Arbre de Merkle 58
4.6 Gain en temps d'exécution sans coût de lecture 62
4.7 Gain en temps d'exécution avec coût de lecture 62
4.8 Gain en temps d'exécution sur plate-forme homogène 63
4.9 Gain en temps d'exécution sur plate-forme hétérogène 63

5.1 Schéma général d'un algorithme de chiffrement clé-alternant [63] 69
5.2 Structure SPN. 70
5.3 Structure PSN. 70

6.1 Chiffre Symétrique Itératif 83
6.2 Résultat d'attaque MLAC sur les traces du concours DPA 93

7.1 Schéma global de CS-Cipher (Tirée de [205]) 97
7.2 Fonction F_{c^i} dans la génération de clés de rondes (Tirée de [205]) 98
7.3 Fonction de Ronde E de CS-Cipher 98
7.4 Fonction M de CS-Cipher 99
7.5 Boîte de Substitution de CS-Cipher (P) 99
7.6 Fonction de Ronde de CSC^* 101

8.1 Chaîne fondamentale des entraves à la sûreté de fonctionnement ([9]) 106

9.1 Calcul sur système distribué de grande taille 117
9.2 Schéma général d'un algorithme tolérant les fautes sur un système de calcul global $\{R, U\}$ 124
9.3 ABFT itératif pour le produit de matrices 139
9.4 Protocole de tolérance aux fautes pour le produit de matrices sur réseau pair-à-pair 147

B.1 Schéma d'implémentation du chiffre DES (tiré de [91]) 174
B.2 Répartition des mesures de courant au point 6374 en fonction de $H(V_2)$ 176
B.3 Répartition des mesures de courant au point 14491 en fonction de $H(V_{15})$ 176
B.4 Complexité de l'attaque MLAC sur AES et DES en simulation 178
B.5 Complexité de l'attaque sur les traces du dpa-contest 179
B.6 Complexité de l'attaque en simulation 179
B.7 Complexité de l'attaque MLAC sur le DES, comparaison entre traces réelles et simulations 179

Liste des Algorithmes

1 Mode Compteur, algorithme séquentiel 55
2 Mode Compteur, algorithme parallèle statique 56
3 Arbre de Merkle, algorithme séquentiel 59
4 Arbre de Merkle, algorithme récursif 60
5 Certification par post-condition pour le produit de matrices 120
6 Produit de matrices par produit externe 121
7 Algorithme générique tolérant aux fautes avec certification de résultat 123
8 Algorithme auto-tolérant aux fautes par codage de la matrice. 131
9 Algorithme tolérant aux fautes par codage des colonnes de la matrice. 132
10 Algorithme tolérant aux fautes par codage des composantes de la matrice. 132
11 ABFT pour produit de matrices 138
12 Algorithme auto tolérant aux fautes à l'aide de codes par résidus. 143

Liste des tableaux

6.1 Résultat d'attaque MLAC sur les traces du concours DPA : résumé 93

7.1 CS-Cipher : Paramètres Globaux 96
7.2 Fonction f 99
7.3 Fonction g 99

9.1 Complexité des algorithmes tolérants aux fautes sur R en fonction du code correcteur 129
9.2 Capacité de détection/correction des algorithmes tolérants aux fautes en fonction du code correcteur 129
9.3 Complexité des algorithmes auto-tolérants aux fautes 133
9.4 Complexité des algorithmes auto-tolérants aux fautes optimisés 145

B.1 Approximations linéaires de $< H(V_2(x,k)), \Gamma_H >$ 177

C.1 Tableau récapitulatif de complexité des opérations arithmétiques $n = \log q$ 182

Introduction 1

Les systèmes d'information prennent peu à peu une importance colossale : la façade physique des institutions privées et publiques, des réseaux sociaux, s'estompe pour être supportée par des systèmes d'information complexes dont les portes d'entrées se trouvent sur Internet. Les systèmes d'information de grande taille permettent aujourd'hui un accès à de grandes tailles de données et offrent leurs services à un nombre d'utilisateurs toujours plus important. Pour ce faire, la puissance de calcul et la capacité de stockage de ces systèmes sont *réparties* (ou *distribuées*) sur un grand nombre de processeurs reliés entre eux par l'intermédiaire de réseaux de télécommunication. La délocalisation des ressources et leur caractère potentiellement hétérogène ont un impact significatif sur la *confiance* qu'un utilisateur peut avoir dans de tels systèmes. La thématique de la confiance dans les *systèmes d'information distribués de grande taille* est le coeur de cette thèse, elle a été abordée selon deux axes de recherche distincts : l'étude des primitives cryptographiques et plus particulièrement des *chiffres par blocs* et l'étude de la *tolérance aux fautes* d'un système distribué.

Les primitives cryptographiques constituent les briques fondamentales de la sécurité d'un système face aux attaques extérieures, elles permettent d'assurer, entre autres, la confidentialité et l'intégrité des données, l'authentification ou encore l'anonymat des utilisateurs. Ces objets mathématiques, et parmi eux, tout particulièrement les chiffres symétriques, sont utilisés de façon intensive dans les systèmes d'information modernes. Leur *intégration* soulève un certain nombre de problèmes en terme de

- *performance* : ils ne doivent pas ralentir le système à délivrer son service.
- *dimensionnement* : ils doivent s'adapter aux différents types de ressources et en particulier aux systèmes embarqués fortement contraints.
- *sécurité* : ils sont généralement considérés comme inconditionnellement sûrs par le concepteur d'un système d'information, ce n'est pourtant jamais le cas en réalité. Savoir évaluer la résistance d'un chiffre face aux méthodes de cryptanalyse permet à la fois de quantifier la confiance que l'on peut mettre dans le système global et de concevoir des chiffres adaptés aux ressources du système.

La capacité d'un système à délivrer un service fiable n'est pas seulement dépendante de la bonne utilisation de fonctions cryptographiques, celles-ci permettant principalement de faire face à de potentielles agressions extérieures. Il s'agit aussi de rendre le système robuste face à ses propres défaillances, défaillances qui sont d'autant plus nombreuses que le système est de grande dimension. Les systèmes que nous étudions dans cette thèse possèdent des ressources hétérogènes évoluant dans un environnement peu ou pas contrôlé ; éventuellement administrées par des entités

différentes, et donc pour lesquelles l'utilisateur ne peut avoir une confiance aveugle. Le système doit donc être capable d'offrir un service correct en présence de défaillances. La *tolérance aux fautes* représente l'ensemble des techniques développées à cette fin. L'étude de la tolérance aux fautes faite dans cette thèse porte sur un cas d'application particulier : les plate-formes distribuées de grande taille utilisées pour le calcul scientifique et, plus particulièrement, le calcul du produit de matrices.

Contexte scientifique et industriel

Cette thèse s'inscrit dans le cadre d'un partenariat *CIFRE* entre le Laboratoire d'Informatique de Grenoble (le *LIG*) et l'entreprise Communication et Systèmes (*C-S*).

Les travaux de recherche se sont déroulés au sein de l'équipe MOAIS (équipe-projet INRIA) du LIG. MOAIS étudie la programmation d'applications pour systèmes distribués.

C-S est concepteur, intégrateur et opérateur de système critiques, cette thèse a été menée dans le cadre d'un projet industriel, dont l'objectif est le développement de solutions de sécurité dans un système d'information global.

Organisation du manuscrit et contributions

Hormis ce chapitre d'introduction et un chapitre de conclusions et perspectives sur l'ensemble du mémoire, le manuscrit est divisé en 7 chapitres structurés en trois parties.

La première partie intitulée "Contexte, Préliminaires et Outils" comporte deux chapitres :

- Une introduction au contexte et aux domaines d'application de la thèse (chapitre 2) : tout d'abord une présentation des systèmes d'information distribués de grande taille, des cas classiques d'applications et des menaces auxquelles ils doivent faire face. Ensuite une première approche de la notion de *sûreté de fonctionnement* d'un système nous permet de dégager concrètement les problématiques abordées par la thèse ;
- Le chapitre 3 a pour but de présenter, de façon cohérente, des notions et outils fondamentaux de la cryptographie et de la théorie des codes qui trouveront leur utilité dans les domaines les plus variés présentés dans ce mémoire.

La seconde partie est intitulée "Cryptographie à clé secrète", elle rassemble quatre chapitres autour de l'étude des chiffres par blocs :

- Tout d'abord dans le chapitre 4 sur des aspects de performance des chiffres symétriques par blocs (ainsi que certains schémas de fonctions de hachage) : l'exécution parallèle sur machines multi-coeurs et multi-processeurs est étudiée. L'utilisation d'un ordonnancement de tâches adaptatif par vol de travail permet d'observer des performances proches de l'optimum dans un environnement hétérogène d'unités de calcul et de masquer le coût de la fonction cryptographique. Ces résultats ont en partie été publié dans l'article [1] ;
- Le chapitre 5 se penche sur une technique classique de cryptanalyse des chiffres par blocs : la cryptanalyse par différentielles impossibles. L'utilisation d'un modèle original englobant les chiffres de Markov nous permet d'évaluer, sous certaines conditions, la résistance d'un

chiffre de type PSN ou SPN par rapport à ces attaques : de nouvelles bornes inférieures et supérieures de la probabilité d'occurrence d'une différentielle sont exhibés pour les chiffres CS-Cipher et AES. L'idée maîtresse du chapitre et une partie des résultats ont été publiés dans l'article [2] ;
- Le chapitre 6 est dédié à la présentation d'une nouvelle attaque par canaux cachés sur les chiffres par blocs, elle est appliquée à une implémentation matérielle du chiffre DES. Cette attaque permet de tisser un nouveau lien entre les attaques par canaux cachés et la cryptanalyse conventionnelle puisque l'attaque proposée utilise les méthodes de la cryptanalyse multi-linéaire classique et d'observer d'un autre point de vue l'utilité des codes correcteurs dans le domaine des attaques par canaux cachés. L'article [4], en attente de publication, reprend une grande partie du contenu de ce chapitre ;
- Enfin, le chapitre 7 étudie le chiffre CS-Cipher, chiffre par bloc cible de la thèse. Cette étude est limitée ici à une rapide présentation du chiffre original de Stern et Vaudenay, les résultats des recherches faites sur le sujet étant gardés confidentiels.

La troisième et dernière partie s'intitule "Tolérance aux fautes", elle est composée de deux chapitres :

- Le chapitre 8 introduit la taxinomie relative aux techniques de tolérance aux fautes ainsi que les modèles de fautes pour les systèmes distribués et un état de l'art des solutions existantes ;
- Le chapitre 9 se concentre sur la tolérance aux fautes construite au niveau algorithmique pour les plate-formes de calcul distribuées de grande taille et son application au calcul du produit de matrices. La construction d'algorithmes tolérants les défaillances se base sur l'utilisation d'information redondante dans la sortie du calcul. Ainsi, sur un modèle de plate-forme de calcul très peu restrictif, l'utilisation de codes correcteurs efficaces est étudiée, ainsi que la mise en oeuvre de deux techniques classiques : l'*ABFT* ("Algorithm-Based Fault Tolerance") et les *systèmes de nombre résiduel*. Une partie des résultats présentés dans ce chapitre a été publiée dans l'article [3].

Sur la figure 1.1, le schéma des dépendances entre chapitres offre une vue d'ensemble du manuscrit et illustre le lien, au niveau des notions fondamentales, entre les chapitres d'introduction et d'état de l'art (partie I) et les chapitres de contributions (partie II et III).

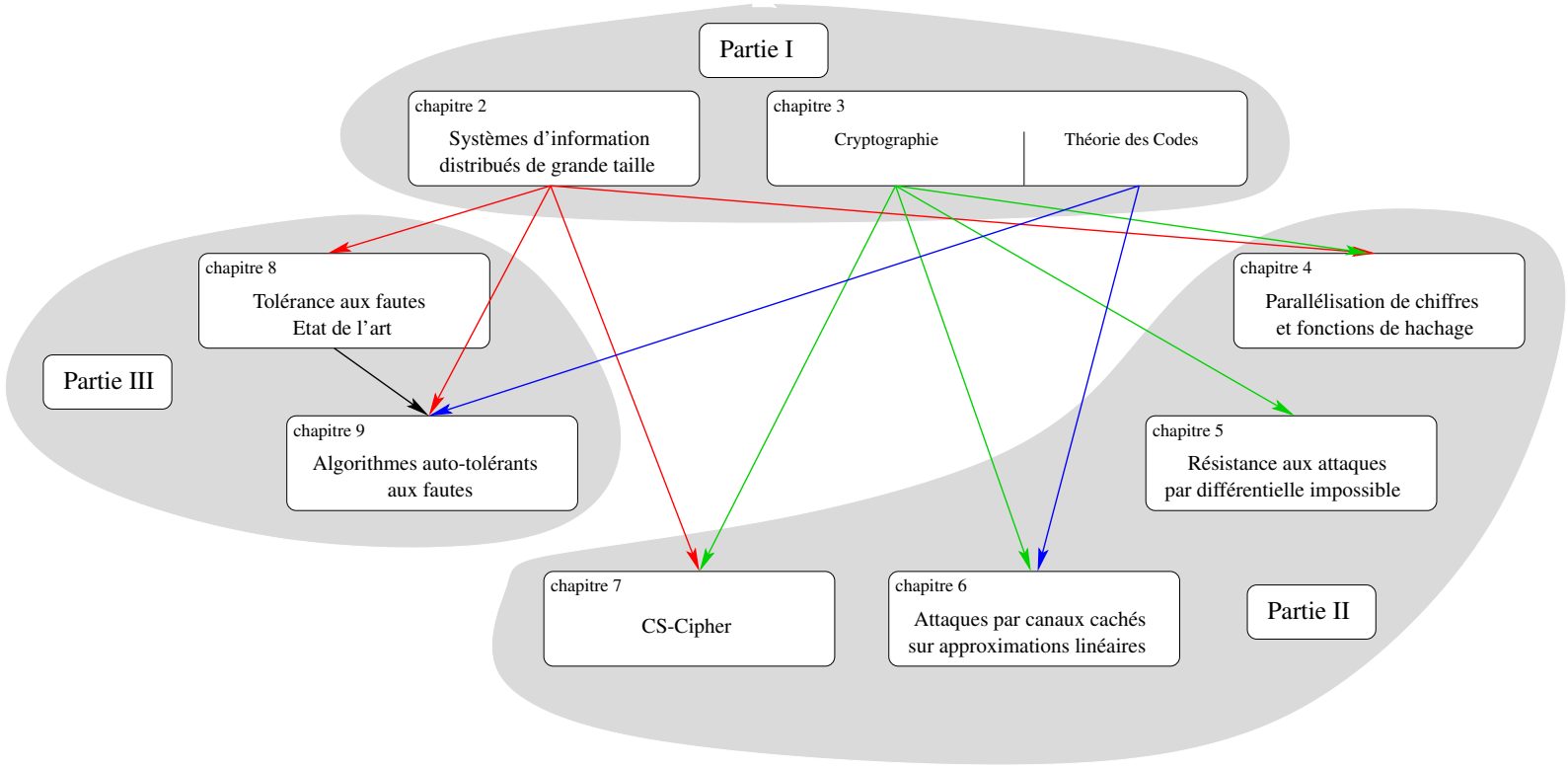

FIG. 1.1 – Diagramme des dépendances entre les chapitres introductifs et chapitres de contributions

Liste des publications

[1] Vincent DANJEAN, Roland GILLARD, Serge GUELTON, Jean-Louis ROCH et Thomas ROCHE : Adaptive loops with kaapi on multicore and grid : Applications in symmetric cryptography. *In* ACM PUBLISHING, éditeur : *Parallel Symbolic Computation'07 (PASCO'07)*, London, Ontario, Canada, July 2007.

[2] Thomas ROCHE, Roland GILLARD et Jean-Louis ROCH : Provable security against impossible differential cryptanalysis application to cs-cipher. *In* Le Thi Hoai AN, Pascal BOUVRY et Pham Dinh TAO, éditeurs : *MCO*, volume 14 de *Communications in Computer and Information Science*, pages 597–606. Springer, 2008.

[3] T. ROCHE, M. CUNCHE et J-L. ROCH : Algorithm-based fault tolerance applied to p2p computing networks. *In Advances in P2P Systems (AP2PS'09), Sliema, Malta. IEEE Computer Society*, 11-16 Oct. 2009.

[4] T. ROCHE et C. TAVERNIER : Side-channel attacks based on linear approximations. preprint : `http://eprint.iacr.org/2009/269.pdf`.

I

Contexte, Préliminaires et Outils

Contexte et Préliminaires 2

Sommaire

2.1 Systèmes d'information distribués de grande taille 9
2.1.1 Premières définitions 9
2.1.2 Architectures des systèmes distribués 10
2.1.2.1 Les "noeuds" du réseau 10
2.1.2.2 Les clusters 12
2.1.2.3 Les grilles 13
2.1.2.4 Les nuages 15
2.2 Sûreté de fonctionnement : Définition de la confiance 16
2.2.1 Attributs 16
2.2.2 Entraves 17
2.2.3 Moyens 18

Dans ce premier chapitre nous établissons le contexte de cette thèse, premièrement en définissant ce que nous entendrons, à travers tout ce mémoire, par *système d'information distribué de grande taille* et en identifiant les menaces auxquels ils doivent faire face, deuxièmement en introduisant les notions basiques de *sûreté de fonctionnement* qui servirons de cadre à nos travaux.

2.1 Systèmes d'information distribués de grande taille

2.1.1 Premières définitions

Le terme *système d'information distribué de grande taille* contraint le système cible sous trois composantes complémentaires :

- Un *système d'information* est pour nous un système informatique en cela qu'il fait intervenir des moyens informatiques pour délivrer un service à des utilisateurs (humains ou machines). De façon très générale, un tel système est attaché à une organisation (groupe structuré d'individus) et a pour objet de traiter, stocker ou communiquer les informations relatives à cette organisation.
- "Un *système distribué* est un système informatique composé de plusieurs ressources de calcul indépendantes agissant du point de vue de ses utilisateurs comme un seul et unique système cohérent", A. S. Tanenbaum [212].

– Un *système de grande taille* est un système composé d'un grand nombre de ressources. Un tel système manipulera une grande quantité de données et/ou devra interagir avec un grand nombre d'utilisateurs.

Ces définitions donnent un cadre à notre système cible, pour rendre ce cadre plus pragmatique nous proposons maintenant quelques applications réelles de ce type de systèmes.

2.1.2 Architectures des systèmes distribués

Depuis l'avènement de l'informatique et des télécommunications, les systèmes d'information sont en évolution constante : il faut composer avec une demande toujours croissante en nombre d'utilisateurs, taille de données et complexité des applications. Cela se traduit, en terme de moyens informatiques, en une augmentation des débits réseaux, des capacités de stockage et puissances de calcul.
Les systèmes distribués sont la dernière évolution répondant à ces besoins. Des limites physiques (dissipation d'énergie thermique) sont bientôt atteintes par nos microprocesseurs actuels en terme de vitesse et tailles de transistors et donc en terme de puissance de calcul. La multiplication de ces microprocesseurs reliés via un réseau apporte une solution permettant la poursuite de la croissance des performances, avant un éventuel bouleversement dans la technologie des processeurs, par exemple avec l'émergence d'*ordinateurs quantiques*.

Bien qu'ils offrent à la fois, des débits réseaux, des capacités de stockage et une puissance de calcul démultipliés, la mise en oeuvre de systèmes distribués est délicate. Les *intergiciels* ("middleware" en anglais) notamment, dont le rôle est de gérer le système de façon transparente pour l'utilisateur, doivent à la fois être fiables et efficaces.

Plusieurs types d'architectures sont utilisés suivant les applications et fonctionnalités du système, elle vont d'architectures complètement structurées suivant une hiérarchie statique des ressources (typiquement le modèle de type client/serveur d'un réseau de téléphonie mobile ou d'un réseau bancaire) jusqu'aux architectures entièrement destructurées où chaque ressource peut, à loisir, prendre le rôle du client ou du serveur (typiquement les réseaux pair-à-pair décentralisés d'échange de données ; par exemple Freenet[1] ou le protocole Gnutella supporté par plusieurs clients de réseaux pair-à-pair, entre autres, Gnucleus[2]).

2.1.2.1 Les "noeuds" du réseau

Avant de présenter quelques architectures type des infrastructures distribuées, faisons un rapide zoom sur les noeuds de ces systèmes, c'est à dire les unités de calcul connectées en réseau. Ces unités de calcul comporteront nécessairement un ou plusieurs processeurs, des blocs de mémoire et une *carte réseau* permettant de se connecter au réseau. Dans la suite nous désignerons par *station de travail* les noeuds possédant une interface avec l'humain (par le biais d'un écran et d'un

[1] `http://freenetproject.org/`
[2] `http://www.gnucleus.com`

clavier par exemple).

Les noeuds peuvent prendre des formes très différentes qu'il convient de regarder plus attentivement. En effet les unités de calcul internes à un noeud vont du simple processeur à une multitude d'entre eux dans le cas d'un *supercalculateur*.

Processeurs multi-coeurs Tout d'abord les processeurs peuvent comporter plusieurs coeurs de calculs identiques. Dans la même puce de silicium, est gravé un ensemble d'unités de calcul (jusqu'à plusieurs dizaines à l'heure actuelle) connectées entre elles et partageant un ou plusieurs niveaux de cache. Les machines de bureau actuelles sont couramment composées de processeurs multi-coeurs.

Multi-processeurs sur puce Suivant la même idée, les MPSoC ("Multi-Processors System On Chip" en anglais), rassemblent plusieurs processeurs (a priori différents) sur la même puce. Cela permet de faire cohabiter des unités de calcul dédiées à des applications particulières liées entre elles par des connections internes à la puce (donc très rapides).

Multi-processeurs Enfin plusieurs processeurs (e.g. multi-coeurs ou MPSoC) peuvent être reliés entre eux sur une même carte via un bus interne ou entre plusieurs cartes via un bus externe (le bus *PCI* "Peripheral Component Interconnect") ou un réseau interne (comme souvent dans les supercalculateurs où le nombre de processeurs est tel qu'un réseau haute performance dédié est utilisé, par exemple dans le Blue Gene d'IBM).
Les machines de bureau actuelles utilisent souvent le bus PCI pour connecter un processeur graphique (*GPU*) ou physique (*PPU*) soulageant le processeur principal (*CPU*) pour des opérations dédiées.

Remarque 1 *Les noeuds d'un système peuvent être eux-mêmes des systèmes distribués, créant ainsi plusieurs niveau de parallélisme et de distribution de l'information à l'intérieur du système distribué global.*

Menaces Nous considérons ici un noeud sans se soucier du système distribué auquel il appartient ; à ce niveau, nous ne considérons pas de comportement malveillant, seulement la possibilité de pannes "naturelles". Les pannes d'un noeud peuvent provenir d'un bogue logiciel, de la vétusté du matériel ou du fait de l'environnement extérieur : variations de température, d'humidité, d'alimentation, projection de rayons cosmiques, de champs électromagnétiques, etc ...
La fiabilité d'un noeud est donc difficile à prévoir, elle dépend de beaucoup de paramètres, mais dans le cas de pannes non malicieuses elle peut souvent être évaluée par des études statistiques sur l'ensemble du système. Plus le système distribué est grand plus les pannes sont fréquentes et plus elles sont difficiles à détecter et réparer. Nous avions donné une définition des systèmes distribués empruntée à A.S. Tanenbaum, citons maintenant la définition pessimiste de L. Lamport qui soulève ce problème :

> "Un système réparti est un système qui vous empêche de travailler lorsqu'une machine, dont vous n'avez jamais entendu parlé, tombe en panne." L. Lamport [135]

2.1.2.2 Les clusters

Le cluster (parfois appelé "grappe" en français) est la forme la plus courante des supercalculateurs référencés dans le top500 (82% en juin 2009). Il regroupe des unités de calcul identiques (il est dit *homogène*) connectés via un réseau local haute performance. Sa gestion et son administration est faite de façon complètement centralisée.

Menaces Les clusters ont beaucoup d'avantages par rapport aux menaces éventuelles : Toutes les machines sont localisées au même endroit, accessibles seulement aux personnes habilités, ce qui permet de contrôler, à tout moment, l'environnement extérieur des machines (température, humidité, etc ...). Le réseau est local, il n'est donc pas accessible depuis l'extérieur. Enfin, il est administré pas une unique entité et les machines sont identiques, ceci permet d'avoir un grande homogénéité dans la politique de sécurité (entre autres, les logiciels utilisés sont les mêmes sur toutes les machines, les "patchs" de sécurité sont installés partout en même temps, etc ...).
Ceci n'empêche bien évidemment pas les attaques : si un individu non autorisé parvient à dérober l'accès administrateur, il est le maître à bord. Il pourrait, par exemple, installer des logiciels malicieux (*malware*) sur les machines et enregistrer toutes les données confidentielles qui transitent sur le système, ou encore créer un *déni de service* : rendre le système inutilisable. Cela peut être fait en bloquant toutes les communications, en "assommant" de travail tous les noeuds ou encore en effaçant tout le contenu des disques.

Une authentification forte, une bonne gestion des droits d'accès et des communications sécurisées en sortie et entrée du cluster sont donc nécessaires pour garder un environnement sain, il ne reste alors plus qu'à prendre en compte les différentes pannes "naturelles" et inévitables du système.

Clusters commerciaux Il peut arriver qu'une organisation possédant un cluster loue du temps de calcul à d'autres organisations pour des besoins ponctuels. Ce cas est particulièrement intéressant du point de vue de la sécurité. En effet, l'utilisateur d'un tel système est en droit de se poser un certain nombre de questions : Comment assurer que les machines du cluster ne comportent pas de *malwares* dont le but est d'enregistrer les informations potentiellement confidentielles envoyées sur le cluster ? Comment s'assurer que les calculs sont conformément exécutés et que des fautes ne sont pas intentionnellement injectées par les noeuds du système ? Comment s'assurer qu'un autre utilisateur, travaillant sur le cluster au même moment, n'est pas capable de soutirer des informations confidentielles ? etc ...

Des réponses à ces questions existent et faire une confiance aveugle dans le propriétaire du cluster n'est pas une fatalité. Par exemple, une solution pour assurer la pérennité des machines est de valider un état correct des machines (l'état prenant en compte la mémoire, le disque dur, etc ...) et de faire une signature électronique de cet état. Ceci permet, par vérification de la signature en cours d'exécution, de certifier qu'aucun logiciel malicieux n'a été installé.

2.1.2.3 Les grilles

Les grilles ont d'abord été définies par Ian Foster comme étant des agrégations de clusters [121]. L'idée maîtresse est de connecter plusieurs clusters répartis à travers le monde entier via un réseau global, généralement Internet.
Aujourd'hui, cette définition a beaucoup évolué, la notion de grille est généralisée à une infrastructure virtuelle distribuée rassemblant des ressources potentiellement hétérogènes, délocalisées et autonomes. Les grilles rassemblent ainsi pratiquement l'ensemble des systèmes distribués de grande taille. Nous précisons maintenant quelques unes des infrastructures typiques qui peuvent être rencontrées.

Grille d'information Les grilles d'information ont pour objet le partage de l'information à travers le monde. L'exemple parfait du genre est le WWW[3] qui permet à n'importe quel internaute d'accéder à l'ensemble des pages web par l'intermédiaire de moteurs de recherche. D'autres exemples existent, comme le GIG[4] maintenu par le Département de la Défense des États-Unis, qui a pour but de partager des informations entre personnels militaires et affiliés.

Grille de stockage Les grilles de stockage permettent de stocker de très grandes quantités de données de façon fiable. En répartissant le stockage à travers un grand nombre de sites, la grille assure l'intégrité des données. Les réseaux pair-à-pair d'échange de données (bitTorrent[5], Freenet, eDonkey2000[6], etc ...) entrent à la fois dans cette catégorie et dans les grilles d'information.

Grille de calcul Les grilles de calcul permettent d'utiliser des ressources à travers le monde entier pour des calculs lourds. Deux grands types de grilles de calcul existent :

- Les grilles de calcul par agrégation de clusters de calcul, administrés par différentes institutions (privées ou publiques). Citons par exemple le projet européen EGEE[7] qui rassemble, en Septembre 2009, quelque 260 sites dans 50 pays pour un total de quelque 92.000 coeurs et plusieurs PetaOctet de stockage. Notons aussi la plate-forme de grille expérimentale Grid5000[8] répartie sur 9 sites en France et un à Porto Alegre, au Brésil, et regroupant 17 laboratoires français.
- Les infrastructures de *calcul global* (*Desktop Grids* en anglais), qui ont une approche différente basée sur le *calcul volontaire* ("Volunteer Computing" en anglais). Le propriétaire d'une station de travail peut autoriser un système de calcul volontaire à exploiter une partie de la puissance de calcul de sa machine. Cette idée est similaire à l'utilisation de certain réseaux pair-à-pair, où l'utilisateur permet au système pair-à-pair d'exploiter une partie de son volume de stockage. Ainsi, n'importe quelle machine connectée à Internet peut prendre part, lors de son "temps libre", à l'exécution d'une application globale proposée par un utilisateur. Cette technique permet d'utiliser des milliers, voire des millions, de machines pour

[3] World Wide Web
[4] Global Information Grid
[5] `http://www.bittorrent.com/`
[6] `http://www.edonkey2000-france.com/`
[7] `http://www.eu-egee.org/`
[8] `https://www.grid5000.org`

un coût très faible. Les principaux projets qui ont fait la célébrité du calcul volontaire sont : *distributed.net*[9] qui s'intéresse au cassage de primitives cryptographiques, *SETI@home*[10] qui tente de trouver des formes de vie extraterrestres par le traitement d'images de l'espace, *Folding@home*[11] dédié à la simulation du repliement des protéines et *BOINC*[12], qui regroupe Folding@home et d'autres applications autour d'un même intergiciel.
Remarque 2 *BOINC a une moyenne de 2 Pflops (Peta opérations flottantes par secondes) sur un ensemble de 570000 machines (juillet 2009), en comparaison, le meilleur supercalculateur du top500, l'IBM* Roadrunner*, atteint 1,026 Pflops avec 129600 coeurs (Juin 2009).*

Menaces Pour aborder les différentes grandes menaces qui pèsent sur les grilles nous séparons le cas des grilles de grappes des autres. En plus des différentes menaces auxquelles chaque cluster doit faire face, l'hétérogénéité des ressources et des administrations taille des brèches dans la sécurité de tels systèmes. Cette hétérogénéité multiplie d'autant les failles potentielles ce qui rend difficile et alourdit une politique de sécurité globale. Une métaphore classique de la sécurité des systèmes est de la comparer à une chaîne : la sécurité est aussi forte que le maillon le plus faible. Ainsi, multiplier les maillons, ne peut rendre la chaîne que plus fragile.
Il faut ajouter un autre désavantage majeur par rapport à un cluster, la grille de grappe fait appel à un réseau global standard pour relier les clusters, les communications sur ce réseau doivent donc être fortement sécurisées.

Dans un cadre plus général, le spectre des attaques sur les grilles s'élargit grandement. Si l'on considère, par exemple, un réseau pair-à-pair ou un *desktop grid*, généralement ouvert à n'importe qui, la confidentialité et l'intégrité de l'information, tout autant que la fiabilité et la disponibilité des ressources, sont des notions pour le moins hasardeuses. Les noeuds du système sont administrés par des particuliers et des organisations qui ne peuvent être dignes de confiance du point de vue de l'utilisateur. Sans compter que les machines, connectées à Internet et potentiellement mal administrées, peuvent faire l'objet d'attaques à grande échelle. De telles attaques permettent de contrôler des fractions non négligeables du système. Les techniques de corruption sont infinies, les *malwares* sont véhiculés par le réseau Internet, ils exploitent des défaillances du système ou humaines pour s'approprier une machine. L'exemple du *ver*[13] Witty est édifiant, ce ver utilisait une faille de sécurité dans certains pare-feux pour s'introduire sur une machine et effacer, aléatoirement, des blocs de disque, rendant ainsi rapidement la machine inutilisable. Il a suffit à Witty de 45 minutes, le 19 Mai 2004, pour compromettre la totalité des 12000 machines vulnérables à cette attaque et réparties dans le monde entier[14].

[9] `http://www.distributed.net/`
[10] `http://setiathome.ssl.berkeley.edu/`
[11] `http://folding.stanford.edu/`
[12] Berkeley Open Infrastructure for Network Computing - `http://boinc.berkeley.edu/`
[13] Un Ver est un type de *malware* qui s'auto-reproduit et se déplace en utilisant des mécanismes réseau, an anglais *Worm*
[14] `http://www.caida.org/research/security/witty/`

2.1.2.4 Les nuages

Le *nuage* (*Cloud* en anglais) est un nouveau terme très controversé qui désigne une certaine évolution de la notion de grille. Il rassemble des idées anciennes autour de points fondamentaux : simplicité, élasticité et sécurité. Ces aspects sont assurés par l'utilisation systématique de *machines virtuelles*. L'utilisateur se voit assigné une machine virtuelle individuelle, indépendante des machines physiques sur lesquelles elle va tourner. Tout le travail de l'intergiciel est de répartir efficacement les machines virtuelles sur un ensemble de machines physiques hétérogènes et décentralisées.
Ces concepts n'ont rien de bien nouveau, il semble que ce soit l'apparition des grilles dans le domaine privé ou commercial qui soit à l'origine de la terminologie. En effet, l'application de ce principe est très largement commercial, il s'agit principalement de généraliser le concept de cluster commercial évoqué précédemment et de simplifier son utilisation du point de vue de l'utilisateur. L'utilisateur se contente de décider d'une puissance de calcul, d'une taille de disque et de proposer une application parallèle (codée à l'aide du *MapReduce*[15] de Google par exemple). De nombreuses entreprises proposent aujourd'hui ce service, les principaux acteurs se trouvent sur la figure 2.1.

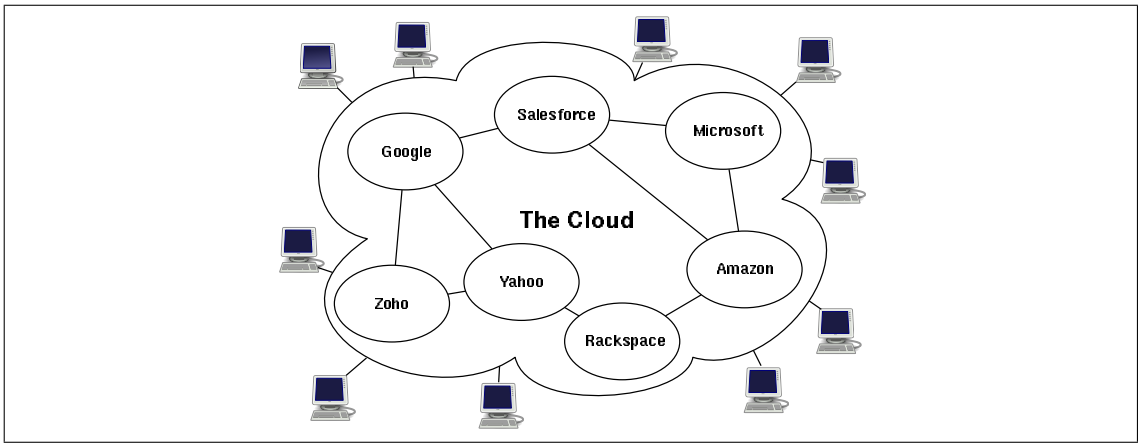

FIG. 2.1 – Le Nuage

Menaces La virtualisation est sensée assurer une bonne sécurité, d'abord parce que les machines virtuelles sont complètement indépendantes les unes des autres : les utilisateurs se retrouvent dans un univers borné, il n'ont pas de moyens d'épier les autres utilisateurs ou de créer des dénis de service. Ensuite parce que les utilisateurs ne contrôlent pas les machines physiques, ils n'ont pas de raisons de savoir de quel type elles sont, ni même leur localisation.

Si la virtualisation permet de protéger des utilisateurs entre eux, il est clair qu'elle rend presque impossible de protéger les utilisateurs contre le propriétaire du nuage. En effet, les machines virtuelles sont des environnements fermés s'exécutant au dessus d'une couche logicielle bas niveau

[15] R. Grossman soutient qu'un développeur n'a pas besoin de plus d'une journée d'entraînement pour se familiariser à MapReduce - IEEE New technologies Conference, 6 Août 2009.

(un système d'exploitation) qui contrôle toutes ses entrées-sorties (accès mémoire, réseau, etc ...).

Par ailleurs, la virtualisation ne suffit pas en elle-même pour accroître la sécurité : un récent article de Ristenpart et al. [182][16], montre qu'une plate-forme en nuage d'Amazon (plate-forme EC2) n'apporte que très peu de sécurité ; les auteurs sont capables de cartographier le nuage, de localiser une machine virtuelle dans le nuage et enfin de créer une instance de machine virtuelle malicieuse qui s'exécutera précisément sur les mêmes machines physiques que la machine virtuelle cible. Elle pourra ainsi récupérer des informations potentiellement confidentielles des calculs opérés par l'application cible en faisant une étude de l'utilisation des processeurs et en particulier de leur cache[17].

2.2 Sûreté de fonctionnement : Définition de la confiance

Nous avons défini, dans la section précédente, les plate-formes privilégiées de notre étude ainsi que quelques types de menaces auxquelles elles doivent faire face. Nous allons introduire maintenant les notions essentielles à la caractérisation et l'évaluation de la *confiance* qu'un utilisateur peut mettre dans une telle plate-forme. Ces notions sont regroupées autour du concept de *sûreté de fonctionnement*, à l'intérieur duquel ont été rigoureusement classés les différents aspects (ou *attributs*) de la confiance, les différentes menaces (ou *entraves*) et les différentes méthodes (ou *moyens*) permettant d'améliorer la confiance.

> "La Sûreté de Fonctionnement d'un système peut être définie comme étant la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. Le service délivré par un système est son comportement tel que perçu ou requis par ses utilisateurs, un utilisateur étant un autre système (humain ou machine) qui interagit avec le système considéré."
>
> J.C. Laprie[18]

Cette approche nous permet de définir les deux types de moyens qui nous intéressent dans cette thèse : la prévention de fautes et la tolérance aux fautes. Pour chacun de ces deux grands domaines, nous nous focaliserons sur une méthode spécifique , respectivement, les chiffres symétriques et les algorithmes auto-tolérants aux fautes.

Cette section reprend brièvement les points fondamentaux de la sécurité de fonctionnement tels qu'ils sont définis dans [9]. La figure 2.2 reprend les trois grandes catégories de la sûreté de fonctionnement d'un système.

2.2.1 Attributs

Six attributs de la sûreté de fonctionnement sont distingués :
- Le fait d'être prêt à l'utilisation conduit à la *disponibilité*.
- La continuité du service conduit à la *fiabilité*.

[16] Le titre évocateur "Hey ! You ! Get off my cloud" reprend les paroles d'une fameuse chanson des Rolling Stones.
[17] http://pro.01net.com/editorial/506055/s-attaquer-au-cloud-possible-mais-pas-si-simple/
[18] IFIP WG 10.4 / J.C. Laprie - LAAS - TOULOUSE

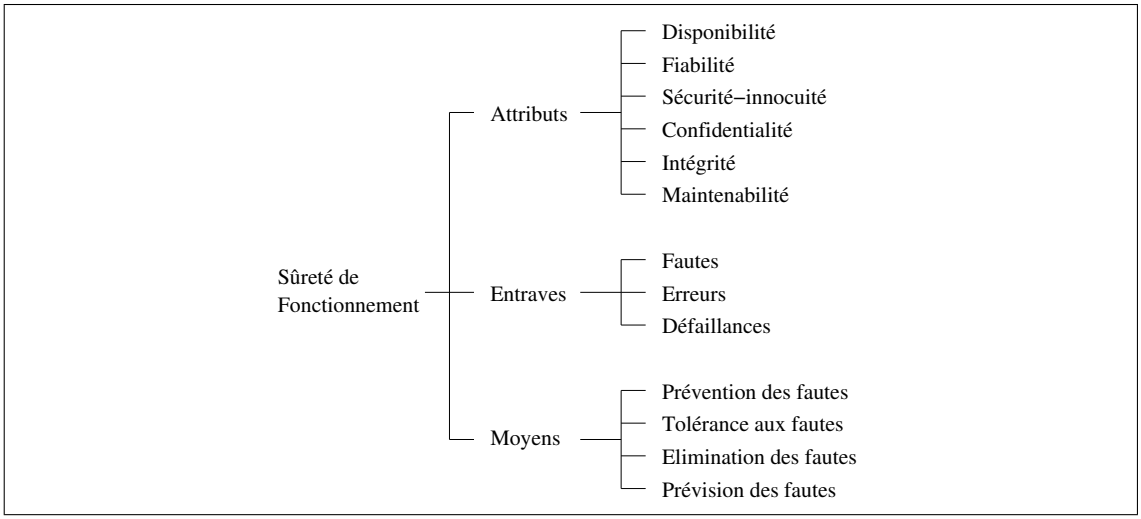

FIG. 2.2 – L'arbre de la sûreté de fonctionnement ([9])

- L'absence de conséquences catastrophiques pour l'environnement conduit à la *sécurité-innocuité*.
- L'absence de divulgations non autorisées de l'information conduit à la *confidentialité*.
- L'absence d'altérations inappropriées de l'information conduit à l'*intégrité*.
- L'aptitude aux réparations et aux évolutions conduit à la *maintenabilité*

Les attributs peuvent être vus comme l'ensemble des points a assurer pour rendre le système fiable. En fonction du système cible, de ses fonctionnalités et de son environnement d'exécution, chacun de ces aspects aura plus ou moins de poids.

2.2.2 Entraves

Trois types d'entrave à la sûreté de fonctionnement sont distingués :

- Les *Défaillances* sont les comportements inacceptables du système du point de vue des utilisateurs.
- Les *Erreurs* représentent les déviations de l'état du système dans un état incorrect, elles sont les sources potentielles de défaillances.
- Les *Fautes* créent les erreurs, elles peuvent être classées en trois grandes catégories non exclusives : les fautes de développement (survenant durant le développement), les fautes physiques (affectant le matériel), et les fautes d'interaction (les fautes qui ont une origine externe au système).

Nous reverrons et détaillerons ces classes de fautes dans le chapitre 8 sur l'étude de la tolérance aux fautes d'un système. Il s'agit ici de constater que chaque défaillance du système est à mettre en rapport avec un (voire plusieurs) attributs de la sûreté de fonctionnement. Ainsi, un *déni de service* correspond à une défaillance affectant la disponibilité du système.

2.2.3 Moyens

Les moyens pour réaliser la sûreté de fonctionnement sont classés en quatre types :
- La *prévention des fautes* consiste à empêcher l'apparition de fautes.
- La *tolérance aux fautes* consiste à fournir un service correct en présence de fautes.
- L'*élimination des fautes* consiste à réduire la présence de fautes.
- La *prévision des fautes* consiste à évaluer le taux présent et futur de fautes.

C'est sur ces aspects que sont identifiées les problématiques de recherche de cette thèse. En effet, les chiffres symétriques et leur intégration dans des systèmes d'information en vue d'assurer confidentialité et l'intégrité (protection contre les falsifications) des informations correspond à une méthode de *prévention des fautes*. Les chiffres symétriques sont étudiés dans la partie II. D'autre part, le sujet des systèmes distribués de grande taille et de leur sécurité rend l'étude des méthodes de *tolérance aux fautes* incontournable : sur de tels systèmes les fautes détruisant la fiabilité et l'intégrité sont souvent inévitables et loin d'être marginales. La tolérance aux fautes est étudiée dans la partie III.

Outils codage et cryptographie 3

Sommaire

3.1 Cryptographie à clé secrète : les chiffrements par blocs 20
3.1.1 Notions de sécurité et théorie de Shannon 20
3.1.1.1 Différentes notions de sécurité 20
3.1.1.2 Entropie 21
3.1.1.3 Produit de chiffres 22
3.1.2 Schéma général d'un chiffrement par bloc : Réseau de Substitution et Permutation (SP-Network) 22
3.1.2.1 Réseau de Substitution et Permutation 23
3.1.2.2 Principaux chiffres par blocs 24
3.1.2.3 Modes opératoires 24
3.1.3 Techniques de cryptanalyse théorique et résistance 25
3.1.3.1 Cryptanalyse Linéaire 26
3.1.3.2 Cryptanalyse Différentielle 28
3.1.3.3 Preuves de résistance 31
3.1.4 Attaques par canaux auxiliaires et contre-mesures 33
3.1.4.1 Introduction 33
3.1.4.2 Analyse de courant 33
3.2 Théorie des codes 38
3.2.1 Préliminaires 38
3.2.1.1 Premières Définitions 38
3.2.1.2 Codes Linéaires 39
3.2.2 Codes de Reed-Muller 40
3.2.2.1 Codes de Reed-Muller binaires d'ordre 1 : $\mathfrak{R}(1, m)$ 41
3.2.2.2 Quelques résultats sur le décodage par liste des codes $\mathfrak{R}(1, m)$ 42
3.2.3 Codes de Reed-Solomon 43
3.2.3.1 Codage et Matrice Génératrice 44
3.2.3.2 Décodage unique des codes de Reed-Solomon 44
3.2.4 Codes par interpolation de polynômes univariés 47
3.2.5 Codes LDPC 47

Ce chapitre introduit les notions mathématiques nécessaires pour la lecture du mémoire.

Nous débuterons ce chapitre par une présentation des chiffrements par blocs en insistant sur certaines méthodes de cryptanalyse et les preuves de résistances ou contre-mesures possibles face à ces attaques. Dans un second temps, nous présenterons les objets et outils de la théorie des codes, qui seront utiles dans la suite du document, notamment les codes de Reed-Muller, de Reed-Solomon et les méthodes dites de décodage par listes. Aucun résultat original n'étant présenté ici, les lecteurs maîtrisant ces notions peuvent directement passer au chapitre suivant.

Note : Dans toute la suite, les notations mathématiques sont celles utilisées dans la littérature. Néanmoins, rappelons que le logarithme en base q est noté $\log_q$. En base 2 nous le noterons parfois simplement $\log$. Nous travaillerons généralement dans le corps fini à q éléments, que nous noterons, de façon interchangeable, $GF(q)$ (le corps de Galois à q éléments) ou de façon plus générique $\mathbb{F}_q$ ($q = p^m$, p premier et m entier).

3.1 Cryptographie à clé secrète : les chiffrements par blocs

Les chiffrements par blocs ou chiffrements itératifs sont des primitives cryptographiques à clé secrète, c'est à dire que le secret est commun pour la fonction de chiffrement et de déchiffrement. Un chiffre symétrique sera noté $S = (\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$ et défini par :

- Un espace de messages clairs ($\mathcal{P}$), un espace de messages chiffrés ($\mathcal{C}$), un espace de clés ($\mathcal{K}$).
- Un algorithme de chiffrement $E : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$.
- Un algorithme de déchiffrement $D : \mathcal{C} \times \mathcal{K} \to \mathcal{P}$.

Dans la suite, nous noterons X, Y et K des variables aléatoires prenant leurs valeurs dans $\mathcal{P}$, $\mathcal{C}$ et $\mathcal{K}$ respectivement.

3.1.1 Notions de sécurité et théorie de Shannon

3.1.1.1 Différentes notions de sécurité

Dans le contexte des chiffrements symétriques, trois notions de sécurité sont employées dans la littérature.

Sécurité parfaite. La notion de *sécurité parfaite* proposée par Claude Elmwood Shannon en 1949 définit l'incapacité théorique de casser le problème au sens de la théorie de l'information [202]. Formellement, la sécurité parfaite peut s'écrire $\Pr[X = x|Y = y] = \Pr[X = x]$.
L'exemple classique de chiffre offrant une telle sécurité inconditionnelle est le chiffre proposé par Gilbert Vernam en 1917 (nommé "One-time Pad" en anglais). Il est très coûteux puisque la clé secrète doit être aussi longue que l'ensemble des messages à échanger et il est dépendant d'une synchronisation continuelle. Il est possible de prouver qu'acquérir une sécurité parfaite demande un coût au moins égal à celui du chiffre de Vernam. Ceci laisse peu de chance opérationnelle à de tels chiffrements. Ainsi d'autre notions de sécurité, moins strictes ont dû être apportées et un compromis entre sécurité et efficacité du chiffre a été recherché.

Sécurité asymptotique. Encore appelée *sécurité prouvée* dans la littérature, cette notion repose sur l'existence de problèmes "bien connus", dont la difficulté est largement acceptée (e.g. Problèmes NP-Complets). Ainsi, par une méthode de réduction, similaire à celles utilisées pour prouver qu'un problème est NP-Complet, il peut être parfois prouvé que casser un chiffre est équivalent à résoudre un problème difficile. Casser un tel chiffre revient donc à résoudre un problème pour lequel le meilleur algorithme connu a une complexité exponentielle en fonction d'un paramètre du système. Il suffit alors de choisir la taille du paramètre convenablement pour empêcher un attaquant possédant une capacité de calcul réaliste de casser le système.

Sécurité empirique. Elle repose sur l'étude de l'ensemble des méthodes connues pour casser le chiffre. Lorsqu'aucune des méthodes connues pour retrouver le secret n'est applicable (i.e., leur coût est supérieur à l'attaque de référence), le chiffre est supposé sûr. C'est la notion la plus faible de sécurité et en contrepartie c'est celle qui permet le développement de chiffres les plus performants. En pratique les chiffres symétriques sont considérés sûrs par leur longévité : plus le chiffre est âgé et plus il a résisté à l'acharnement des cryptanalystes, plus il est "sûr".
Le coût de l'attaque de référence donne la résistance a priori du chiffre. Généralement, cette attaque est la recherche exhaustive de la clé (attaque par force brute). Ainsi un chiffre de taille de clé secrète n bits aura une sécurité d'ordre $O(2^n)$ (chaque clé possible est testée) jusqu'à ce qu'un cryptanalyste trouve une attaque plus efficace.

La large majorité des chiffrements symétriques utilisés à l'heure actuelle possède la plus faible notion de sécurité. Dorénavant, lorsqu'on parlera de chiffrement symétrique sûr il s'agira d'une *sécurité empirique*.

3.1.1.2 Entropie

La notion d'entropie, introduite par Claude Shannon en 1948 dans le domaine de la théorie de l'information [201], est un outil particulièrement bien adapté pour étudier la sécurité des chiffrements. L'entropie peut être interprétée comme une mesure de la quantité d'information contenue dans une source d'information. Ainsi, pour une variable aléatoire X, l'entropie est définie par :

$$H(X) = -\sum_{x} \Pr[X = x] \cdot \log(\Pr[X = x])$$

Quelques propriétés de l'entropie

- $H(X, Y) \geq H(X)$. L'égalité est vérifiée si et seulement si Y ne dépend que de X.
- $H(X, Y) \leq H(X) + H(Y)$. L'égalité est vérifiée si et seulement si X et Y sont indépendants.
- Soit $|X|$ le cardinal de l'ensemble des valeurs prises par X avec une probabilité non nulle, si $|X| = n$ alors $H(X) \leq \log(n)$. L'égalité est vérifiée si l'événement $X = x$ est équiprobable pour tout x.

Sécurité parfaite. La notion de sécurité parfaite peut être formellement décrite à l'aide de l'entropie, elle équivaut à $H(X|Y) = H(X)$ où $H(X|Y)$ est l'*entropie conditionnelle*. Celle-ci mesure, en moyenne, la quantité d'information sur X qui n'est pas révélée par Y :

$$H(X|Y) = -\sum_{y}\sum_{x} \Pr[Y = y] \cdot \Pr[X = x|Y = y] \cdot \log[X = x|Y = y]$$

3.1.1.3 Produit de chiffres

Cette notion fondamentale introduite par Shannon en 1949 est aujourd'hui incontournable dans la construction de chiffrements symétriques sûrs. Considérons deux fonctions de chiffrement $S_1(\mathcal{P}_1, \mathcal{C}_1, \mathcal{K}_1, E_1, D_1)$ et $S_2(\mathcal{C}_1, \mathcal{C}_2, \mathcal{K}_2, E_2, D_2)$. Le produit de ces deux chiffres, noté $S_1 \times S_2$, correspond à $(\mathcal{P}_1, \mathcal{C}_2, \mathcal{K}_1 \times \mathcal{K}_2, E, D)$ où E et D sont définis comme suit :

$$\begin{array}{lrcl} E: & \mathcal{P}_1 \times \mathcal{K}_1 \times \mathcal{K}_2 & \rightarrow & \mathcal{C}_2 \\ & (X, K_1, K_2) & \rightarrow & E_2(E_1(X, K_1), K_2) \\ D: & \mathcal{C}_2 \times \mathcal{K}_1 \times \mathcal{K}_2 & \rightarrow & \mathcal{P}_1 \\ & (Y, K_1, K_2) & \rightarrow & D_1(D_2(Y, K_2), K_1) \end{array}$$

Dans certains cas, le produit $S_1 \times S_2$ est plus sûr que S_1 et S_2 pris séparément. Comme nous allons le voir c'est une technique très utilisée pour la construction de chiffrements par blocs. Elle permet de construire des chiffres sûrs à l'aide de chiffres plus simples et donc plus aptes à être étudiés séparément.

3.1.2 Schéma général d'un chiffrement par bloc : Réseau de Substitution et Permutation (SP-Network)

Shannon [202] définit la fonction de chiffrement comme une cascade de substitutions (permettant la confusion de l'information) et de permutations (permettant la diffusion de l'information). Ainsi la plupart des chiffrements par blocs actuels est constituée d'une suite de compositions de fonctions intégrant ces deux grandes opérations. La structure en "réseau de substitution et permutation" [96] (nous la noterons SPN dans la suite pour Subtitution-Permutation Network en anglais) regroupe les chiffrements par blocs les plus utilisés.

Soit $S(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$ un chiffre symétrique. En pratique nous aurons :
- $\mathcal{P} = \mathcal{C} = \{0;1\}^n$ avec $64 \leq n \leq 256$ la taille de bloc (taille du message et du chiffré en nombre de bits).
- $\mathcal{K} = \{0;1\}^k$ avec $56 \leq k \leq 512$ la taille de clé.

Les chiffres par blocs sont également appelés chiffres itératifs car ils sont construits par produits de chiffres (voir section précédente). C'est par le produit de chiffres plus simples (appelés rondes ou tours S_i) qu'un chiffre est construit $S = S_1 \times S_2 \times \cdots \times S_R$ de sorte que la clé secrète doit être choisie dans l'ensemble $\mathcal{K}_1 \times \mathcal{K}_2 \times \cdots \times \mathcal{K}_R$. En pratique, de façon à ne pas faire augmenter trop librement la taille de la clé secrète, les clés $K_1, K_2, \cdots, K_R$ (appelées *clés de rondes*) sont dérivées à partir d'une clé maître ; véritable clé secrète du chiffre. Cette opération est

l'*expansion de clé*, elle constitue une partie essentielle du chiffrement et est faite par l'intermédiaire d'un algorithme *génération de clés* qui peut-être vu comme un générateur pseudo-aléatoire prenant la clé maître pour graine.

La figure 3.1 représente un algorithme de chiffrement par bloc dans sa forme la plus générale.

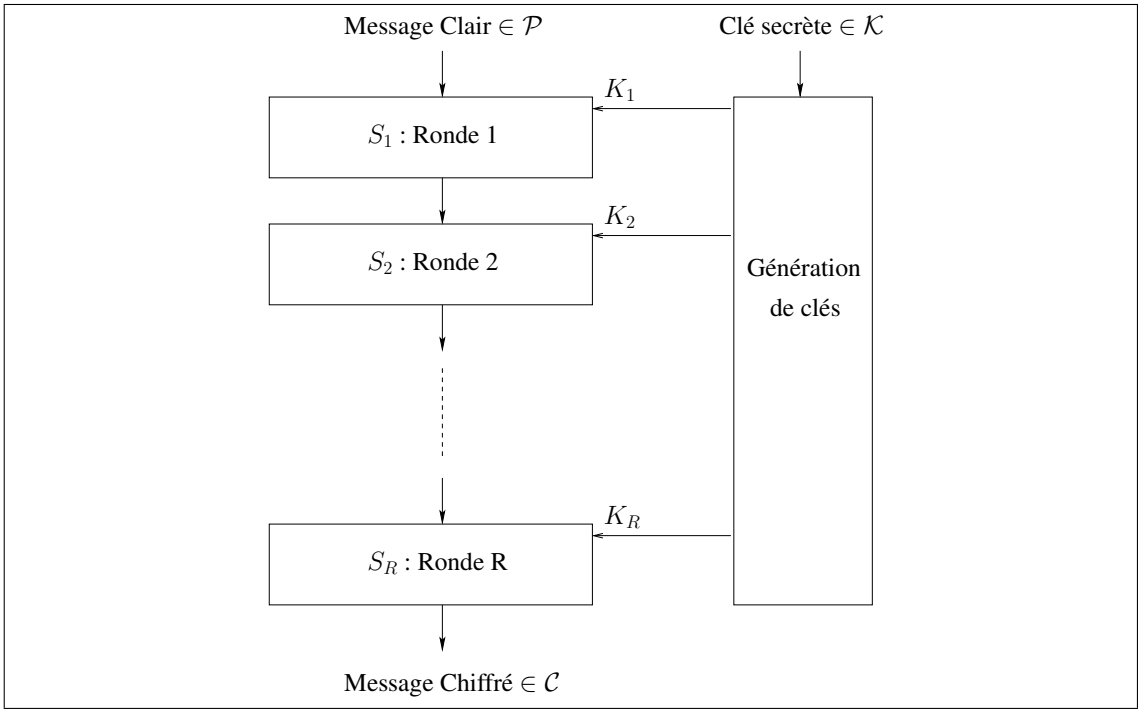

FIG. 3.1 – Schéma général d'un chiffrement itératif

3.1.2.1 Réseau de Substitution et Permutation

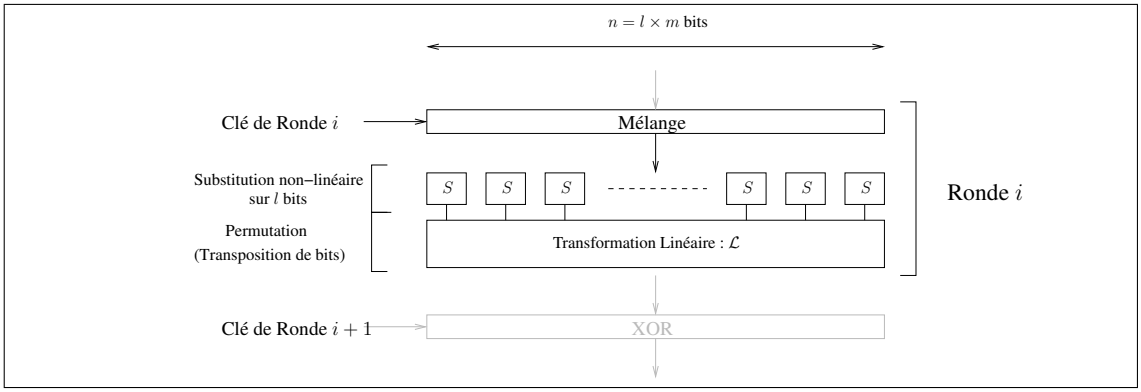

FIG. 3.2 – Schéma d'une ronde de type réseau de substitution-permutation

Considérons maintenant une ronde seule. D'après la remarque faite par Shannon, cette ronde doit faire intervenir à la fois de la confusion et de la diffusion. La figure 3.2 illustre un exemple de réseau de substitution et permutation, ainsi une ronde est constituée de trois éléments (dans cet ordre pour une structure SPN) :

- Une fonction qui mélange la clé de ronde avec le message clair.
- Une **zone de confusion**. Elle est constituée de *boîtes de substitution*, nommées *boîtes-S* dans la suite (*S-Box* en anglais). Ces fonctions travaillent sur une fraction de l'entrée, leur taille d'entrée varie selon le chiffre mais ne dépasse généralement pas 8 bits. Ces boîtes sont limitées en taille pour plusieurs raisons. Tout d'abord, augmenter la taille rend la construction plus difficile car ces boîtes suivent des contraintes de non-linéarité très fortes. D'autre part, la non-linéarité des boîtes rend leur implémentation matérielle ardue et coûteuse. La complexité augmenterait avec la taille des boîtes.
- Une **zone de diffusion**. Elle contient une application linéaire ou *fonction de diffusion* permettant de répartir la redondance du message en diffusant des parties du message à travers la totalité du message.

Remarque 3 *Il peut arriver que les zones de diffusion et de confusion soient échangées, nous parlerons alors de structure* PSN *(e.g., CS-Cipher est un chiffre de type PSN, pour plus de détails voir chapitre 7).*

3.1.2.2 Principaux chiffres par blocs

L'histoire des chiffrements par blocs modernes commence avec le *Data Encryption Standard*, noté **DES**, qui est accepté en 1977 par le *NBS* (National Bureau of Standart) sous la norme FIPS PUB 46. Bien que présentant des faiblesses, il est encore aujourd'hui largement utilisé à travers le monde. En 2000, après un concours international proposé par le NIST ("National Institute Of Standart and Technology", successeur du NBS), un nouveau standard fut accepté pour remplacer le chiffre DES : **AES**, pour *Advanced Encryption Standard* sous la norme FIPS 197.
Mis à part ces deux standards, qui jouissent d'une notoriété particulière, de multiples chiffres par blocs ont été développés et utilisés depuis ces 20 dernières années. Nous noterons, par exemple : **IDEA**, proposé par Xuejia Lai et James Massey au début des années 90 [133], **GOST**, équivalent du DES pour l'ex-Union soviétique [168], **Blowfish**, développé par Bruce Schneier en 1993 [195], **SAFER**, proposé par James Massey en 1993 [147] (et ses successeurs SAFER+ et SAFER++), **CS-Cipher**, proposé par Jacques Stern et Serge Vaudenay en 1998 [205] il fera l'objet d'étude du chapitre 7, **Camellia**, un standard de chiffrement Japonais, proposé par Aoki et al. en 2000 [7], **Serpent**, conçu par Ross Anderson, Eli Biham et Lars Knudsen pour le concours AES dont il fait partie des 6 finalistes [5], **FOX**, publié par Pascal Junod et Serge Vaudenay en 2004 [108], basé sur le schéma de IDEA et aujourd'hui breveté par la société Mediacrypt (sous le nom de IDEA NXT), ... et tant d'autres.

3.1.2.3 Modes opératoires

Les modes opératoires sont là pour pallier au fait que les chiffres par blocs ont une taille de message fixée. Ils définissent la marche à suivre pour chiffrer un message de longueur arbitraire. Les premiers modes opératoires sont spécifiés dans la norme FIPS PUB 81 [164] du NIST. Ils ont

été créés pour le chiffre DES et peuvent être utilisés pour assurer la confidentialité du message ou à des fins d'authentification. A l'heure actuelle, de nombreux modes opératoires ont été développés. Certains intègrent les deux fonctionnalités en même temps (confidentialité et authentification). Voir, par exemple, les modes acceptés par le NIST `http://csrc.nist.gov/groups/ST/toolkit/BCM/current_modes.html` et les modes proposés au NIST, en cours de validation : `http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html`.
Nous reviendrons sur un mode opératoire particulier dans le chapitre 4.

3.1.3 Techniques de cryptanalyse théorique et résistance

Nous appelons cryptanalyse théorique, la science des attaques sur les chiffres vus comme objets mathématiques. Elle est à opposer à la cryptanalyse pratique ou physique qui correspond aux attaques prenant en compte l'environnement extérieur dans lequel l'objet mathématique évolue (abordée dans la section 3.1.4).
Loin de présenter de façon exhaustive l'ensemble des techniques de cryptanalyse théorique connues à ce jour, cette section a pour but de présenter deux outils fondamentaux de cryptanalyse statistique : les cryptanalyses dites linéaire et différentielle. Depuis leur découverte, ces deux outils ont été la base de la plus grande majorité des attaques sur les chiffrements par bloc. Aujourd'hui encore, la plupart des attaques proposées reposent, au moins en partie, sur les principes qui en découlent. De fait, l'étude de résistance à ces types d'attaques est une étape indispensable dans la construction de chiffres symétriques. Nous verrons comment parvenir à prouver la résistance, ou du moins à donner de puissants arguments de résistance, face à ces attaques.
Notons tout de même que les attaques statistiques ne sont pas les seules types d'attaques sur les chiffrements par blocs. Par exemple on peut citer les attaques par clés corrélées [25] (Related-key Attacks en anglais) et une de ses variantes, les attaques "slide" [34, 35], ou encore les attaques algébriques [60, 18].

Attaques et Complexité Soit une attaque $\mathcal{A}$ sur un algorithme de chiffrement E. $\mathcal{A}$ repose sur un certain nombre d'hypothèses qui peuvent être interprétées comme une modélisation des capacités de l'attaquant. Dégageons les principales hypothèses utilisées :

- *Chiffre cible connu* : Certaines attaques sont possibles sans cette hypothèse, pourtant il est largement accepté que supposer l'algorithme de chiffrement inconnu est illusoire en pratique (cf. Principes de Kerckhoffs 1883).
- *Chiffrés seuls* : L'attaquant a accès à la sortie de l'algorithme de chiffrement uniquement (notons que le modèle jumeau "Clairs seuls" ne présente aucun intérêt).
- *Clairs connus* ou *Chiffrés connus* : l'attaquant a accès à un ensemble de couples clair-chiffrés qu'il ne peut choisir (ils sont donc supposés choisis aléatoirement).
- *Clairs choisis* : l'attaquant a accès à un oracle de chiffrement (contenant la clé secrète), auquel il peut lancer une requête de chiffrement avec un message clair et récupérer le chiffré associé.
- *Chiffrés choisis* : Identique à Clairs choisis avec un oracle de déchiffrement.

Les hypothèses de $\mathcal{A}$ permettent une première classification des attaques. Un deuxième niveau de classification se fait par l'analyse de complexité de $\mathcal{A}$.

Plusieurs paramètres peuvent être pris en compte pour évaluer la complexité de $\mathcal{A}$. Notamment :

- Complexité en données : nombre de données nécessaires pour monter l'attaque. Cette complexité équivaut à une complexité algorithmique de l'attaque car le nombre de données est du même ordre de grandeur que le nombre de fois qu'est appelé l'algorithme de chiffrement (ou de déchiffrement).
- Complexité en mémoire : taille mémoire nécessaire pour réussir l'attaque.

Les deux prochaines sections présentent les cryptanalyses linéaires et différentielles. Pour une présentation plus détaillée de ces deux méthodes se reporter à l'excellent tutoriel de Howard M. Heys [95].

3.1.3.1 Cryptanalyse Linéaire

La cryptanalyse linéaire a été proposée par Mitsuru Matsui en 1993 [148]. Cette méthode a conduit à une attaque du chiffre DES plus efficace (en ordre de grandeur) que la force brute [149]. L'idée de base de cette méthode de cryptanalyse est la recherche et l'utilisation d'approximations linéaires de l'algorithme de chiffrement par bloc.

Définitions et notations utiles

Définition 1 (Approximation linéaire) *Soit f une fonction de $\{0;1\}^n$ dans $\{0;1\}^m$. La donnée de deux vecteurs de bits (appelés masques) $(\alpha, \beta) \in \{0;1\}^n \times \{0;1\}^m$ et d'un réel $\epsilon \in [-\frac{1}{2}, \frac{1}{2}]$ définit une approximation linéaire de f de biais ϵ si et seulement si*

$$\Pr_X[\langle \alpha, X \rangle \oplus \langle \beta, f(X) \rangle = 0] = 1/2 + \epsilon \tag{3.1}$$

où $\langle ., . \rangle$ est le produit scalaire de vecteurs d'éléments dans $GF(2)$ et $\oplus$ le ou exclusif bit à bit (ou encore l'addition dans $GF(2)^n$).

Définition 2 (Probabilité Linéaire) *La probabilité linaire d'une fonction f par rapport à un couple de masques (α, β), notée $LP^f(\alpha, \beta)$ est définie par :*

$$LP^f(\alpha, \beta) = (2 \Pr_X[\langle \alpha, X \rangle \oplus \langle \beta, f(X) \rangle = 0] - 1)^2 \tag{3.2}$$

Si l'approximation linéaire de f relativement au masque (α, β) a un biais ϵ, alors

$$LP^f(\alpha, \beta) = (2\epsilon)^2 \tag{3.3}$$

La probabilité linéaire maximale est définie par :

$$LP^f_{max} = \max_{a \neq 0, b} LP^f(a, b) \tag{3.4}$$

Définition 3 (Probabilité Linéaire Espérée) *La probabilité linaire espérée d'une fonction f_k, dépendante d'un paramètre aléatoire k, par rapport à un couple de masques (α, β), notée $ELP^{f_k}(\alpha, \beta)$ est définie par :*

$$ELP^{f_k}(\alpha, \beta) = \mathbf{E}[LP^{f_k}(\alpha, \beta)] \tag{3.5}$$

où l'espérance est prise sur la distribution de k.

Définition 4 (Caractéristique Linéaire) *Soit un chiffre $S(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$, dont l'algorithme de chiffrement est constitué de R rondes $E_K = f_{k_R} \circ \cdots \circ f_{k_1}$. On appelle une r-rondes caractéristique linéaire Ω, une suite de $R+1$ masques $(\omega_0, \cdots, \omega_R)$.*

Nous pouvons alors définir la probabilité caractéristique linéaire.

Définition 5 (Probabilité Caractéristique Linéaire) *La probabilité caractéristique linéaire d'une fonction de chiffrement $E_K = f_{k_R} \circ \cdots \circ f_{k_1}$ sur R rondes est définie par rapport à la caractéristique linéaire $\Omega = (\omega_0, \cdots, \omega_R)$ comme suit :*

$$LCP^{E_K}(\Omega) = \prod_{i=1}^{R} LP^{f_{k_i}}(\omega_{i-1}, \omega_i) \tag{3.6}$$

Définition 6 (Probabilité Caractéristique Linéaire Espérée) *La probabilité caractéristique linéaire espérée d'une fonction de chiffrement $E_K = f_{k_R} \circ \cdots \circ f_{k_1}$ sur R rondes est définie par rapport à la caractéristique linéaire $\Omega = (\omega_0, \cdots, \omega_R)$ comme suit :*

$$ELCP^{E_K}(\Omega) = \prod_{i=1}^{R} ELP^{f_{k_i}}(\omega_{i-1}, \omega_i) \tag{3.7}$$

où l'espérance est prise sur la distribution des clés de rondes.

La cryptanalyse linéaire, de façon très générale, utilise une approximation linéaire de l'algorithme de chiffrement dont le biais ϵ n'est pas nul. Un biais nul caractérise une équation linéaire totalement décorrélée de l'algorithme de chiffrement et donc ne portant aucune information intéressante (si ce n'est le fait qu'elle n'en porte pas). Au contraire, une approximation linéaire de biais $1/2$ ou $-1/2$ définit exactement le comportement de l'algorithme de chiffrement.
Une attaque par cryptanalyse linéaire consiste, classiquement, à trouver une approximation linéaire de l'algorithme de chiffrement de biais non négligeable, c'est à dire un couple de masques de probabilité linéaire grande. Matsui a notamment montré qu'une telle approximation permet de monter une attaque à clairs connus de complexité en données $O(1/\epsilon^2)$ [148].

Cryptanalyse Multi-linéaire Notons au passage l'une des variantes de l'attaque de Matsui : les attaques multi-linéaires. Elles reposent sur l'observation que l'utilisation simultanée de plusieurs approximations linéaires permet d'accélérer l'attaque originale. Ainsi, A. Biryukov, C. De Cannière et M. Quisquater ont prouvé [33] que n approximations linéaires indépendantes de biais respectifs $\epsilon_1, \epsilon_2, \cdots, \epsilon_n$ permettent de monter une attaque de complexité en données $\frac{1}{\sum_{i=1}^{n} \epsilon_i^2}$.

Description d'une attaque L'attaque comporte deux étapes disjointes :

Trouver une approximation linéaire. Cette étape n'est à faire qu'une seule fois, le cryptanalyste pourra réutiliser l'approximation trouvée chaque fois qu'il cherchera à déchiffrer des messages chiffrés à l'aide du chiffre cible. La façon traditionnelle de procéder est de construire une caractéristique linéaire de l'algorithme de chiffrement. Il faut donc trouver des approximations linéaires suffisamment proches de la fonction de ronde qui puissent être chaînées

les unes aux autres. Matsui en propose de très bonnes pour le chiffre DES. Mais aujourd'hui encore, trouver les meilleurs approximations linéaires reste un sujet ouvert. Le nombre d'équations linéaires possibles sur $GF(2)$ à n variables est de 2^n, ce qui rend la recherche exhaustive impossible dans le cas des chiffres modernes. Heureusement la structure des rondes de l'algorithme de chiffrement nous aide car, à y regarder de plus près, les seules parties non-linéaires (et donc qui nécessitent d'être approximées) sont les boîtes-S. On en déduit directement que de deux caractéristiques linéaires, celle qui fera intervenir le plus petit nombre de boîtes-S sera sûrement la meilleure (si on oublie bien sûr que certaines boîtes peuvent être mieux approximées que d'autre). Cette remarque est fondamentale pour la compréhension des techniques de preuves de résistances que nous verrons dans la section suivante.
Lorsque plusieurs boîtes-S doivent être approximées au sein d'une ronde, le lemme *Piling-Up* de Matsui [148] permet d'évaluer le biais de l'approximation linéaire résultante sur l'ensemble de la ronde.

Lemme 1 (Piling-Up) *Soit n variables aléatoires binaires indépendantes $X_1, X_2, \cdots, X_n$ de biais respectifs $\epsilon_1, \epsilon_2, \cdots, \epsilon_R$ (c'est à dire $\forall i, \Pr[X_i = 0] = p_i = 1/2 + \epsilon_i$) alors,*

$$\Pr[X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0] = 1/2 + 2^{R-1} \prod_{i=1}^{R} \epsilon_i \tag{3.8}$$

Ainsi, en considérant que les boîtes-S sont toutes indépendantes à l'intérieur d'une ronde, il est possible d'évaluer le biais de l'approximation linéaire globale.

Remarque 4 *D'autres techniques permettent de trouver des approximations linéaires. Nous présenterons notamment une méthode récente liée au décodage par listes de codes correcteurs dans le chapitre 6.*

Recouvrer la clé secrète. Une fois qu'une approximation linéaire est trouvée, il existe plusieurs algorithmes pour retrouver la clé secrète. Nous présentons ici l'attaque la plus simple : récupération de la dernière clé de ronde à l'aide d'une approximation linéaire de $E(R-1)$ (E dont on aurait supprimé la dernière ronde). Une fois la dernière clé de ronde retrouvée il faut réitérer l'attaque sur $E(R-2)$ et ainsi de suite pour retrouver l'ensemble de la clé.
Soit ϵ le biais de notre approximation linéaire de $E(R-1)$, β le masque de sortie de l'approximation et K_R la dernière clé de ronde.
Pour chaque valeur de K_R possible, évaluer sur un ensemble de $1/\epsilon^2$ de couples clairs-chiffrés l'équation linéaire. Si elle est vérifiée sur une grande fraction de l'ensemble des couples clairs-chiffrés, alors le choix de clé était le bon, sinon recommencer avec le choix suivant.

3.1.3.2 Cryptanalyse Différentielle

La cryptanalyse différentielle a été proposée par Biham et Shamir en 1991 [29], deux ans avant la cryptanalyse linéaire. Comme cette dernière, elle a été inventée pour être appliquée au chiffre DES, mais ne conduit pas directement à une attaque effective du chiffre. Il est intéressant de noter que si DES ne possédait que 15 rondes au lieu de 16, l'attaque aurait été meilleure que

la force brute ! Beaucoup en ont conclu que la cryptanalyse différentielle était connue par les créateurs de DES lors de sa conception presque 20 ans auparavant. Un des concepteur du standard DES, Don Coppersmith, confirmera en 1994 [58] qu'une technique similaire de cryptanalyse était effectivement connue dès 1974.
Notons qu'une deuxième version de l'attaque par cryptanalyse différentielle, proposée par Biham et Shamir l'année d'après, est la première attaque plus efficace que la force brute sur l'ensemble du chiffre DES [30].
Alors que la cryptanalyse linéaire fait appel à des approximations linéaires de l'algorithme de chiffrement, la cryptanalyse différentielle fait, elle, appel à des approximations de différences encore appelées "différentielles".

Définitions et notations utiles

Définition 7 (Probabilité Différentielle) *La probabilité différentielle d'une fonction f par rapport à un couple de différences (α, β), notée $DP^f(\alpha, \beta)$ est définie par :*

$$DP^f(\alpha, \beta) = \Pr[f(X) \oplus f(X \oplus \alpha) = \beta] \tag{3.9}$$

La probabilité différentielle maximale :

$$DP^f_{max} = \max_{\alpha \neq 0, \beta} DP^f(\alpha, \beta) \tag{3.10}$$

Définition 8 (Probabilité Différentielle Espérée) *La probabilité différentielle espérée d'une fonction f_k, dépendante d'un paramètre aléatoire k, par rapport à un couple de différences (α, β), notée $EDP^{f_k}(\alpha, \beta)$ est définie par :*

$$EDP^{f_k}(\alpha, \beta) = \mathbf{E}[DP^{f_k}(\alpha, \beta)] \tag{3.11}$$

où l'espérance est prise sur la distribution de k.

Définition 9 (Caractéristique Différentielle) *Soit un chiffre $S(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$, dont l'algorithme de chiffrement est constitué de R rondes $E_K = f_{k_R} \circ \cdots \circ f_{k_1}$. On appelle une r-rondes caractéristique différentielle Ω, une suite de $R+1$ différences $(\omega_0, \cdots, \omega_R)$.*

Nous pouvons alors définir la probabilité caractéristique différentielle.

Définition 10 (Probabilité Caractéristique Différentielle) *La probabilité caractéristique différentielle d'une fonction de chiffrement $E_K = f_{k_R} \circ \cdots \circ f_{k_1}$ sur R rondes est définie par rapport à la caractéristique différentielle $\Omega = (\omega_0, \cdots, \omega_R)$ comme suit :*

$$DCP^{E_K}(\Omega) = \prod_{i=1}^{R} DP^{f_{k_i}}(\omega_{i-1}, \omega_i) \tag{3.12}$$

Définition 11 (Probabilité Caractéristique Différentielle Espérée) *La probabilité caractéristique différentielle espérée d'une fonction de chiffrement $E_K = f_{k_R} \circ \cdots \circ f_{k_1}$ sur R rondes est définie par rapport à la caractéristique différentielle $\Omega = (\omega_0, \cdots, \omega_R)$ comme suit :*

$$EDCP^{E_K}(\Omega) = \prod_{i=1}^{R} EDP^{f_{k_i}}(\omega_{i-1}, \omega_i) \tag{3.13}$$

où l'espérance est prise sur la distribution des clés de rondes.

La cryptanalyse différentielle utilise des différentielles (couples de différences) possédant une grande probabilité d'occurrence. Il a été montré qu'à partir d'une différentielle de probabilité $p >> 1/(2^n)$ (n étant la taille de bloc du chiffre), une attaque différentielle peut être montée avec une complexité en donnée de $O(1/p)$.

Cryptanalyse par impossible différentielle En 1999, E. Biham, A. Biryukov et Ali Shamir [27] proposèrent une attaque par impossible différentielle, qui, au lieu de s'intéresser aux différentielles de hautes probabilité repose sur les différentielles de probabilité nulle. A ce jour, les attaques par impossible différentielle font partie des meilleurs attaques à clairs choisis sur le chiffre AES [142]. De plus, contrairement à la cryptanalyse différentielle classique, il n'y a, à ce jour, de preuve de résistance permettant de montrer qu'un chiffre est robuste face à cette attaque. Nous discuterons de ce problème au chapitre 5.

Description d'une attaque De la même façon que pour l'attaque par cryptanalyse linéaire, l'attaque par cryptanalyse différentielle comporte deux grandes étapes.

Trouver une différentielle. La méthode est très similaire à celle que l'on a vu pour la cryptanalyse linéaire. Il s'agit de trouver une caractéristique différentielle et d'évaluer sa probabilité d'occurrence. De même que pour la recherche d'approximations linéaires, il n'y a pas de méthode miracle pour trouver de bonnes différentielles, et ici encore, c'est sur les boîtes-S que repose toute l'attention puisqu'une application linéaire ne possède que des différentielles exactes ou nulles : Si $E(., K)$ est linéaire, $E(X, K) \oplus E(X \oplus \alpha, K) = E(\alpha, K)$. Pour une caractéristique donnée, chaque boîte-S dont la différence en entrée n'est pas nulle implique une nouvelle différentielle non exacte sur cette boîte dont la probabilité sera multipliée aux autres en diminuant d'autant la probabilité de la différentielle globale. Les boîtes-S dont la différence en entrée pour une caractéristique donnée est non nulle sont appelées boîte-S actives. Ainsi, de la même façon que pour la cryptanalyse linéaire, la meilleur différentielle est celle qui fera intervenir le moins possible de boîtes-S actives.

Recouvrer la clé secrète. L'attaque présentée dans le cadre de la cryptanalyse linéaire est strictement transposable dans le contexte de la cryptanalyse différentielle.

Méthodes de cryptanalyse dérivées De nombreuses méthodes de cryptanalyse statistique reposent sur le principe de la cryptanalyse linéaire ou différentielle. De façon non exhaustive : Impossibles différentielles [27], Différentielles tronquées et de plus haut degrés [126], cryptanalyse multi-linéaire [107], cryptanalyse différentielle-linéaire [137], attaques Boomerang ou Rectangle [222, 119, 28], Attaque χ^2 [218], Attaque Non-Surjective [69].

3.1.3.3 Preuves de résistance

Le sujet des preuves de résistance, brièvement abordé dans la section précédente, est primordial dans la conception de chiffres. Nous verrons que malheureusement, même en considérant des attaques particulières (linéaires et différentielles), il est très difficile, voire impossible, de prouver la résistance des chiffres symétriques ou encore de proposer un schéma qui le serait.
Mais avant toute chose, posons le modèle qui servira de support pour l'étude de sécurité des chiffres par blocs face aux attaques linéaires et différentielles.

Un modèle : Chiffre de Markov

Définition 12 (Chaîne de Markov) *Soit $(X_n)_{n\geq 0}$ une suite de variables aléatoires discrètes. Cette suite forme une chaîne de Markov, si et seulement si :*

$$\forall i < n, \forall (x_0, \cdots, x_{i+1}), \Pr(X_{i+1} = x_{i+1} | X_i = x_i, \ldots, X_0 = x_0) = \Pr(X_{i+1} = x_{i+1} | X_i = x_i) \tag{3.14}$$

De plus, si $\forall (x_i, x_{i+1}), \Pr(X_{i+1} = x_{i+1} | X_i = x_i)$ est indépendant de i, la chaîne de Markov est dite homogène.

Autrement dit, une chaîne de Markov est un outil pour modéliser un processus aléatoire dont l'état présent contient toute l'information de conditionnement de l'état futur, les états passés n'apportent pas d'information complémentaire.

En 1991, Lai, Massey et Murphy [134] proposèrent d'utiliser cet outil pour l'étude de la récente cryptanalyse différentielle. En effet, ils remarquèrent que pour les algorithmes de chiffrement par blocs, en supposant les clés de rondes indépendantes les unes des autres, la suite de variables aléatoires décrivant les différences de données entre chaque ronde est une chaîne de Markov. De plus, lorsque les clés de rondes suivent la même distribution (par exemple uniforme) la chaîne de Markov est homogène.

Définition 13 (Chiffre de Markov [134]) *Un chiffre par bloc avec une fonction de ronde $Y = f(X, K)$ est un* chiffre de Markov *si il y a une opération de groupe $\oplus$, d'élément neutre e, permettant de définir des différences et telles que pour tout choix de α ($\alpha \neq e$) et β ($\beta \neq e$),*

$$\Pr_K(\Delta Y = \beta | \Delta X = \alpha, X = \gamma) \tag{3.15}$$

est indépendant de γ quand la clé K est uniformément aléatoire.

Le résultat majeur des auteurs est que si l'on suppose les clés de rondes indépendantes et uniformément distribuées, un chiffre de Markov tend vers un chiffre immune contre les attaques différentielles lorsque le nombre de rondes tend vers l'infini. Ce résultat repose sur le fait que l'algorithme de chiffrement E est une bijection en fixant la clé K. Ainsi il est facile de voir que si le choix de la différence en entrée est uniformément distribué sur toutes les différences distinctes possibles de message clairs, alors la différence en sortie l'est aussi. Ceci est donc un état stationnaire de la chaîne de Markov (point fixe de la matrice de transition). Pour une chaîne de Markov *régulière* ce point fixe est unique et lorsque le nombre d'étapes (de rondes ici) tend vers l'infini,

la matrice de transition tend vers un état stationnaire redistribuant parfaitement les différences d'entrées sur l'ensemble des différences de sorties. Prouver que la chaîne de Markov est régulière se fait généralement en montrant que les rondes de la fonction de chiffrement génèrent le groupe alterné ou le groupe symétrique [98].
Ce modèle justifie la définition de DCP introduite à la section précédente, car pour un chiffre de Markov, la DCP coïncide avec la probabilité d'occurrence d'une caractéristique $\Omega = (\omega_0, \cdots, \omega_R)$:

$$\Pr[E_K(\Omega)] = \prod_{i=1}^{R} \mathrm{DP}^{f_{k_i}}(\omega_{i-1}, \omega_i)$$

$E_K(\Omega)$ représentant l'événement où la caractéristique est suivie tout au long de l'algorithme de chiffrement.
Malheureusement cette technique ne nous donne pas une solution pratique pour évaluer la différentielle de probabilité maximum après un nombre de rondes fixées. La matrice de transitions de la chaîne de Markov d'un chiffre de Markov est de taille $(2^n)^2$ où n est la taille de bloc. Il est donc complètement illusoire de l'utiliser telle qu'elle est. Nous reviendrons sur ce problème dans le chapitre 5.

Résistance sous conditions De nombreux travaux ont été entrepris pour essayer de prouver la résistance aux attaques différentielles et linéaires. Comme nous l'avons vu dans la section précédente, cette résistance sera sujette au nombre minimal de boîtes-S intervenant dans une caractéristique linéaire ou différentielle.
Les preuve de résistances se concentrent donc sur l'évaluation de la fonction de diffusion et l'étude des boîtes-S.

Diffusion. Il faut que la diffusion soit la meilleur possible pour qu'un chemin (une caractéristique) fasse intervenir le plus de boîtes-S possible. Ainsi la probabilité d'une caractéristique pourra être maximisée en fonction de la probabilité différentielle ou linéaire maximale des boîtes-S (DP_{max}^{SBox}, LP_{max}^{SBox}). Citons par exemple les travaux de Heys et Tavares [96] avec la notion d'"ordre de diffusion", de Daemen [61], Vaudenay [198, 219] avec l'idée de "multi-permutations" ou encore de Daemen, Knudsen et Rijmen qui utilisent un code MDS pour la construction du chiffre Square [62].

Boîtes-S Il faut trouver des boîtes qui sont particulièrement adaptées, c'est-à-dire que leur probabilité différentielle et linéaire maximale doit être la plus petite possible. Notons les travaux de Nyberg [162].

Un certain nombre d'articles, dérivés de ces travaux, a donné les preuves de résistance aux attaques différentielles et linéaires en bornant les probabilités de caractéristiques différentielles ou linéaires, entre autre [8, 111, 163, 112, 117, 165, 166]. Ces "preuves de sécurité" sont plus des "arguments de sécurité" que des preuves au sens mathématique du terme. Par exemple, une borne maximale de la probabilité d'occurrence d'une caractéristique différentielle n'implique pas qu'il existe une borne maximale intéressante de la probabilité d'occurrence d'une différentielle et donc qu'une attaque différentielle n'est pas possible. Par ailleurs, ces preuves sont souvent faites sous des conditions rarement vérifiées, comme l'indépendance des clés de rondes.

3.1.4 Attaques par canaux auxiliaires et contre-mesures

Nous avons vu dans la première partie de ce chapitre les attaques que nous appelons théoriques (ou conventionnelles) car elles se limitent à l'étude du chiffre comme objet purement mathématique. Pourtant, il a été remarqué que l'utilisation du même chiffre dans la vie réelle pouvait introduire des faiblesses. Pour être plus précis, si l'on considère que l'attaquant peut, d'une façon ou d'une autre, observer ou manipuler l'environnement dans lequel évolue le chiffre, il peut avoir accès à de nouvelles informations que les concepteurs de chiffre n'avaient pas pris en compte.
Le pouvoir de l'attaquant sur l'environnement peut être désastreux. Imaginons qu'un attaquant veuille retrouver la clé secrète utilisée par une carte à puce pour chiffrer des données sensibles. Si l'attaquant a accès à la carte et est capable de lire le contenu des mémoires qui stockent la clé secrète, alors le chiffre le plus fort qui soit ne servira à rien.
Si empêcher un attaquant de lire la mémoire sensible d'une carte à puce ou plus généralement d'un circuit intégré est faisable (et de toute façon n'est pas du ressort du cryptologue), certaines attaques, dites par "canaux auxiliaires" doivent être prises en compte, à la fois par le concepteur du circuit et par le concepteur du chiffre.

3.1.4.1 Introduction

Les attaques par canaux auxiliaires ont connu un tournant d'envergure en 1996. Bien que certaines attaques s'y rapprochant avaient déjà été tentées avec succès dans le passé, la dynamique de recherche autour de cette thématique n'a pris son envol qu'avec les papiers de Kocher [128] où est présenté une attaque par *analyse de temps d'exécution* et de Dan Boneh, Richard DeMillo et Richard Lipton [42] qui introduit les *attaques par perturbations*. En 1998 Paul Kocher, Joshua Jaffe et Benjamin Jun présentaient les attaques par *analyse de courant* [127, 129]. En 1999 Biham et al. [31] portaient les attaques par perturbation dans le domaine des chiffres symétriques. Enfin, en 2000, Jean-Jacques Quisquater et David Samyde [171] introduisaient les attaques par analyse de rayonnement électromagnétique (EMA).
Depuis lors, la recherche autour de ce domaine n'a fait que se multiplier. Les attaques qui découlent de ces principes d'observation ou de perturbation se sont trouvées être d'une efficacité redoutable. Les concepteurs de chiffres embarqués sont aujourd'hui dans la nécessité d'implémenter des contre-mesures à ces attaques.
Nous nous intéresserons plus particulièrement aux attaques par *analyse de courant*. Brièvement présentées dans la section suivante, elles seront l'objet d'étude du chapitre 6.

3.1.4.2 Analyse de courant

Les attaques par analyse de courant reposent sur l'idée que les variations de courants, durant l'exécution d'un algorithme, sont dépendantes des instructions utilisées et des données manipulées par l'algorithme.
Considérons une fonction implémentée en technologie CMOS (Complementary Metal Oxide Semiconductor), comme c'est le cas pour une très grande majorité des circuits intégrés aujourd'hui. Le circuit intégré est constitué de portes logiques, elles-mêmes constituées de transistors. Dans un tel circuit, la transition d'un bit de 0 à 1 ou de 1 à 0 consomme sensiblement plus d'énergie qu'une transition de 0 à 0 ou de 1 à 1. Cette asymétrie permet à un attaquant, observant la puissance

consommée durant deux exécutions d'un même circuit, de différencier l'exécution qui a conduit à un plus grand nombre de changements de bits, à un moment fixé. Les premières attaques par analyses de courant (PA pour Power Analysis en anglais), décrites dans [127], sont : l'analyse de courant simple (SPA pour Simple Power Analysis en anglais) et l'analyse de courant différentielle (DPA pour Differential Power Analysis en anglais). Aujourd'hui il faut y ajouter les attaques basées sur les collisions [200, 199, 139], basées sur la cryptanalyse différentielle [49, 92], basées sur les attaques algébriques [177], ou encore basées sur les approximations linéaires (MLPA pour Multi-linear Power Analysis), Ce dernier type d'attaque sera abordé plus en détail au chapitre 6.

SPA Une attaque en SPA consiste à déduire la valeur du secret recherché par simple observation de la consommation de courant. Cette attaque est particulièrement efficace lorsque l'algorithme exécuté contient des branchements conditionnels conduisant à deux chemins de consommation différents. Cette forme de SPA n'est pas très efficace contre les chiffrements symétriques, des contre-mesures peu coûteuses existent [55].
Une autre forme bien plus dangereuse et généralisée par Suresh Chari, Josyula Rao et Pankaj Rohatgi [52] sous le nom d'attaque par *Templates* existe. Elle repose sur la création d'une bibliothèque de mesures de consommation pendant une phase précédent l'attaque, chaque consommation correspondant à un choix d'une entrée et d'une clé. En comparant les consommations lors de l'exécution du chiffre avec une clé inconnue et les consommations de la bibliothèque, l'attaquant est capable de récupérer le secret. Cette attaque est sûrement la technique de cryptanalyse par canaux auxiliaires la plus efficace et qui pose le plus de problèmes à contrer. Elle implique toutefois que l'attaquant a accès à une implémentation d'entraînement strictement similaire à l'implémentation cible.

DPA Les attaques par DPA se regroupent autour d'une idée simple : faire une hypothèse sur la clé (ou plutôt sur une sous-partie de la clé). Cette hypothèse permet d'évaluer la valeur d'un ou plusieurs bits manipulés au niveau d'un registre ou d'un bus à partir d'un message d'entrée connu. Cette évaluation correspond au choix d'une *fonction de sélection*. Enfin, il s'agit de comparer l'évolution de la consommation par rapport aux valeurs devinées du registre. Si cette comparaison met en évidence un lien entre les deux échantillons de valeurs (devinées et mesurées), alors l'hypothèse de clé est la bonne ; sinon c'est que la consommation et la valeur du registre sont décorrélées, l'hypothèse de clé est donc fausse.

Remarque 5 *Les valeurs étudiées sont presque toujours des valeurs de registres ou de données échangées sur un bus. En effet, ces données sont stables et synchronisées avec les fronts de l'horloge, alors que les valeurs prises par les fils à l'intérieur de la logique combinatoire du circuit ne sont pas fiables (phénomène de "glitch") et non synchronisées.*

L'idée générale de la DPA peut être minutieusement raffinée pour un modèle de consommation et pour le choix de la méthode statistique permettant de distinguer une bonne d'une mauvaise clé.

Modèles de consommation Bien qu'ils restent des modèles grossiers, les modèles de consommation les plus utilisés sont le poids de Hamming (HW) et la distance de Hamming (HD). L'idée du modèle HD suit directement la remarque introductive de cette section sur la consommation en

technologie CMOS. Ainsi, il paraît naturel de poser comme modèle de consommation au niveau d'un registre, le poids de Hamming de la différence entre la donnée précédemment stockée dans le registre et la valeur qui la remplace au cycle suivant. La consommation est proportionnelle au poids de Hamming de cette différence. En d'autres termes, plus il y a de bits qui changent de valeur, plus la consommation est grande. La figure 3.3 présente l'évolution de la consommation en fonction de la différence de Hamming pour une constellation de 80000 points produits pour le *concours DPA* proposé par le groupe de recherche VLSI du département COMELEC de l'école d'ingénieur française Télécom ParisTech (voir `http://www.dpacontest.org/`). Le modèle HW est plus simple car il fait directement appel au poids de Hamming de la valeur stockée (et non la différence). Ce modèle est généralement moins proche de la réalité. Ces modèles peuvent être raffinés pour une implémentation particulière, en considérant que l'attaquant a des informations précises de l'implémentation matérielle.

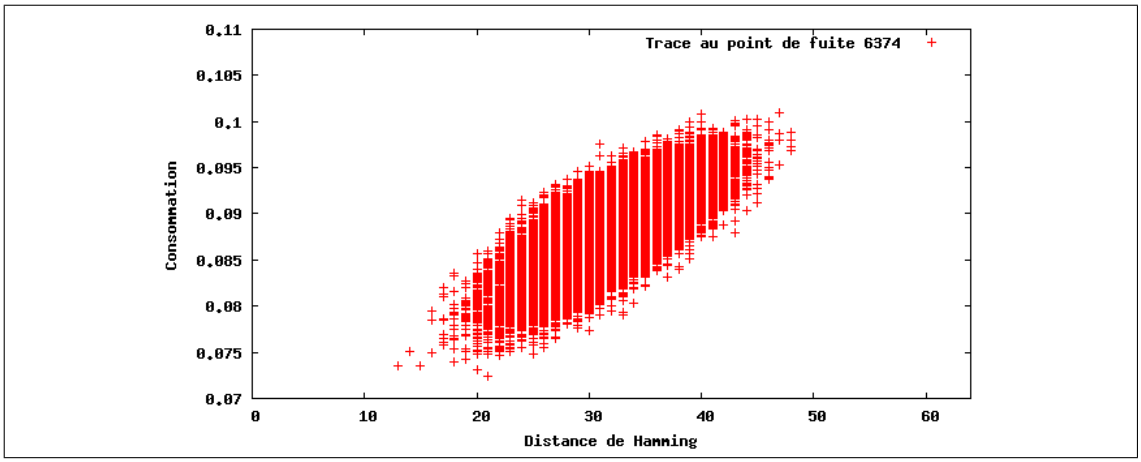

FIG. 3.3 – Consommation en fonction de la distance de Hamming

Méthodes statistiques Une fois le choix de la fonction de sélection fait, on se trouve devant un problème classique de statistique : Quelle dépendance y a-t-il entre les deux variables aléatoires, considérant leurs valeurs prises sur un échantillon ? Les deux variables aléatoires considérées $I_K(X)$ et $I_{\widehat{K}}(X)$ sont respectivement la consommation mesurée à un moment précis et la prédiction de cette consommation au même moment via la fonction de sélection. Ainsi, considérant la fonction de sélection comme un choix pertinent, si $\widehat{K} = K$, alors les deux variables aléatoires sont corrélées. Inversement, si $\widehat{K} \neq K$, alors les deux variables ne le sont pas.
Plusieurs méthodes statistiques sont utilisées dans la littérature. Les plus importantes sont :

Distance des moyennes. Proposée dans l'article original de Kocher et al. [127], l'idée est très intuitive. La fonction de sélection permet de séparer les traces en différents ensembles suivant la valeur de consommation prédite. Sur chaque ensemble, la consommation moyenne doit être significativement différente. Ainsi, la distance des moyennes des différents ensembles met au jour une potentielle corrélation. Si il n'y a pas de dépendance entre la fonction de sélection et les consommations mesurées, les ensembles de traces devraient avoir une moyenne très proche ; la distance des moyenne sera donc proche de 0. La distance des

moyennes pour une fonction de sélection sur un bit (conduisant à deux ensembles de traces $S^0_{\hat{K}}$ et $S^1_{\hat{K}}$) s'écrit :

$$\Delta_{\hat{K}} = \frac{\sum_{I_K(M_i)\in S^1_{\hat{K}}} I_K(M_i)}{|S^1_{\hat{K}}|} - \frac{\sum_{I_K(M_i)\in S^0_{\hat{K}}} I_K(M_i)}{|S^0_{\hat{K}}|} \tag{3.16}$$

où les M_i sont les différents messages clairs connus pendant l'attaque. Elle peut s'écrire aussi :

$$\Delta_{\hat{K}} = \mathbf{E}[I_K(X)|I_K(X) \in S^1_{\hat{K}}] - \mathbf{E}[I_K(X)|I_K(X) \in S^0_{\hat{K}}] \tag{3.17}$$

où l'espérance est prise sur la variable aléatoire X, représentant les messages clairs uniformément distribués lors d'une attaque à clairs connus. Ici K est fixé, la valeur $\Delta_{\hat{K}}$ la plus grande pour un choix de $\hat{K}$ particulier implique $K = \hat{K}$.
Lorsque la fonction de sélection est multi-bits, plusieurs solutions par distance des moyennes ont été proposées [153, 138]. Par exemple [138] :

$$\Delta_{\hat{K}} = \sum_{i=0}^{n} a_i \mathbf{E}[I_K(X)|I_K(X) \in S^i_{\hat{K}}] \tag{3.18}$$

Corrélation. Une autre méthode, bien connue dans le domaine des statistiques, permettant de mesurer la dépendance entre deux variables aléatoires, est d'utiliser des coefficients de corrélation de Pearson. Ce type de DPA est appelé CPA (pour Correlation Power Analysis en anglais), il a été proposé par Brier et al. [45] en 2004. Bien que limitée à la mesure de dépendance linéaire entre deux variables, c'est une méthode très efficace pour notre problème lorsque le modèle par distance de Hamming est utilisé [220]. Le coefficient est défini comme suit :

$$\rho(X,Y) = \frac{\sum_{x,y}(x-\bar{X})(y-\bar{Y})}{\sqrt{\sum_x (x-\bar{X})^2 \cdot \sum_y (y-\bar{Y})^2}} \tag{3.19}$$

où $\bar{X}$ signifie la moyenne.

Information Mutuelle. L'utilisation de l'information mutuelle pour une attaque DPA porte le nom MIA (pour Mutual Information Analysis en anglais) et a été proposée par Gierlichs et al.[86] en 2008. L'information mutuelle est une méthode plus générale de mesure de dépendance entre deux variables aléatoires. Elle est définie par :

$$I(X,Y) = \sum_{x}^{y} \Pr[X=x, Y=y] \log_2 \left(\frac{\Pr[X=x, Y=y]}{\Pr[X=x] \cdot \Pr[Y=y]} \right) \tag{3.20}$$

et est liée à l'entropie H que nous avons vue à la section 3.1.1.2 :

$$\begin{aligned} I(X,Y) &= H(X) - H(X|Y) \\ &= H(X) + H(Y) - H(X,Y) \\ &= H(X,Y) - H(X|Y) - H(Y|X) \end{aligned}$$

L'information mutuelle permet de mesurer les dépendances plus généralement qu'une simple corrélation, en particulier, elle n'est pas limitée à une dépendance linéaire.
En pratique, le modèle de consommation en distance de Hamming a une dépendance linéaire avec la consommation, ainsi la méthode par corrélation est plus efficace, car plus précise. Dans le cas où le signal est très bruité (par ajout de bruit par exemple), l'information mutuelle sera sûrement plus efficace car plus stable [220].

HO-DPA Déjà dans ses travaux de 1998, Kocher mentionne les attaques par analyse de courant différentielle de plus haut degré. Dans ce type de DPA, l'attaquant ne se limite pas à un seul point d'attaque dans l'algorithme de chiffrement. Il peut ainsi combiner des fuites à divers moments durant le chiffrement. Cette technique est particulièrement redoutable face aux contre-mesures qui ont été développées pour la DPA simple. Aujourd'hui encore, construire une implémentation efficace et robuste contre une attaque HO-DPA de degré d est un problème non résolu.

Contre-mesures à la DPA La puissance des attaques de type DPA a créé l'urgence dans le développement de contre-mesures. L'idée maîtresse est de rendre la fuite d'information inutilisable par l'attaquant et donc de dé-corréler la consommation du circuit et les opérations qu'il effectue. Six grands types de contre-mesures existent :

L'ajout de bruit : Cela consiste à ajouter des opérations inutiles en parallèle de l'exécution de la primitive cryptographique. L'idée est de rendre le bruit assez fort pour masquer les variations de consommation dues à l'exécution du chiffre de façon à rendre les attaques trop coûteuses.

Lissage : L'ajout de filtres entre la source de courant et le circuit permet de lisser la consommation et ainsi d'amoindrir les variations de courant. Tout comme l'ajout de bruit, cette technique tente de rendre l'attaque trop coûteuse : arriver à une consommation constante est impensable avec les technologies actuelles, le but est de s'en rapprocher le plus possible.

Désynchronisation : La DPA repose sur le fait que l'attaquant est capable de répertorier des mesures de consommation correspondant à l'exécution de la même instruction avec des données différentes. La désynchronisation cherche donc à modifier, de façon aléatoire, le moment où une instruction est exécutée. Ceci peut être fait par ajout de boucles d'attentes ou en modifiant régulièrement la fréquence de l'horloge [215].

Duale rail : Il s'agit de dédoubler chaque fil à l'intérieur du circuit ; chaque pair de fil contenant une valeur et son complémentaire [47, 54].

Design au Niveau Logique : Comme nous l'avons vu, la DPA utilise le fait que les portes logiques consomment différemment, suivant les valeurs qu'elle manipulent. L'idée est de modifier les portes logiques pour qu'elles aient une consommation constante, soit en reconstruisant de nouvelles portes logiques [14], soit en utilisant de bonnes combinaisons de celles existantes [214].

Masquage : Les techniques de masquages, sont sûrement les contre-mesures qui ont fait couler le plus d'encre. L'idée est de masquer aléatoirement les données en entrée de l'algorithme de chiffrement, de façon à ne manipuler que des valeurs indépendantes des données réelles lors du chiffrement. Par exemple, pour un masquage additif, la valeur d'entrée est le message clair M et la valeur qui sera chiffrée sera $M' = M \oplus R_1$ où R_1 est un vecteur de bits choisi

aléatoirement de même taille que M et $\oplus$ est l'opération : ou exclusif bit à bit. A la fin du chiffrement, il faut alors démasquer la valeur chiffrée à l'aide d'un masque R_2 de façon à ce que $E(M', K) \oplus R_2 = E(M, K)$ [89, 51, 4, 144, 183]. Bien que coûteuses, des techniques de masquage existent et sont utilisées pour faire face à une attaque DPA simple ou d'ordre faible (HO-DPA d'ordre ≤ 2) [59]. Il n'est pour l'instant pas praticable de faire face à des HO-DPA d'ordres plus hauts.
A notre connaissance la seule solution connue à ce jour a été présentée par Y. Ishai et al. [103] en 2003. Elle rend impossible une attaque HO-DPA d'ordre d arbitraire mais pour un prix exorbitant : grossièrement en multipliant la taille de l'implémentation par un facteur d'ordre $O(d)$.

3.2 Théorie des codes

Nous avons vu, dans la section précédente les outils cryptographiques qui nous servirons dans les chapitres suivants. Il nous faut maintenant introduire quelques outils, définitions et notations de la théorie des codes, nécessaires pour la suite.
Pour une présentation très large de la théorie des codes voir [101] ou [36].

3.2.1 Préliminaires

3.2.1.1 Premières Définitions

Soit un alphabet $\mathcal{Z}$ contenant q éléments (appelés *symboles*), soit un entier naturel n, un *code correcteur d'erreur* (appelé *code* dans la suite) C est un sous ensemble de $\mathcal{Z}^n$.
L'entier n est la *taille de bloc* du code, et les éléments de C sont les *mots de code*. Si $q = 2$ alors le code est appelé *code binaire*.
La *dimension* du code $k = \log_q |C|$ (où $|C|$ est le cardinal de C) permet de définir le *rendement* du code : $R = \frac{k}{n}$, ainsi que le *nombre de symboles de redondance* du code $r = n - k$. Le rendement correspond au taux d'information source contenu dans un mot de code, tandis que le nombre de symboles de redondance représente le nombre de symboles nécessaires pour coder l'information (on passe d'une information décrite sur k symboles, à une information plus redondante nécessitant n symboles).
La *distance minimale* d_{min} du code C est la distance de Hamming minimale entre deux mots de code :

$$d_{min} = \min_{(c_1, c_2 \neq c_1) \in C^2} \Delta(c_1, c_2)$$

$\Delta(.,.)$ dénote la distance de Hamming : pour deux vecteurs d'éléments de $\mathcal{Z}$ de même taille, c'est le nombre de coordonnées où ces vecteurs diffèrent.

Codage/Décodage Considérons un message $x \in \mathcal{Z}^k$ que l'on veut envoyer sur un canal bruité. La première étape est de coder ce message à l'aide d'un *algorithme de codage* consistant en une fonction bijective entre $\mathcal{Z}^k$ et C. Le mot de code obtenu y est envoyé sur un canal bruité, il peut être reçu avec des erreurs, i.e. une partie des symboles ont changé. Dans ce cas, si le mot reçu $\hat{y}$ n'est plus un mot de code ($\hat{y} \in \mathcal{Z}^n / C$), l'erreur est détectée. La détection d'erreur se fait donc par

test d'appartenance du mot reçu à C. Dans le cas d'une erreur le mot peut parfois être corrigé et remplacé par le mot de code d'origine à l'aide d'un *algorithme de décodage*.
Un code C peut détecter jusqu'à $d_{min} - 1$ erreurs et en corriger jusqu'à $\lfloor \frac{d_{min}-1}{2} \rfloor$. Un *effacement* est une erreur dont on connaît la localisation, cela correspond par exemple à une perte de paquet. De manière général, en présente d' code C peut corriger t erreurs et e effacements si et seulement si $d_{min} > 2t + e$.

3.2.1.2 Codes Linéaires

Un code linéaire est défini par un sous espace vectoriel de $\mathbb{F}_q^n$. Rappelons que $\mathbb{F}_q$ est le corps fini ou corps de Galois à q éléments, q peut donc être un nombre premier ou la puissance d'un nombre premier.
La dimension d'un code linéaire C est alors la dimension de l'espace vectoriel qu'il définit. On notera $[n, k, d]_q$ un code linéaire de taille de bloc n, de dimension k et de distance minimale d défini sur $\mathbb{F}_q$.

Matrice Génératrice et Matrice de Parité Nous avons vu qu'un code linéaire $C = [n, k, d]_q$ est un sous espace vectoriel de dimension k de $\mathbb{F}_q^n$. L'algorithme de codage pour un code linéaire est donc un isomorphisme G d'espace vectoriel de $\mathbb{F}_q^k$ vers C. La représentation matricielle, de dimension $k \times n$, de cet isomorphisme est la *matrice génératrice*, sa seule donnée caractérise complètement C.
De façon duale, il existe un morphisme H d'espace vectoriel entre $\mathbb{F}_q^n$ et $\mathbb{F}_q^{n-k}$ tel que $C = Ker(H)$ ($Ker(H)$ étant le noyau de H). Sa représentation matricielle, de dimension $(n-k) \times n$, est appelée *matrice de parité*, sa seule donnée caractérise complètement C.
Ainsi, à la réception d'un mot $\hat{y}$ à la sortie d'un canal bruité, le calcul du *syndrome* $s = H(\hat{y})$ permet directement de détecter d'éventuelles erreurs ($s \neq 0$). De plus, considérant une erreur e, telle que $\hat{y} = e + y$ (y étant le mot de code d'origine et $+$ l'opération d'addition sur $\mathbb{F}_q^n$), on a $s = H(e+y) = H(e) + H(y) = H(e)$ puisque H est une application linéaire. Décoder $\hat{y}$ revient donc à trouver le vecteur e de poids de Hamming minimal tel que $H(e) = s$. Si une telle solution e, de poids inférieur à $\lfloor \frac{d_{min}-1}{2} \rfloor$, existe alors elle est unique, ainsi la résolution de cette équation, dans une boule de rayon $\lfloor \frac{d_{min}-1}{2} \rfloor$ et centrée sur le vecteur nul, est appelée *décodage unique*. De façon strictement équivalente, le décodage a pour but de chercher un mot de code dans une boule de même dimension mais cette fois-ci centrée sur $\hat{y}$. Si l'algorithme de décodage unique ne trouve aucune solution de poids inférieur à $\lfloor \frac{d_{min}-1}{2} \rfloor$ c'est qu'il y a eu trop d'erreurs pour qu'elles puissent être corrigées.

Définition 14 (Codes Systématiques) *un code linéaire $[n, k, d]_q$ est dit* systématique *lorsque, pour tout élément $x \in \mathbb{F}_q^k$, le mot de code $y \in \mathbb{F}_q^n$ coïncide avec x sur ses k premières coordonnées. La matrice génératrice G d'un tel code est de la forme :*

$$G = \left(\begin{array}{c|c} Id_{k\times k} & T_{(n-k)\times k} \end{array} \right)$$

Pour tout code linéaire C, de matrice génératrice G, il existe une matrice de passage P telle que le code C_{sys} engendré par la matrice $P \times G$ est systématique. C_{sys} et C sont dits *équivalents*, ils possèdent les même paramètres (dimension, taille de mots, distance minimale, etc ...).

Théorème 1 (Borne du singleton) *Soit un code linéaire $[n, k, d]_q$, sa distance minimale d est telle que :*

$$d \leq n - k + 1 \tag{3.21}$$

Preuve Soit un code linéaire $C \subset \mathbb{F}_q^n$ de dimension k. Soit E l'ensemble contenant tout les éléments de $\mathbb{F}_q^n$ ayant leurs $k-1$ premières composantes nulles

$$E = \{u \in \mathbb{F}_q^n | u_1 = \cdots = u_{k-1} = 0\}$$

E est trivialement un sous-espace vectoriel de $\mathbb{F}_q^n$ de dimension $n - k + 1$, nous avons donc $dim(E) + dim(C) = n + 1 > dim(\mathbb{F}_q^n)$ et donc $E \cap C \neq \emptyset$. Soit $x \in E \cap C$, la distance de Hamming entre le mot de code x et le mot de code nul (toujours présent dans un code linéaire), vaut $n - k + 1$. La distance minimale d de C vérifie bien $d \leq n - k + 1$. □

Définition 15 (Codes MDS) *Les codes linéaires atteignant la borne du singleton sont dit MDS (pour "Maximum Distance Separable" en anglais).*

Décodage par liste Le *décodage par liste* est une alternative au décodage unique. Comme son nom l'indique ce type de décodage renvoie une liste de mots de codes éligibles pour le décodage du mot reçu, au lieu de renvoyer une unique solution. L'intérêt premier du décodage par liste est l'agrandissement de l'espace de recherche (et donc la capacité de décodage). Or lorsqu'on augmente les dimensions de la boule centrée sur $\hat{y}$, il est possible que plusieurs mots de codes s'y trouvent. L'algorithme de décodage renvoie donc la liste des mots de codes possibles.
Un des sujets initiaux d'étude en décodage par liste est donc l'évaluation du nombre de mots de codes contenus dans une boule de rayon fixé pour un code considéré. Dans un second temps, il faut bien évidement trouver un algorithme de décodage efficace. Nous allons voir, dans la section suivante, un exemple de décodage par liste pour les codes de Reed-Muller binaires d'ordre 1.

3.2.2 Codes de Reed-Muller

Les *codes de Reed-Muller* sont une des plus anciennes familles de codes linéaires (1954 [155, 174]). Bien qu'ayant une distance minimale relativement petite (et donc un capacité de correction faible) ils ont l'avantage de posséder un algorithme de décodage généralement simple et efficace.
Un code de Reed-Muller d'ordre r, de taille de bloc $n \leq q^m$ défini sur $\mathbb{F}_q$, avec $m \geq r$, est l'ensemble des polynômes à m variables de degré au plus r à valeurs dans $\mathbb{F}_q$. Si $q = 2$ c'est un code de Reed-Muller binaire et sera noté $\mathfrak{R}(r, m)$.
La dimension d'un code $\mathfrak{R}(r, m)$ vaut $\binom{r+m}{r}$ et la distance minimale vaut, au moins, $n(1 - r/q)$.
Il y a plusieurs manières de définir un code $\mathfrak{R}(r, m)$. Une manière particulièrement intéressante pour nous est de voir un mot de code de $\mathfrak{R}(r, m)$, qui est donc le codage d'un polynôme sur un élément de $\mathbb{F}_2^{2^m}$, comme l'évaluation de ce polynôme en 2^m points. L'algorithme de décodage consiste à retrouver le polynôme qui coïncide le plus possible avec les évaluations bruitées reçues.

3.2.2.1 Codes de Reed-Muller binaires d'ordre 1 : $\mathfrak{R}(1, m)$

Encore appelés *codes de Hadamard* binaires, les codes $\mathfrak{R}(1, m)$ nous seront utiles dans le chapitre 6.
Le code $\mathfrak{R}(1, m)$ est donc constitué de toutes les fonctions affines (polynômes de degré 1) à m variables définies sur $\mathbb{F}_2$. Notons f une telle fonction affine, f peut s'écrire

$$f(x_1, \cdots, x_m) = f_0 + f_1 x_1 + \cdots + f_m x_m$$

avec $(f_0, f_1, \cdots, f_m) \in \mathbb{F}_2^m$.
Ainsi $\mathfrak{R}(1, m)$ est de dimension $k = m + 1$, l'algorithme de codage converti un mot $(f_0, f_1, \cdots, f_m)$ de longueur k en un mot de code y de taille 2^m tel que $y = (f(x))_{x \in \mathbb{F}_2^m}$ est l'évaluation de f en tout point de $\mathbb{F}_2^m$. Le rendement du code $\mathfrak{R}(1, m)$ est donc $R = \frac{2^m}{m+1}$ et sa distance minimale est $d_{min} = 2^{m-1}$.
Ces codes particuliers peuvent être décodés très rapidement à l'aide de la *transformée d'Hadamard* rapide (notée FHT) [10, 188].

Définition 16 (Transformée d'Hadamard) *Soit Φ une fonction de $\mathbb{F}_2^m$ dans $\mathbb{C}$, la transformée d'Hadamard de Φ est définie par :*

$$\hat{\Phi}(u) = \sum_{x \in \mathbb{F}_2^m} \Phi(x)(-1)^{\langle u, x \rangle}$$

où $\langle ., . \rangle$ dénote le produit scalaire de vecteurs sur $\mathbb{F}_2^m$.
Soit f un mot de code de $\mathfrak{R}(1, m)$, la fonction affine peut s'écrire $f(x) = f_0 + \langle (f_1, \cdots, f_m), x \rangle$. Il est alors facile de voir, en notant $F(x) = (-1)^{f(x)}$, que

$$\hat{F}(u) = \begin{cases} (-1)^{f_0} 2^m & \text{si } u = (f_1, \cdots, f_m) \\ 0 & \text{sinon} \end{cases}$$

Enfin, en notant $g(x) = f(x) + e(x)$ le mot reçu après l'ajout d'une erreur $e(x)$, on peut montrer que : Si $\#\{x | e(x) = 1\} \leq \lfloor \frac{d_{min}-1}{2} \rfloor$ alors

$$|\hat{G}(f_1, \cdots, f_m)| = \max_{u \in \mathbb{F}_2^m} |\hat{G}(u)|$$

où $G(x) = (-1)^{g(x)}$ et $\hat{G}$ est la transformée d'Hadamard de G.
La stratégie de décodage unique qui dérive de ces remarques est très simple et très efficace :

1. Récupérer un mot g et calculer $G = (-1)^g$.
2. Calculer la transformée de Hadamard de G en tout point de $\mathbb{F}_2^m$. Pour une FHT, la complexité est de $m2^m$ en temps et 2^m en mémoire.
3. Récupérer la valeur $u \in \mathbb{F}_2^m$ telle que $|\hat{G}(u)|$ soit maximale et en déduire la valeur de $(f_1, \cdots, f_m)$.
4. La valeur de f_0 est déduite du signe de $\hat{G}(u)$.

3.2.2.2 Quelques résultats sur le décodage par liste des codes $\mathfrak{R}(1, m)$

Nous avons vu un algorithme efficace de décodage unique des codes de Reed-Muller binaires d'ordre 1. Nous présentons maintenant un résultat important sur le *décodage par liste* de ces codes. L'algorithme que nous présentons est dû à G. Kabatiansky et C. Tavernier [109, 110] et se base sur les travaux de O. Goldreich et al. [87, 88]. Il a été adapté à la recherche d'approximations linéaires pour un algorithme de chiffrement symétrique (voir section 3.1.3.1 et chapitre 6) par R. Fourquet et al. [77].

L'algorithme de décodage par liste pour le code $\mathfrak{R}(1, m)$ permet de décoder dans une boule de rayon $T = 2^m(1/2 - \epsilon)$ autour du mot $g(x)$ reçu. Ainsi, pour $0 < \epsilon < 1/2$, le décodage se fait en dehors des limites du décodage unique, l'erreur $e(x)$ est alors telle que :

$$\frac{d_{min} - 1}{2} \leq \#\{x|e(x) = 1\} \leq d_{min} - 2^m\epsilon$$

Pour un biais ϵ, la complexité en pire cas de cet algorithme est $O(m^2\epsilon^6 \log \frac{1}{\epsilon}(\log m + \log \frac{1}{\epsilon} + \log \frac{1}{P_{err}}))$ où P_{err} est la probabilité de se tromper dans le décodage [77].
D'autre part, la *borne de Johnson* établit que le nombre de mots de codes contenus dans une boule de rayon $2^m(1/2 - \epsilon)$ est toujours limité, en ordre de grandeur, par $O(1/\epsilon^2)$.

Description de l'algorithme de décodage L'algorithme de décodage est itératif, sur $m + 1$ étapes. Soit g un mot reçu sous forme d'un élément de $\mathbb{F}_2^{2^m}$. Le but du décodage est de trouver la liste des fonctions affines à m variables sur $\mathbb{F}_2$ coïncidant avec g sur au moins $2^m(1/2 + \epsilon)$ éléments de $\mathbb{F}_2^m$, notons $L_\epsilon(g)$ cette liste. Nous pouvons écrire (avec les notations de la section précédente) :

$$L_\epsilon(g) = \{(c_0, c) \in \mathbb{F}_2 \times \mathbb{F}_2^m | (-1)^{c_0}\hat{G}(c) = (-1)^{c_0} \sum_{x \in \mathbb{F}_2^m} (-1)^{g(x) \oplus \langle c, x \rangle} \geq 2\epsilon 2^m\}$$

ou encore,

$$L_\epsilon(g) = \{c \in \mathfrak{R}(1, m) | \Delta(g, c) \leq 2^m(1/2 - \epsilon)\}$$

où $\Delta(g, c)$ est la distance de Hamming des deux vecteurs $(g(x))_{x \in \mathbb{F}_2^m}$ et $(c(x))_{x \in \mathbb{F}_2^m}$.
A la $i$ème étape, l'algorithme construit la liste $\mathcal{L}_i^g(\epsilon)$ qui correspond à la liste $L_\epsilon(g)$ dont chaque élément $c = (c_1, \cdots, c_m)$ est tronqué à partir de la $i$ème variable ($c^{(i)} = (c_1, \cdots, c_i)$). Notons $\bar{c}^{(i)} = (c_{i+1}, \cdots, c_m)$, pour tout élément $x \in \mathbb{F}_2^m$, $\langle c, x \rangle = \langle c^{(i)}, x^{(i)} \rangle \oplus \langle \bar{c}^{(i)}, \bar{x}^{(i)} \rangle$. Ainsi :

$$|\hat{G}(c)| = |\sum_{s \in \mathbb{F}_2^{m-i}} (-1)^{\langle \bar{c}^{(i)}, s \rangle} \sum_{r \in \mathbb{F}_2^i} (-1)^{g(r,s) \oplus \langle c^{(i)}, r \rangle}| \leq \sum_{s \in \mathbb{F}_2^{m-i}} |(-1)^{\langle \bar{c}^{(i)}, s \rangle} \sum_{r \in \mathbb{F}_2^i} (-1)^{g(r,s) \oplus \langle c^{(i)}, r \rangle}|$$

où $r = (r_1, \cdots, r_i)$ et $s = (s_{i+1}, \cdots, s_m)$.
Enfin nous avons, pour tout $c \in L_\epsilon(g)$,

$$\sum_{s \in \mathbb{F}_2^{m-i}} |\sum_{r \in \mathbb{F}_2^i} (-1)^{g(r,s) \oplus \langle c^{(i)}, r \rangle}| \geq 2\epsilon 2^m \tag{3.22}$$

L'équation 3.22 nous donne un test pour valider ou invalider l'insertion du préfixe $c^{(i)}$ à la liste $\mathcal{L}_i^g(\epsilon)$. L'algorithme se déroule donc comme suit :

Initialisation Construire la liste $\mathcal{L}_0^g(\epsilon)$, vide.

Étape i Supposons que la liste $\mathcal{L}_{i-1}^g(\epsilon)$ contient, au moins, pour tout élément $c \in L_\epsilon(g)$ le préfixe $c^{(i-1)}$.
Pour chaque élément $c^{(i-1)} \in \mathcal{L}_{i-1}^g(\epsilon)$ construire le couple $\{(c^{(i-1)}, 0), (c^{(i-1)}, 1)\}$ et faire le test défini par l'équation 3.22 pour chacun d'eux. En déduire la liste $\mathcal{L}_i^g(\epsilon)$.

Fin La liste $\mathcal{L}_m^g(\epsilon)$ vaut exactement $L_\epsilon(g)$.

Qui plus est, il est possible de prouver, qu'à chaque étape i de l'algorithme, la liste $\mathcal{L}_i^g(\epsilon)$ est limitée en ordre de grandeur par $O(1/\epsilon^2)$ (voir [110]).

Complexité Lorsque m est petit, l'évaluation de $\sum_{s\in\mathbb{F}_2^{m-i}} |\sum_{r\in\mathbb{F}_2^i} (-1)^{g(r,s)\oplus\langle c^{(i)},r\rangle}|$ dans l'équation 3.22 peut se faire en utilisant une transformée de Fourier rapide spécialisée (voir [102]). Lorsque m est trop grand pour l'évaluer exactement, il faut se contenter de calculer $\sum_{s\in S} |\sum_{r\in R} (-1)^{g(r,s)\oplus\langle c^{(i)},r\rangle}|$ où S et R sont deux sous-ensembles choisis aléatoirement respectivement dans $\mathbb{F}_2^{m-i}$ et $\mathbb{F}_2^i$.

En pratique cet algorithme a une complexité de l'ordre de $O(m/\epsilon^2)$. Ce résultat est détaillé dans [77].

3.2.3 Codes de Reed-Solomon

Les *codes de Reed-Solomon*, proposés en 1960 par I. Reed et G. Solomon [175], sont en fait une classe de codes de Reed-Muller. Il s'agit des codes de Reed-Muller sur une seule variable ($m = 1$) pour lesquels la taille de bloc vaut $n = q - 1$ pour un code de Reed-Solomon défini sur $\mathbb{F}_q$. Notons $\mathfrak{RS}(r, q-1)_q$ le code de Reed-Solomon représenté par les polynômes univariés de degré au plus r de $\mathbb{F}_q[X]$, la dimension de $\mathfrak{RS}(r, q-1)_q$ vaut $k = \binom{r+1}{r} = r + 1$. Contrairement au cas général des codes de Reed-Muller, ces codes ont une distance minimale optimale, et donc peuvent corriger (dans le sens du décodage unique) un nombre maximal d'erreurs par rapport au nombre de symbole de redondance $n - k = n - r - 1$.

Remarque 6 *Une autre définition des codes de Reed-Solomon est plus souvent utilisée car elle permet une étude plus approfondie. Nous n'en parlerons pas ici car elle nécessite d'introduire trop de notions complémentaires inutiles dans ce document. Notons tout de même que cette définition place ces codes dans la famille bien connue des codes BCH (du nom des auteurs : Bose, Ray-Chaudhuri et Hocquenghem), permettant ainsi de construire un décodage très efficace. Par ailleurs, elle permet aussi de construire facilement des codes de Reed-Solomon systématiques, ce qui n'est pas le cas avec la construction présentée ici.*

Théorème 2 *Les codes de Reed-Solomon sont MDS.*

Preuve La preuve est triviale en utilisant les résultats sur les codes de Reed-Muller présentés à la section précédente. Ainsi, nous savons que la distance minimale d'un code de Reed-Muller vaut au moins $n(1-r/q) = n - \frac{nr}{n+1} \geq n - r$, donc pour un code de Reed-Solomon $[n,k,d]_q$, nous avons $d \geq n - r$. Et comme $k = r + 1$, nous avons finalement $d \geq n - r \geq n - k + 1$ et donc $d = n - k + 1$ (d'après la borne du singleton). □

La structure des codes de Reed-Solomon permet à la fois un codage et un décodage très efficace (quasi-linéaire).

3.2.3.1 Codage et Matrice Génératrice

Soient $(x_1, \cdots, x_n)$ les n éléments non nuls de $\mathbb{F}_q$ ($n = q-1$), soit $m = (m_0, m_1, \cdots, m_{k-1})$ le message original. Le codage dans le code $\mathfrak{RS}(k, q-1)_q$ du message $M(X) = \sum_{i=0} k - 1 m_i \cdot X^i$ (vu cette fois comme un polynôme de degré inférieur à k à coefficients dans $\mathbb{F}_q$), correspond à :

$$\begin{array}{lcl} \mathbb{F}_q^k & \longrightarrow & \mathbb{F}_q^n \\ (m_0, m_1, \cdots, m_{k-1}) & \longrightarrow & (\sum_{i=0}^{k-1} m_i \cdot x_1^i, \cdots, \sum_{i=0}^{k-1} m_i \cdot x_n^i) \end{array}$$

De plus, $\mathbb{F}_q$ étant un corps fini, il possède un élément primitif, α, tel que $\{\alpha^i\}_{0 \leq i \leq n-1}$ parcours l'ensemble des éléments non nuls de $\mathbb{F}_q$. En réordonnant la liste $(x_1, \cdots, x_n)$, la fonction de codage devient :

$$\begin{array}{lcl} \mathbb{F}_q^k & \longrightarrow & \mathbb{F}_q^n \\ (m_0, m_1, \cdots, m_{k-1}) & \longrightarrow & (\sum_{i=0}^{k-1} m_i, \sum_{i=0}^{k-1} m_i \cdot \alpha^i, \cdots, \sum_{i=0}^{k-1} m_i \cdot (\alpha^{n-1})^i) \end{array}$$

Qui n'est autre que la transformée de Fourier discrète (noté DFT pour "Discrete Fourrier Transform") de m et qui, dans sa version rapide, a une complexité en $O(n \log n)$. La matrice génératrice est alors la matrice de Vandermonde de taille $k \times n$ et dont la deuxième ligne est $(x_1, x_2, \cdots, x_n)$.

3.2.3.2 Décodage unique des codes de Reed-Solomon

Pour décoder il faut donc trouver le polynôme de degré inférieur à k coïncidant avec les évaluations reçues et ceci en le plus grand nombre de points possible. En effet le mot y reçu correspond à l'évaluation du message original $M(X)$ en n points, si il n'y a pas d'erreurs, une simple interpolation (une transformée de fourrier discrète inverse) permet de retrouver le polynôme $M(X)$, en cas d'erreur, il faut trouver le polynôme de degré inférieur à k le plus proche. Nous proposons ici une approche peu standard mais qui a le mérite d'être simple, efficace et facilement généralisable (voir section 3.2.4). Il s'agit d'un algorithme de décodage *sans syndrome* que nous avons retrouvé sous le nom de l'algorithme de Gao dans la littérature [80].

Soit $Y(X)$ le polynôme de degré inférieur strictement à n trouvé par interpolation des évaluations bruitées reçues (DFT inverse). On peut noter $Y(X) = M(X) + E(X)$ où $E(X)$ un polynôme de degré strictement inférieur à n représentant l'erreur.

Remarque 7 *On peut supposer ici que $E(X)$ n'est pas nul, car sinon le polynôme $Y(X)$ reconstruit est de degré inférieur à k, le décodage est fini $Y(X) = M(X)$.*

Soit $t = \frac{q-k-1}{2}$ le nombre maximal d'erreurs que peut corriger le code $\mathfrak{RS}(k, q-1)_q$, nous supposons donc qu'il existe l ($1 \leq l \leq t$) indices $(k_1, \cdots, k_l)$ tels que $E(x_{k_i}) \neq 0$, où $(x_1, \cdots, x_n)$ représente toujours l'ensemble des éléments non nuls de $\mathbb{F}_q$. Pour alléger les formules nous noterons $\Pi(X) = \prod_{i=1}^{n}(X - x_i)$, $\Pi_F(X) = \prod_{i \in \{k_1, \cdots, k_l\}} (X - x_i)$ et $\Pi_V(X) = \prod_{i \notin \{k_1, \cdots, k_l\}} (X - x_i)$.
Nous pouvons ainsi écrire le polynôme $E(X)$ sous la forme suivante :

$$E(X) = Z(X) \cdot \Pi_V(X)$$

où $Z(X)$ est un polynôme de degré au plus t.
Pour des raisons évidentes, $Z(X)$ est premier avec le polynôme $\Pi(X)$ (ils n'ont aucune racine commune), nous en déduisons donc que

$$PGCD(\Pi(X), E(X)) = \Pi_V(X)$$

où $PGCD(.,.)$ correspond au plus grand diviseur commun de deux polynômes. Ainsi, dans la séquence des reste de la division euclidienne de $\Pi(X)$ et $E(X)$, le dernier reste non nul est $\Pi_V(X)$.

Théorème 3 *$\Pi_F(X) \cdot M(X)$ apparaît, à un facteur scalaire près, dans la suite des restes de l'algorithme d'Euclide appliqué à $\Pi(X)$ et $Y(X)$. De plus, il s'agit du premier reste de degré inférieur strictement à $k + t$.*

Preuve Nous aurons besoin des deux lemmes suivant énoncés et démontrés dans [81], les preuves, assez simples, ne sont pas données ici. Soient f et g deux polynômes à coefficients dans un corps $\mathbb{K}$ de degrés respectifs n et $m < n$. Notons $\{r_i\}$ la suite restes de l'algorithme d'Euclide appliqué à f et g et $\{s_i\}$ et $\{t_i\}$ la suite des coefficients de l'algorithme d'Euclide étendu tels que $\forall i, s_i \cdot f + t_i \cdot g = r_i$. Enfin $r_{N+1} = 0$ le premier reste nul et $deg(0) = -\infty$.

Lemme 2 *On suppose que $r = s \cdot f + t \cdot g$ avec $deg(r) + deg(t) < n$. Soit i compris entre 1 et $N + 1$ tel que $deg(r_i) \leq deg(r) < deg(r_{i-1})$. Alors il existe $\gamma \in \mathbb{K}$ tel que $r = \gamma \cdot r_i$, $s = \gamma \cdot s_i$ et $t = \gamma \cdot t_i$*

Lemme 3 *Soit $0 \leq v \leq m$. Alors v n'apparaît pas dans la séquence des degrés des restes de l'algorithme d'Euclide appliqué à f et g si et seulement si, il existe s, t dans $\mathbb{K}[X]$ tels que $t \neq 0$, $deg(s) < m - v$, $deg(t) < n - v$ et $deg(s \cdot f + t \cdot g) < v$.*

Remarquons maintenant que

$$Z(X) \cdot \Pi(X) - \Pi_F(X) \cdot Y(X) = \Pi_F(X) \cdot M(X)$$

Nous pouvons appliquer le lemme 2 en posant $f = \Pi(X)$, $g = Y(X)$, $r = \Pi_F(X) \cdot M(X)$, $s = Z(X)$ et $t = \Pi_F(X)$ car nous avons bien $deg(f) = n > deg(g)$ et $deg(r) + deg(t) < n$ puisque $deg(t) = l$ et $deg(r) < l + k$ (rappelons que $n = 2t + k$).

Soit $0 < i \leq N+1$ tel que $deg(r_i) \leq deg(r) < deg(r_{i-1})$, alors il existe un polynôme γ tel que $r = \gamma \cdot r_i$, $s = \gamma \cdot s_i$ et $t = \gamma \cdot t_i$. Or nous savons que $Z(X)$ et $\Pi_F(X)$ et donc s et t sont premier entre eux, donc $\gamma = \lambda \in \mathbb{F}_q$ est un scalaire. Enfin nous en déduisons bien que $\lambda \cdot \Pi_F(X) \cdot M(X)$ apparaît bien dans la suite des reste de l'algorithme d'Euclide appliqué à $\Pi(X)$ et $Y(X)$.

La deuxième partie du théorème 3 se prouve par application directe du lemme 3 en gardant f, g, r, s et t fixés comme au-dessus. Nous en déduisons que v n'apparaît pas dans la suite des degrés des restes lorsque

$$\begin{aligned} v &< deg(Y(X)) - deg(Z(X)) \\ v &< deg(\Pi(X)) - deg(\Pi_F(X)) \\ v &> deg(\Pi_F(X)) + deg(M(X))) \end{aligned}$$

et donc, comme $deg(Y(X)) = deg(Z(X)) + deg(\Pi_V(X))$,

$$\begin{aligned} v &< n - l \\ v &< n - l \\ v &> l + k - 1 \end{aligned}$$

Nous avons donc bien que le premier reste de degré inférieur strictement à $k + t$ est de degré, au plus, $l + k - 1$ et donc est, à un facteur scalaire près, $\Pi_F \cdot M(X)$. □

Le théorème 3 nous permet de conclure. En appliquant l'algorithme d'Euclide étendu à $\Pi(X)$ et $Y(X)$, et en s'arrêtant lorsque le degré du reste est inférieur à $k + t$, nous obtenons les deux polynômes $\alpha(X)$ et $\beta(X)$ tels que

$$\alpha(X) \cdot \Pi(X) + \beta(X) \cdot Y(X) = \lambda \cdot \Pi_F(X) \cdot M(X)$$

et donc

$$\alpha(X) \cdot \Pi(X) + \beta(X) \cdot E(X) = 0$$

finalement, pour retrouver $M(X)$, il suffit de calculer

$$M(X) = Y(X) + \frac{\alpha(X) \cdot \Pi(X)}{\beta(X)}$$

Coût de l'algorithme Le coût de l'algorithme de décodage correspond donc au coût d'une interpolation et d'un algorithme d'Euclide. Dans le cas des codes de Reed-Solomon, comme pour le codage, l'interpolation de $Y(X)$ peut se faire avec une complexité de $O(n \log n)$ à l'aide de la DFT inverse. D'autre part, l'algorithme d'Euclide peut lui aussi être fait rapidement sous la forme d'un *HGCD* qui a alors une complexité, en moyenne, de $O(n \log^2 n)$ (voir par exemple [204]). Pour plus d'informations, la complexité de l'algorithme de Gao est étudiée avec précision dans [53].

Remarque 8 *Il existe un autre décodage efficace des codes de Reed-Solomon, il se base sur la deuxième façon de représenter ces codes (remarque 6), nous nous contenterons de donner sa complexité. Le lecteur intéressé en trouvera une description soignée dans [101] : le décodage*

efficace de Reed-Solomon se base sur l'algorithme de Berlekamp-Massey [146] dont il emprunte la complexité : $O(n \log^2 n)$.
La complexité est donc la même, en ordre de grandeur, que le décodage présenté ici, en fait il s'agit d'un algorithme très proche et d'ailleurs l'équivalence entre l'algorithme de Berlekamp-Massey et l'algorithme d'Euclide est connue depuis longtemps [71].

Il est aisé de constater que ces algorithmes de codage et décodage s'appliquent aussi bien pour tout $n \leq q-1$, en effet, le nombre de points d'évaluation n'est pas dépendant de la valeur de q (il faut bien évidement que les points soient distincts et donc $n \leq q-1$). Nous obtenons, pour $n < q-1$ des codes de Reed-Solomon *raccourcis*. Notons tout de même que pour avoir un codage efficace en $n \log n$ avec la une généralisation de la DFT, il est nécessaire de choisir un élément du corps d'ordre supérieur ou égale à n et d'utiliser des points qui sont des puissances *consécutives* de cette élément. Dans le cas contraire, il est toujours possible d'utiliser la DFT mais sur l'ensemble des points de $\mathbb{F}_q$ (puis supprimer les points non utilisés), la complexité est alors $(q-1)\log(q-1)$ (qui n'est pas forcément plus efficace que le cas général). Dans le cas général, sans passer par un élément d'ordre supérieur à n, l'évaluation comme l'interpolation polynomiale peut se faire rapidement (par généralisation de la DFT) avec une complexité de $O(n \log^2 n)$ et un précalcul en $O(n \log^2 n)$ (voir, par exemple, le cours de Bruno Salvy sur le sujet[1]).

3.2.4 Codes par interpolation de polynômes univariés

Nous considérons ici une généralisation simple des codes de Reed-Solomon.

On peut facilement remarquer que les algorithmes de codage et décodage présentés pour les codes de Reed-Solomon ne reposent pas sur la structure des corps fini, en fait la seule nécessité est que l'ensemble des polynômes considérés soit un anneau euclidien (i.e. muni de la division euclidienne), c'est le cas pour tout ensemble de polynômes à coefficients dans un corps commutatif. Ainsi, les codes de Reed-Solomon s'étendent, avec les algorithmes de codage et décodage proposés (et donc leur complexité algorithmique), lorsque le message $M(X)$ est considéré comment un polynôme à coefficients entiers, réels, complexes, etc ...

Remarque 9 *Les codes par interpolation de polynômes univariés rentrent dans une famille de codes appelés les* codes par résidus polynomiaux*, que nous aborderons dans le chapitre 9.*

3.2.5 Codes LDPC

Les codes LDPC ("Low Density Parity-Check" en anglais) sont des codes linéaires proposés par Robert G. Gallager en 1960 [79], ils tiennent leur nom du faible nombre d'entrées non nulles dans leur matrice de parité. En effet, sont considérés LDPC, tous les codes dont la matrice de parité a moins d'entrées non nulles que de 0.

[1] `http://algo.inria.fr/salvy/mpri/Cours5.pdf`

Il est généralement difficile d'évaluer la distance minimale d'un code (problème NP-Hard) et la structure des codes LDPC ne leur donne pas de moyen de le faire (contrairement aux codes présentés dans les sections précédentes). Cette distance minimale n'a pas de raison d'être grande, les expériences le confirment [99]. Par contre il a été montré que certain codes LDPC atteignent une capacité de décodage optimale lorsque les erreurs sont aléatoirement distribuées. Dans ces conditions, certains codes LDPC font parti des codes les plus efficaces et performants aujourd'hui, la structure de leur matrice de parité permettant de construire des algorithmes de décodage probabiliste de complexité linéaire [68, 180].

Bien que la complexité du décodage soit très faible ($O(n)$), le codage quant à lui peut coûter plus cher ; il correspond à un produit de matrices ou à la résolution d'un système linéaire. Il existe des sous-classes de codes LDPC qui possèdent des algorithmes de codage de complexité linéaire, les codes "Repeat-Accumulate" [70] par exemple ou encore, les codes LDGM [78]("Low Density Générator Matrix" en anglais) qui, comme leur nom l'indique, ont à la fois une matrice de parité creuse mais aussi une matrice génératrice creuse.

II

Cryptologie à clé secrète

Primitives cryptographiques : Traitement parallèle 4

Sommaire

4.1 Stratégie de parallélisme pour chiffrements symétriques et hachage cryptographique 52

4.1.1 Chiffrement par bloc 52

4.1.1.1 Algorithme de chiffrement 52

4.1.1.2 Modes Opératoires 53

4.1.2 Fonction de Hachage 55

4.1.2.1 Fonction de compression 56

4.1.2.2 Modes Opératoire 57

4.1.3 Conclusions 58

4.2 Implémentation efficace de codes parallèles 60

4.2.1 Principe du vol de travail 60

4.2.2 Ordonnancement dynamique de tâches par vol de travail 60

4.3 Application aux chiffrements symétriques avec mode opératoire Compteur 61

La cryptographie à clé secrète est utilisée en permanence (contrairement à la crypto publique qui n'intervient que de façon ponctuelle pour échange de clés ou authentification), il est donc important d'en optimiser les performances afin qu'elle ne ralentisse pas l'application qui la supporte. Dans ce cadre nous avons étudié l'optimisation de fonctions de chiffrement en implémentation logicielles et matérielles (cf. chapitre 7). Alors que ces techniques, à chiffre fixé, sont limités par une borne inférieur incompressible (nombre d'opérations de la fonction), nous nous intéressons maintenant à la parallélisation de fonctions cryptographiques, prenant ainsi avantage des capacités des machines actuelles : multi-processeurs/coeurs, GPU.
Ce chapitre étudie, sur des construction génériques, la parallélisation "haut niveau" des chiffrements symétriques et des fonctions de hachages.

4.1 Stratégie de parallélisme pour chiffrements symétriques et hachage cryptographique

La construction "haut niveau" de chiffrements symétriques et fonctions de hachage cryptographiques est faite sur le même modèle. Une fonction limitée par la taille de ses entrées et de ses sorties est itérée pour construire une fonction de chiffrement ou de hachage globale sur des entrées de très grandes tailles (voire non bornées). La méthode d'itération est appelée *mode opératoire*. La sécurité d'une telle fonction globale s'étudie en considérant indépendamment la fonction bloc et le mode opératoire (celui-ci étant considéré sûr lorsque sa résistance cryptographique repose entièrement sur la résistance de la fonction bloc).

Ainsi deux niveaux de parallélisme sont envisageables : soit "bas niveau" sur la *fonction bloc*, soit haut niveau sur le *mode opératoire*. Ces deux niveaux de parallélisme sont complémentaires et non exclusifs. Nous allons voir dans cette section comment ces deux niveaux de parallélismes peuvent-être mis en oeuvre sur une plate-forme multi-coeurs/processeurs. Dans toute cette section nous considérerons que toutes les unités de calcul de la plate-forme sont identiques (même fréquences, même taille de registres, taille de cache, etc ...), dans la section qui suit nous étudierons une gestion efficace du parallélisme au niveau mode opératoire sur plate-formes multi-coeurs ou multi-processeurs lorsque cette hypothèse n'est pas vérifiée.

4.1.1 Chiffrement par bloc

Un chiffre symétrique par bloc, tel que défini dans la section 3.1, est noté $S = (\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$. Dans la grande majorité des cas, l'ensemble des messages clairs et l'ensemble des messages chiffrés sont tels que : $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^n$ (n étant la *taille de bloc* du chiffre). Pour chiffrer, à l'aide de S, un message de longueur $m > n$, il est nécessaire d'appeler plusieurs fois l'algorithme de chiffrement E sur des blocs de taille n du message.

4.1.1.1 Algorithme de chiffrement

Il s'agit ici de rendre parallèle l'exécution de la fonction de chiffrement E. Nous avons vu dans le chapitre 3, section 3.1.2 une construction générique d'algorithme de chiffrement symétrique : les réseaux de Substitution-Permutation en rondes itérées. Dans une ronde de chiffrement symétrique de ce type, l'étape la plus lourde est souvent l'application des boîtes de substitution (en particulier en implémentation hardware), cette étape est trivialement parallélisable puisque chaque boîte est appliquée indépendamment sur une sous-partie de l'entrée. Il en va de même avec la fonction de mélange avec la clé de ronde. Par contre la fonction linéaire de diffusion pose un problème. Son rôle est de diffuser la confusion introduite par les boîtes-S, il est donc important qu'elle mixe entre elles les sorties des boîtes-S, introduisant ainsi une grande dépendance de donnée (bête noire du parallélisme). Il faudra donc travailler sur des données intermédiaires synchronisées en milieu et fin de rondes.
Cette découpe est de grain très fin, i.e., les tâches exécutées en parallèle ne comportent que quelques opérations atomiques. Si le coût dû au parallélisme (créations de tâches, ordonnancement des tâches) est trop grand par rapport au coût d'une tâche, notamment pour une exécution logicielle, l'exécution parallèle de l'algorithme de chiffrement est inutile, voir désastreux en terme

de performances. Pour une implémentation matérielle, ce parallélisme interne est souvent utilisé, il s'agit de faire un compromis entre gain de temps et taille du circuit. De nombreuses implémentations matérielles (ASIC ou FPGA) ont été proposées pour les algorithmes de chiffrement les plus connus, tels que AES [46] ou Serpent [65].

4.1.1.2 Modes Opératoires

La figure 4.1 décrit l'exemple le plus simple de *mode opératoire* pour le chiffre S. Il s'agit du *mode EBC* (pour "Electronic CodeBook" en anglais), le message est divisé en $\lceil \frac{m}{n} \rceil$ blocs, indépendamment chiffrés. Le dernier bloc, si il n'est pas de taille n, doit être complété suivant une convention adoptée publiquement ("padding" en anglais). Ce mode d'opération, le plus efficace qui soit, possède un certain nombre de désavantages en termes de sécurité ; entre autres, tout les blocs identiques du message correspondent à des blocs de messages chiffrés identiques, permettant ainsi de reconnaître des paternes dans le message d'origine à partir du message chiffré.

Un mode opératoire plus utilisé est le *mode CBC* (pour "Cipher Block Chaining" en anglais) décrit sur la figure 4.2. L'utilisation du chaînage permet de supprimer une grande partie des failles de sécurité présentes pour le mode ECB. Il est important de faire attention au *vecteur d'initialisation*, bien qu'étant une donnée publique, l'utilisation d'un même vecteur d'initialisation pour deux messages différents sans changer la clé secrète donne lieu à des attaques par *re-jeu* ou par *le milieu* ("man in the middle" en anglais). Le principal désavantage du mode CBC est qu'une erreur dans le traitement d'un bloc du message affecte tout les blocs suivants et, pour les mêmes raisons, le chiffrement/déchiffrement d'une partie du message ne peut se faire sans avoir au préalable chiffré/déchiffré les blocs précédents. Plus important pour nous ici, le chiffrement ne peut se paralléliser.

Le NIST ("National Institute of Standart and Technology") recommande 5 modes opératoires pour les chiffrements symétriques [1]. Ils ont chacun leurs avantages et inconvénients [196], le dernier en date est le mode *Compteur* (noté simplement *CTR* dans la suite).

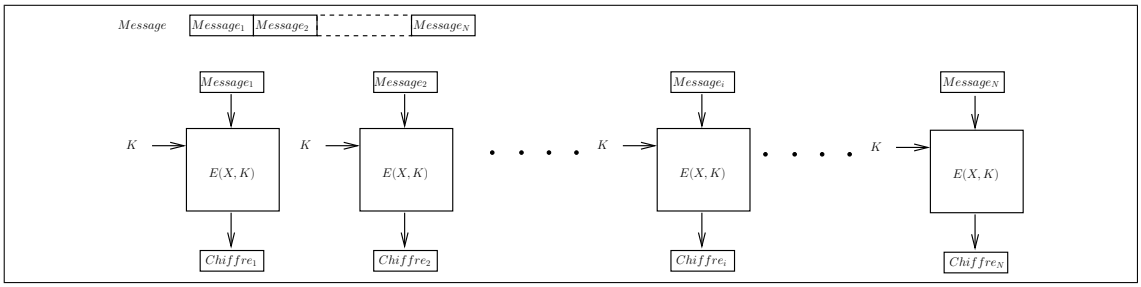

FIG. 4.1 – Mode Opératoire ECB

Mode Compteur Le *mode CTR* (figure 4.3) possède des avantages particulièrement intéressants pour notre analyse, ainsi il permet un traitement complètement parallèle tout en ayant une *sécurité prouvée* si l'on suppose qu'une valeur initiale de compteur n'est utilisée qu'une seule fois pour

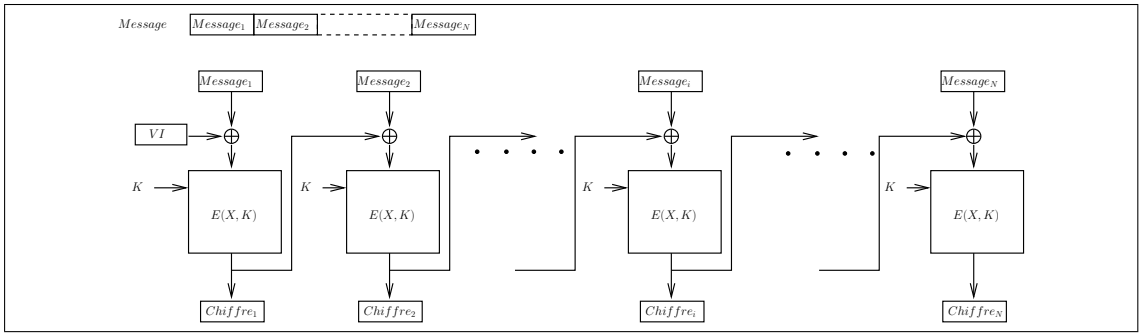

FIG. 4.2 – Mode Opératoire CBC

une clé secrète donnée [20, 93]. Pour un mode opératoire, avoir une sécurité prouvée signifie que le mode est robuste sous l'hypothèse que le chiffrement par bloc qu'il emploie est lui-même sûr, i.e., l'utilisation du chaînage n'introduit pas faiblesse.

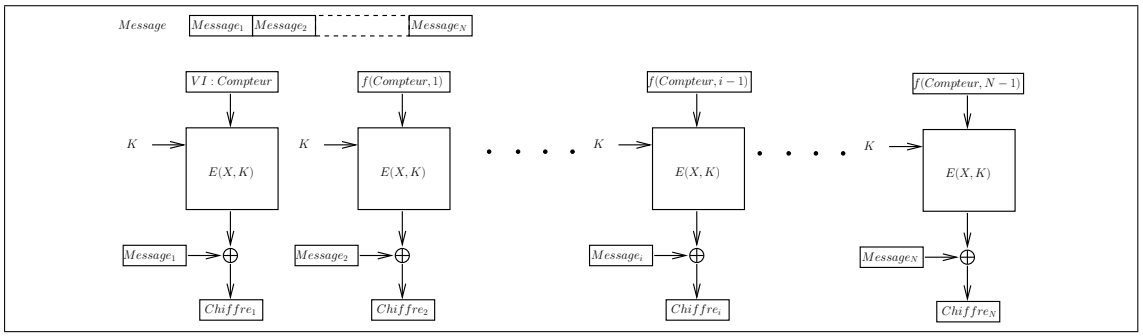

FIG. 4.3 – Mode Opératoire CTR

Remarque 10 *Sur la figure 4.3, la fonction $f(compteur, i)$ est la fonction d'incrémentation du compteur. Cette fonction doit être simple pour ne pas ajouter de sur-coût au calcul et avoir une grande période par rapport à i, i.e., $f(compteur, i) = f(compteur, j)$ si et seulement si $i = j$ ou $j >> i$. En pratique et dans la suite, l'addition modulo un grand entier est utilisée (e.g., $f(x, i) = x + i \mod 2^{64}$).*

Le mode de chaînage CTR est souvent vu comme la construction d'un générateur pseudo-aléatoire à partir d'un algorithme de chiffrement par bloc. En effet, il consiste à générer une suite de blocs de données (clé pseudo-aléatoire) à partir de la valeur initiale du compteur (graine). Ces données servent alors à masquer le message clair pour obtenir le message chiffré (le masquage est fait par "ou exclusif bit à bit"). L'intérêt de cette technique est triple, premièrement il est possible de pré-calculer la clé pseudo-aléatoire avant l'arrivée d'un message clair ou chiffré à traiter, deuxièmement elle crée une fonction de chiffrement involutive, une implémentation peut servir à la fois au chiffrement et au déchiffrement, enfin, le message chiffré est de la même taille que le message clair, il n'est plus nécessaire de compléter le dernier bloc du message.

Par contre, comme c'est le cas pour les autres modes de chaînage hormis ECB, la synchronisation est primordiale entre l'envoyeur et le receveur d'un message, une perte de paquet rend le déchiffrement impossible.

Algorithme Séquentiel L'algorithme 1 présente le traitement séquentiel, il nécessite d'entretenir une variable de compteur.

Algorithme 1: Mode Compteur, algorithme séquentiel

Entrée : VI : Valeur Initiale du compteur
$\{M_i\}_{1\leq i\leq N}$: Message en N blocs de taille n
K : clé secrète
Sortie : $\{C_i\}_{1\leq i\leq N}$: Message chiffré

$Compteur \leftarrow VI$;
pour $i = 1 \cdots N$ **faire**
  $C_i \leftarrow M_i \oplus E(Compteur, K)$;
  $Compteur \leftarrow Compteur + 1$;
retourner $\{C_i\}_{1\leq i\leq N}$

Algorithme Parallèle L'algorithme 2 présente le traitement parallèle statique, les blocs sont répartis également entre les p différentes ressources, considérées identiques ici. Il nécessite de diviser les données en p parts égales et d'initialiser la valeur de chaque compteur, chaque ressource entretient une variable de compteur. A la fin de son calcul, chaque ressource renvoie un résultat qu'il faut placer au bon endroit dans le message de sortie.

Temps d'exécution Le temps d'exécution sur p processeurs de l'algorithme 2 est de l'ordre de :

$$T_{statique} = \frac{1}{\Pi}\left(\frac{W_{seq}}{p}\right) + O(p)$$

où Π est la fréquence des processeurs et W_{seq} est le nombre total d'opérations intervenant dans l'algorithme séquentiel 1.

4.1.2 Fonction de Hachage

De même que les chiffres symétriques, les fonctions de hachage cryptographiques représentent un outil indispensable à la sécurité des systèmes d'information. Elles sont principalement utilisées pour la *signature électronique* de documents ainsi que pour contrôler leur intégrité.

Définition 17 (Fonction de compression cryptographique) *Une fonction f de $\mathbb{F}_2^m$ dans $\mathbb{F}_2^n$ est une fonction de compression cryptographique si et seulement si*

- *Collision : Trouver un couple $(x, x') \in (\mathbb{F}_2^m)^2$ tel que $h(x) = h(x')$ est difficile (complexité de l'ordre de $O(2^{n/2})$)*

Algorithme 2: Mode Compteur, algorithme parallèle statique

Entrée : VI : Valeur Initiale du compteur
$\{M_i\}_{1\leq i\leq N}$: Message en N blocs de taille n
K : la clé secrète
p : nombre de processeurs
Sortie : $\{C_i\}_{1\leq i\leq N}$ Message chiffré

$Compteur_1 \leftarrow VI$;
$\{M_k^1\}_{1\leq k\leq \lceil \frac{N}{p} \rceil} \leftarrow \{M_k\}_{1\leq k\leq \lceil \frac{N}{p} \rceil}$;
pour $i = 2 \cdots p$ **faire**
  $Compteur_i \leftarrow Compteur_{i-1} + \lceil \frac{N}{p} \rceil$;
  $\{M_k^i\}_{1\leq k\leq \lceil \frac{N}{p} \rceil} \leftarrow \{M_k\}_{(i-1)\lceil \frac{N}{p} \rceil+1\leq k\leq i\lceil \frac{N}{p} \rceil}$;
// **pour chaque** : Lance une nouvelle tâche sur chaque processeur.
pour chaque $i = 1 \cdots p$ **faire**
  // Appel d'une implémentation de l'algorithme séquentiel 1
  $\{C_k^i\}_{1\leq k\leq \lceil \frac{N}{p} \rceil} \leftarrow$ Mode_Compteur($Compteur_i$, $\{M_k^i\}_{1\leq k\leq \lceil \frac{N}{p} \rceil}$, K);
retourner $\{C_i\}_{1\leq i\leq N}$

- *Première Préimage : Soit $y \in \mathbb{F}_2^n$, trouver $x \in \mathbb{F}_2^m$ tel que $h(x) = y$ est difficile (complexité de l'ordre de $O(2^n)$)*
- *Seconde Préimage : Soit $x \in \mathbb{F}_2^m$, trouver $x' \in \mathbb{F}_2^m$ tel que $h(x) = h(x')$ est difficile (complexité de l'ordre de $O(2^n)$)*

Définition 18 (Fonction de hachage cryptographique) *Une fonction de hachage cryptographique h est une fonction de compression pour laquelle la taille des entrées n'est pas fixée. La* taille de résumé *de h est la taille de la sortie de h.*

Pour une présentation détaillée, voir le chapitre sur les fonctions de hachages du livre de Menezes, Oorschot et Vanstone [151].

4.1.2.1 Fonction de compression

Comme pour les algorithmes de chiffrement symétriques, la fonction de compression peut-être parallélisée à grain fin. La construction la plus commune, par itération de rondes est assez proche d'un chiffre par bloc (eux-même pouvant être utilisés comme fonction de compression) et implique aussi une dépendance de données forte et donc une parallélisation limitée. Ce niveau de parallélisation est donc souvent, comme pour les chiffrements par bloc, limité aux implémentations matérielles.
Notons que certaines nouvelles fonctions de hachage proposées dans le concours international SHA3 (voir remarque 11) présentent des rondes particulièrement bien parallélisables, entre autres, MD6 [184, 94], permettant ainsi de rendre efficace une telle parallélisation sur plusieurs coeurs d'un GPU [24].

4.1.2.2 Modes Opératoire

Les fonctions de hachages les plus utilisées de nos jours sont basées sur la structure de chaînage *Merkle-Damgård* proposée par Damgård en 1990 [66] (figure 4.4). Cette structure se parallélise très difficilement puisque chaque itération est dépendante des données calculées à l'itération précédente. Dans le papier original de Damgård, celui-ci propose déjà une construction en *arbre de Merkle* [152] dans l'idée de son implémentation parallèle. Bien que cette version soit tout aussi robuste que la version classique, elle est moins efficace que la première pour un traitement séquentiel. La première version a donc été la plus utilisée et a donné naissance aux familles de fonction de hachage MD et plus tard SHA.

Remarque 11 *Au moment de l'écriture de ce manuscrit, un concours international lancé par le NIST bât son plein. Il s'agit de trouver un nouveau standard pour les fonctions de hachage, successeur de SHA1 et SHA2 qui ont montré quelques signes de faiblesse ces dernières années. Le concours est intitulé SHA3*[1]*, nom du prochain standard, et comporte plusieurs soumissions, parmi elles, la proposition d'une équipe américaine dirigée par Ronald L. Rivest (MD6 [184]) est basée sur le schéma en arbre de Merkle quaternaire.*

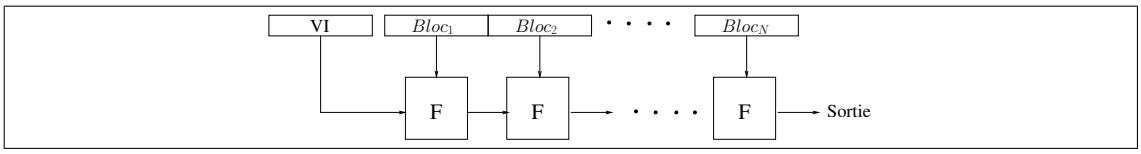

FIG. 4.4 – Mode Opératoire arbre de Merkle

Arbre de Merkle La figure 4.5 décrit une fonction de hachage de type arbre de Merkle binaire basée sur une fonction de compression $F : \mathbb{F}_2^{2n} \leftarrow \mathbb{F}_2^n$. Comme nous l'avions fait pour le mode opératoire Compteur, nous décrivons maintenant l'algorithme séquentiel et parallèle correspondant.

Algorithme Séquentiel L'algorithme 3, par *liste de Radix*, nécessite de gérer une pile de taille $\log(N)$, où N est le nombre de blocs de taille n du message à hacher. Cet algorithme correspond à un traitement "de gauche à droite" de l'arbre de Merkle décrit figure 4.5.

Algorithme Parallèle Un algorithme parallèle statique peut être simplement construit à partir de l'algorithme 3. Il faut diviser le message d'entrée en $p-1$ groupes de 2^l blocs de taille n et un groupe de c ($\leq 2^l$) blocs de telle façon que $N = p \cdot 2^l + c$. Sur chacun des groupes est exécuté indépendamment l'algorithme 3. A la fin de cette étape le haché intermédiaire est de taille p blocs, suivant les dimensions du système, l'étape est ré-itérée sur une sous partie des processeurs ou terminée en séquentiel.

[1] `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`

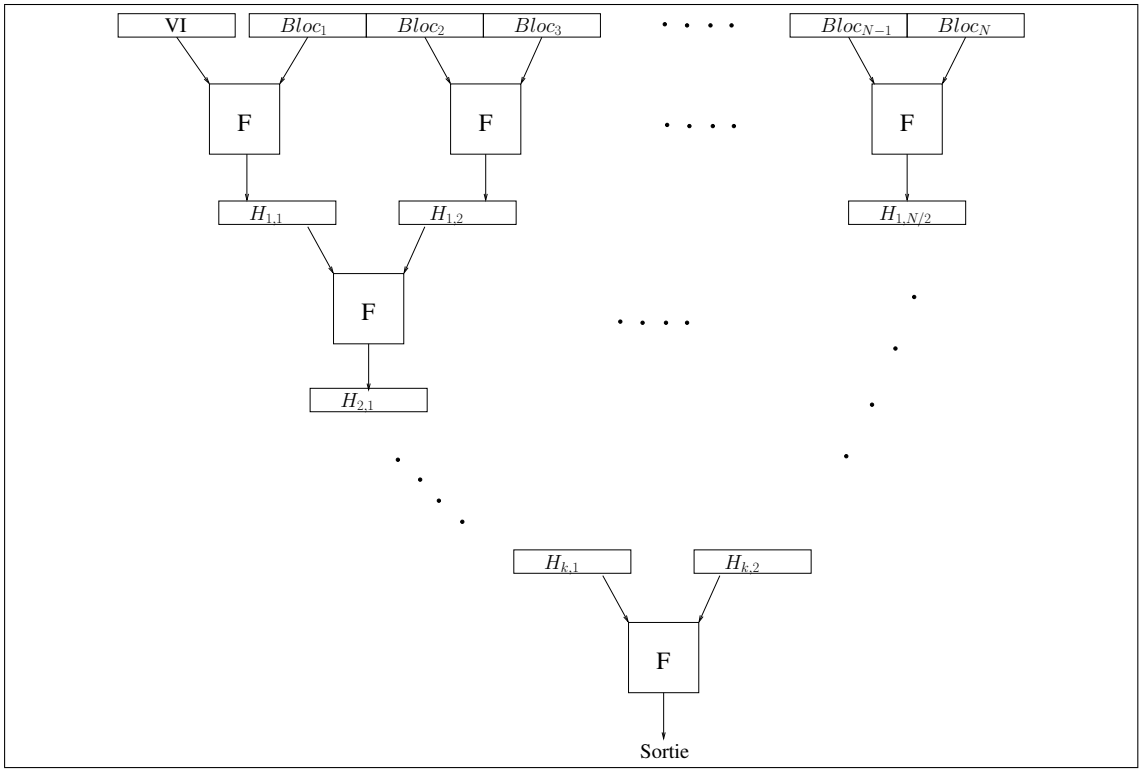

FIG. 4.5 – Mode Opératoire Arbre de Merkle

Temps d'exécution De même que pour l'étude de la section précédente sur le mode opératoire Compteur, le temps d'exécution de l'algorithme parallèle statique sur p processeurs est de l'ordre de :

$$T_{statique} = \frac{1}{\Pi}\left(\frac{W_{seq}}{p}\right) + O(p)$$

où Π est la fréquence des processeurs et W_{seq} est le nombre total d'opérations intervenant dans l'algorithme séquentiel 3.

4.1.3 Conclusions

Deux stratégies de parallélisation s'offrent aux chiffres symétriques et, de façon assez similaire, aux fonctions de hachage cryptographiques. Ces deux stratégies, complémentaires et non exclusives, correspondent en fait à deux niveaux bien distincts de parallélisation. Un niveau bas de parallélisation à grain fin adapté à une implémentation matérielle efficace de la primitive cryptographique et un niveau haut, adapté à une exécution sur des unités de calcul multiples.
Plusieurs études de parallélisation de primitives cryptographiques sur machines multi-coeurs/multi-processeurs (parallélisation haut niveau donc) ont été proposées. Elles reposent principalement sur les algorithmes parallèles simples proposées dans les sections précédentes, agrémentées, pour les chiffrements symétriques, d'une représentation des blocs ingénieuse dû à Biham [26] nommée *bit slicing* (voir par exemple [211, 82, 150]). Nous nous intéressons, dans la section suivante, à

Algorithme 3: Arbre de Merkle, algorithme séquentiel

Entrée : VI : Valeur Initiale de taille n
$\{M_i\}_{1 \leq i \leq N}$: Message en N blocs de taille n
Sortie : R : Résumé de taille n

TableVide $T[\lceil \log(N) \rceil + 1]$;
$T[0] \leftarrow VI$;
pour $i = 1 \cdots N$ **faire**
  $Variable \leftarrow M_i$;
  pour $k = 0 \cdots \lceil \log(N) \rceil$ **faire**
    si $T[k]$ *est occupé* **alors**
      $Variable \leftarrow F(T[k]||Variable)$;
      $T[k] \leftarrow \emptyset$;
    sinon
      $T[k] \leftarrow Variable$;
      break;

// Tout les blocs ont été traités, il ne reste plus qu'à vider la pile (T).
// La première case occupée de la pile est à l'indice k.
$Variable \leftarrow T[k]$;
pour $j = k + 1 \cdots \lceil \log(N) \rceil$ **faire**
  si $T[j]$ *est occupé* **alors**
    $Variable \leftarrow F(T[j]||Variable)$;

retourner $Variable$

un point quelque peu négligé dans la recherche sur ce sujet et pourtant très important dans ce contexte : la perturbation et l'hétérogénéité des unités de calcul.

Nous avons vu un algorithme parallèle simple pour le mode opératoire Compteur et pour l'arbre de Merkle, ces algorithmes sont très efficaces dans le cadre de systèmes répartis homogènes et non perturbés. En effet, la charge de travail est répartie également entre tout les processeurs et ne peut être modifiée en cours d'exécution. Ainsi, il suffit d'un processeur récalcitrant, de plus faible vitesse que les autres, pour ralentir l'ensemble du calcul : le temps d'exécution global est dépendant de la fréquence minimale des processeurs utilisés pour le calcul.
Ces algorithmes sont proches de l'optimal lorsqu'il sont implémentés pour des machines à processeurs identiques et exclusivement dédiées à un calcul donné (chiffrement ou hachage). Cette hypothèse n'est vérifiée que dans des cas particuliers, nous allons voir comment répartir efficacement, et en cours d'exécution, la charge de travail entre les processeurs, de façon à ce que l'algorithme s'adapte à des perturbations, à des systèmes hétérogènes (processeurs avec différentes puissances de calcul) ou encore, lorsque la fréquence des processeurs varie au cours du temps.

4.2 Implémentation efficace de codes parallèles

La répartition dynamique de travail peut être gérée efficacement, dans certains cas et en particuliers dans ceux qui nous intéressent ici, par une technique de *vol de travail* entre processeurs.

4.2.1 Principe du vol de travail

Le principe du vol de travail est le suivant : Lorsqu'un processeur n'a plus de travail à exécuter, il vole du travail à un processeur choisi aléatoirement. Cette technique permet d'assurer une performance proche de l'optimal dans le contexte de systèmes répartis hétérogènes [40]. Ce principe d'ordonnancement dynamique de tâches est une source de recherche active, plusieurs implémentations ont été développées , notons entre autres les logiciels Cilk [39], IntelTBB [176] et Kaapi [185, 83].

Considérons l'algorithme récursif 4 pour une fonction de hachage suivant un arbre de Merkle binaire (avec $M_0 = VI$, la valeur initiale). Pour simplifier la présentation de l'algorithme il a été supposé que le nombre de blocs de données est une puissance de 2, il est toujours possible de le modifier pour s'affranchir de cette hypothèse sans changer en ordre de grandeur sa complexité (en temps d'exécution et en mémoire). Un algorithme identique peut être construit pour un chiffrement symétrique avec mode opératoire Compteur.

Algorithme 4: Arbre de Merkle, algorithme récursif

`Hachage_Recursif`$(\{M_i\}_{0\leq i<N}, N)$:
Entrée : $\{M_i\}_{0\leq i<N}$: Données d'entrées divisée en $N = 2^L$ blocs de taille n
N : Nombre de blocs de données
Sortie : R : Résumé

si $N = 2$ **alors**
  $R \leftarrow F(M_0||M_1)$;
  retourner R;
$R_0 \leftarrow$ `Hachage_Recursif`$(\{M_i\}_{0\leq i<\frac{N}{2}}, \frac{N}{2})$;
$R_1 \leftarrow$ `Hachage_Recursif`$(\{M_i\}_{\frac{N}{2}+1\leq i<N}, \frac{N}{2})$;
$R \leftarrow F(R_0||R_1)$;
retourner R;

4.2.2 Ordonnancement dynamique de tâches par vol de travail

L'écriture récursive de l'algorithme permet de voir simplement comment le vol de travail est opéré. Chaque processus possède une pile de tâches prêtes et une pile d'exécution, lorsqu'un processus veut exécuter une tâche, il va chercher sa tâche prête la plus récente et l'exécute sur sa pile d'exécution. Une tâche comprent deux types d'instructions :

- Instruction séquentielle de calcul.
- Instruction de création de tâche (`fork, spawn`) ou déblocage de données ou processus (signal, sémaphore, mutex).

Le deuxième type d'instructions créent le parallélisme, une nouvelle tâche créée sera ajoutée sur la pile des tâches prêtes. La succession de création de tâches sur l'ensemble du programme peut toujours être représenté par un graphe binaire dont chaque noeud correspond à une instruction du deuxième type. Notons D la profondeur de ce graphe.
Lorsqu'un processus est oisif, i.e., sa pile de tâche prête est vide, il choisi aléatoirement un processus et récupère la tâche de plus grande profondeur (qui correspond souvent à la tâche la plus ancienne) de sa pile des tâches prêtes en la marquant comme volée. Pour plus de détails se référer à [40, 84, 83].

Le vol de travail sera intéressant si cette profondeur D est petite devant le travail total W (nombre d'instructions du programme). En effet, lorsque les vitesses des processeurs varient dans un intervalle borné, il est possible d'évaluer avec une grande probabilité le nombre de synchronisations (i.e., nombre de vols) : $O(p \cdot D)$, où p est le nombre de processeurs [84]. Garder un nombre de vols faible ($W >> p \cdot D$) est nécessaire pour garantir une exécution efficace : avec une grande probabilité, le temps d'exécution sur p processeurs [21, 84] :

$$T_{vol} = \frac{1}{p \cdot \Pi_{moy}} \left(W + O(p \cdot D)\right)$$

où Π_{moy} est la fréquence moyenne de l'ensemble des p processeurs.
Cette fois-ci le temps d'exécution est dépendant de la fréquence moyenne des processeurs, cette solution est donc bien plus adaptée aux systèmes hétérogènes.

Remarque 12 *Un programme travaillant sur N entrées, aura, au minimum, un travail total $W = O(N)$ et une profondeur minimale $D = \log_2(N)$. Pour notre étude, sur un message de taille N, la profondeur est minimale ($D = \log_2(N)$) en considérant l'algorithme 4 avec, pour chaque appel récursif, une création de tâche. Notre application est donc bien adaptée au vol de travail.*

Vol de travail adaptatif Il peut être remarqué que le travail total W_{par} de l'algorithme 4 est sensiblement plus important que le travail W_{seq} de l'algorithme purement séquentiel, que ce soit pour l'arbre de Merkle ou le mode opératoire Compteur. Ainsi, l'idée développée dans [67] est de profiter de l'algorithme séquentiel efficace sans pour autant perdre le vol de travail. Pour cela il faut remarquer que même en cours d'exécution des algorithmes 1 et 3 séquentiels il est toujours possible d'extraire des tâches indépendantes des opérations en cours. Sans rentrer dans les détails décrits dans [67], il s'agit de ne créer des tâches que lorsqu'il y a un vol, cette technique permet d'obtenir, avec une grande probabilité, un meilleur temps d'exécution :

$$T_{vol\ adaptatif} = \frac{1}{\Pi_{moy}} \left(\frac{W_{seq}}{p} + O(D)\right)$$

4.3 Application aux chiffrements symétriques avec mode opératoire Compteur

Le logiciel Kaapi [83] et la bibliothèque libaws [22], qui implémentent le vol de travail adaptatif, nous ont permis d'évaluer les performances d'un algorithme de chiffrement en mode Comp-

teur exécuté en parallèle. Les tests ont été fait sur deux plate-forme très différentes : une machine multi-coeurs et un machine MPSoC (pour "Multi Processor System on Chip" en anglais).

Remarque 13 *Nous ne présentons pas de résultats pour les fonctions de hachage par arbre de Merkle, pour de tels résultats, se référer par exemple au travail de Rivest et al. [184] sur une implémentation parallèle par vol de travail de la fonction MD6 (logiciel Cilk).*

La première expérience, sur la plate-forme multi-coeurs[2] avec Kaapi, montre l'utilité de la parallélisation : Il s'agit de chiffrer un flot de données, le débit de lecture des données correspond à une lecture sur disque. Les figures 4.6 et 4.7 représentent le gain en temps par rapport à l'exécution séquentielle ("relative speedup" en anglais) en fonction du nombre de coeurs utilisés (maximum 8). La figure 4.6 donne les résultats lorsque le temps de lecture sur disque est négligé, il est aisé de constater que la parallélisation est très efficace, proche de l'optimal (le décrochement aux alentours de 4 coeurs est dû à l'architecture de l'Itanium : les 4 bi-coeurs forment deux groupes de 2 bi-coeurs). La figure 4.7 reprend les résultats, cette fois-ci en prenant en compte le temps de lecture sur le disque. L'écrasement de la courbe est donc dû à la lecture, nous en déduisons que le parallélisme permet rapidement de masquer le temps chiffrement. Ce résultat n'a rien de surprenant puisque la lecture ne peut être parallélisée, tous les processeurs lisent au même endroit, simulant ainsi l'arrivée d'un flot de données chiffrées sur un réseau.

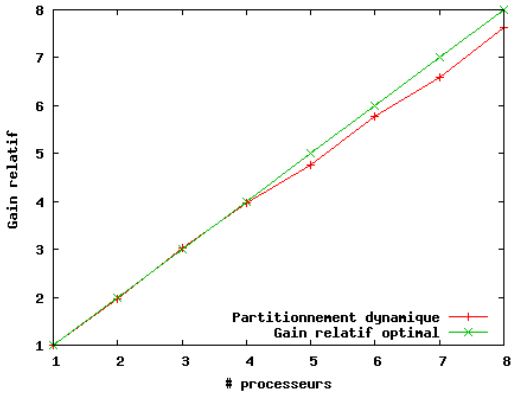

FIG. 4.6 – Gain en temps d'exécution sans coût de lecture

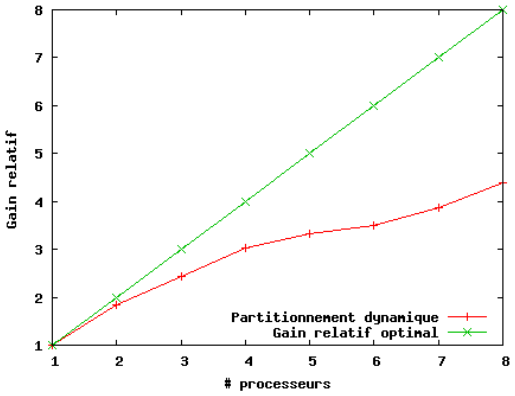

FIG. 4.7 – Gain en temps d'exécution avec coût de lecture

Les tests sur la plate-forme MPSoC[3], avec libaws, permettent de mettre en évidence l'efficacité du vol de travail sur plate-forme hétérogène. La plate-forme n'étant pas naturellement hétérogène, elle a été artificiellement perturbée, la figure 4.8 donne les résultats du gain en temps par rapport à une exécution séquentielle en fonction du nombre de processeurs lorsque ces processeurs ont la même fréquence ; comme prévu dans ce cas, le partitionnement statique donne le meilleur résultat. La figure 4.9 montre, quand à elle, les gains relatifs lorsque les 3 processeurs sont utilisés en

[2]Itanium IA64 (NUMA), 4 bi-coeurs
[3]Traviata, stm8010, 3 CPUs st231, 64MO de RAM

fonction de la perturbation qui est apportée. La perturbation α est telle que la vitesse moyenne Π_{moy} sur l'ensemble des processeurs est constante : le premier processeur a une vitesse Π_{moy}, le second est plus lent $\frac{10}{10+\alpha}\Pi_{moy}$ et le troisième est plus rapide $\frac{10}{10-\alpha}\Pi_{moy}$.

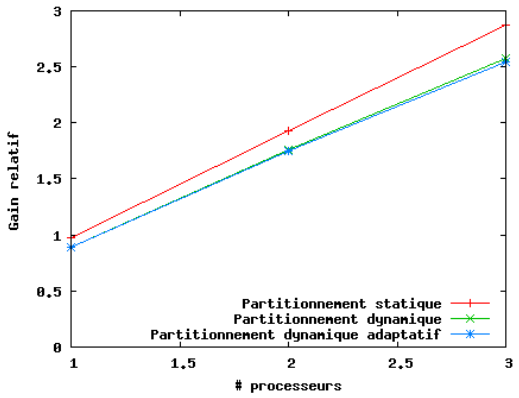

FIG. 4.8 – Gain en temps d'exécution sur plate-forme homogène

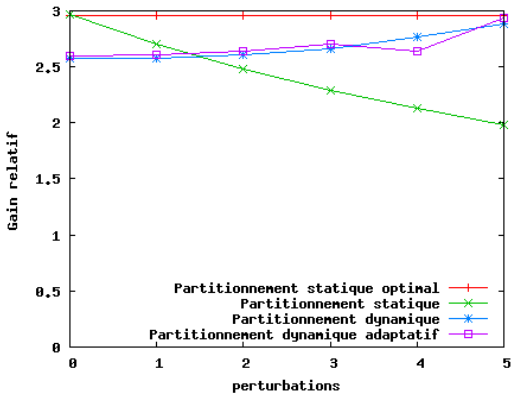

FIG. 4.9 – Gain en temps d'exécution sur plate-forme hétérogène

Preuve de Résistance à la cryptanalyse par différentielles impossibles 5

Sommaire

5.1 Préliminaires 65
5.1.1 Sécurité prouvée face aux attaques différentielles 66
5.1.2 Cryptanalyse par différentielle impossible 67
5.1.2.1 Mesures de résistance vs. Différentielle Impossible 67
5.2 Vers la sécurité prouvée face aux attaques par différentielles impossibles . 68
5.2.1 Matériel et Notations 68
5.2.1.1 Quelques définitions et notations 68
5.2.1.2 Agrégation forte de chaînes de Markov 70
5.2.1.3 Chiffre de Markov en Support 71
5.2.2 Bornes supérieures et inférieures de EDP 72
5.2.2.1 Théorèmes principaux 72
5.2.2.2 Évaluation 76
5.2.3 Résultats pour AES et CS-Cipher 77
5.2.3.1 CS-Cipher 77
5.2.3.2 AES 78
5.2.3.3 Conclusion et améliorations 79

Ce chapitre présente une nouvelle façon d'approcher (ou plutôt de borner) la *probabilité différentielle espérée* (EDP) d'un algorithme de chiffrement ayant une structure de type *SPN*. Cette approche permettra, sous des conditions non triviales, d'encadrer par une borne inférieure et une borne supérieure la valeur de EDP pour un chiffre donné. Nous en déduirons une preuve de résistance aux attaques dites par différentielles impossibles. Les notions de cryptographie utilisées dans ce chapitre ont été introduites dans le chapitre 3 (sections 3.1.2, 3.1.3.2, 3.1.3.3).

5.1 Préliminaires

Nous avons abordé dans la section 3.1.3.2 les notions essentielles de la cryptanalyse différentielle, une méthode générique pour monter une attaque sur un chiffre symétrique à l'aide de ces

notions et les idées utilisées pour prouver (sous conditions) la résistance d'un algorithme de chiffrement face à de telles attaques. Revenons rapidement sur ces dernières.

5.1.1 Sécurité prouvée face aux attaques différentielles

Les attaques par cryptanalyse différentielle sont possibles lorsque l'attaquant est capable de trouver un couple de *différences* (α, β) tel que la probabilité différentielle correspondante $\mathrm{DP}(\alpha, \beta)$ sur l'ensemble ou une partie de l'algorithme de chiffrement est "grande". Elle doit être grande par rapport à un tirage aléatoire, en effet, pour un algorithme de chiffrement parfait, de taille de bloc n, $\forall(\alpha, \beta), \mathrm{DP}(\alpha, \beta) = 2^{-n}$.

Remarque 14 *Nous retrouvons ici une notion fondamentale de la cryptographie. Un* distingueur *est une métrique permettant de distinguer la fonction cible d'une fonction parfaitement aléatoire. Sur la base d'un distingueur, une attaque peut-être montée. Notons qu'ainsi, dans le cadre des attaques différentielles, nous retrouvons une notion très similaire de* sécurité parfaite *qui est abordée dans la section 3.1.1.1 dans sa forme la plus générale définie par Shannon. Bien entendu, la* sécurité parfaite *au sens de Shannon implique cette "sécurité parfaite" pour la cryptanalyse différentielle :*

$$\Big(\Pr_X(X = x|Y = y) = \Pr_X(X = x)\Big) \Rightarrow \Big(\Pr_{X,X'}(X \oplus X' = \alpha | Y \oplus Y = \beta) = \Pr_{X,X'}(X \oplus X' = \alpha)\Big)$$

Avec $Y = E(X)$ et $Y' = E(X')$ pour un algorithme de chiffrement E.

Ainsi, prouver la résistance d'un algorithme de chiffrement face à une attaque par cryptanalyse différentielle correspond à l'évaluation de MEDP, la *probabilité différentielle espérée maximum* pour tout couple de différences. Kanda et al. [111] ont classifié 4 niveaux de sécurité prouvée pour les attaques par cryptanalyse différentielle et linéaire :

Mesure Précise : Trouver la valeur exacte MEDP et ainsi montrer que MEDP est trop proche de 2^{-n} pour qu'une attaque soit possible.

Mesure théorique : Trouver une borne supérieure M de MEDP, elle-même assez proche de 2^{-n}

Mesure Heuristique : Trouver la valeur exacte de la *probabilité caractéristique différentielle espérée maximum* (MEDCP). Cette mesure est justifiée par le fait que lorsqu'un attaquant veut monter une attaque différentielle, il aura besoin de construire une caractéristique différentielle avec une grande probabilité d'occurrence.

Mesure Pratique Trouver une borne supérieure de MEDCP.

Les résultats concernant les deux dernières mesures sont les plus nombreux, du moins pour les chiffres qui ont un design adapté à ce type de preuves (par exemple [219]). Nous sommes plus intéressés dans ce chapitre par les deux premières mesures.
Les résultats récents en la matière se regroupent autour des travaux de Hong et al. [97], qui donnent une borne supérieure à MEDP pour deux rondes d'un chiffrement de type SPN avec une fonction de diffusion parfaite ou presque parfaite. Ce résultat a été généralisé par Kang et al. [112] à toutes fonctions de diffusion. Park et al.[165, 166] proposent deux méthodes pour borner cette même MEDP sur deux rondes et en déduire une borne sur 4 rondes. Keliher et al.[117, 116, 114, 115]

proposèrent plusieurs algorithmes (en particulier KMT2-DC pour la cryptanalyse différentielle) pour borner MEDP pour les chiffrements de type SPN, pour n'importe quel nombre de rondes. Enfin, en 2005, Keliher et Sui [118] proposent le calcul exacte de MEDP pour l'AES sur deux rondes (à savoir MEDP $= 53/2^{34}$), permettant ainsi de trouver de meilleurs bornes supérieures pour 4 rondes et sur l'ensemble de l'AES ($\approx 1.881 \times 2^{-114}$).

5.1.2 Cryptanalyse par différentielle impossible

La cryptanalyse par différentielle impossible, présentée très brièvement dans la section 3.1.3.2, inventée par Biham et al. [27] en 1999, repose sur l'existence de couples de différences dont la probabilité différentielle sur une partie de l'algorithme de chiffrement est nulle (ou, par extension, est très faible par rapport la probabilité uniforme). Ainsi, prouver la résistance à ces attaques correspond à évaluer la *probabilité différentielle espérée minimum* (notée mEDP dans ce document).

5.1.2.1 Mesures de résistance vs. Différentielle Impossible

Les mesures de sécurité présentées dans la section précédente peuvent être ré-interprétées dans le contexte de la cryptanalyse par différentielles impossibles :

Mesure Précise : Trouver la valeur exacte mEDP et ainsi montrer que mEDP est trop proche de 2^{-n} pour qu'une attaque soit possible.

Mesure théorique : Trouver une borne inférieure m de mEDP, elle-même assez proche de 2^{-n}.

Mesure Heuristique : Cette mesure n'aurait pas de raison d'être, en pratique il est toujours possible de construire une caractéristique impossible, i.e. mEDCP $= 0$.

Mesure Pratique cf. Mesure Heuristique.

Les deux dernières mesures ne sont pas pertinentes dans le cas de la cryptanalyse différentielle, c'est peut-être une des raisons pour lesquelles les preuves de résistance aux attaques par différentielles impossibles sont très rares, et en générales basées sur des hypothèses très contraignantes. Notons par exemple les travaux de Sugita et al. [209], qui proposent une méthode de preuve de résistance face aux attaques par différentielles tronquées et impossibles. Ce résultat repose sur la notion d'algorithmes de chiffrement à *sortie différentielles aléatoires* (*Random Output-Differential Ciphers*, notion définie dans l'article). Sous ces hypothèses théoriques, clairement non vérifiées en pratique, les auteurs peuvent évaluer mEDP et ainsi affirmer la résistance face aux attaques par différentielles impossibles.

Remarque 15 *Les solutions mentionnées dans la section précédente pour approcher ou borner MEDP ne sont pas utilisables (telles quelles) pour borner mEDP. En effet, ces techniques utilisent l'évaluation de MEDP sur un faible nombre de rondes (en pratique 2 rondes) pour obtenir une borne sur plusieurs rondes. Soit de façon triviale (une borne supérieure de MEDP sur 2 rondes est aussi une borne supérieure pour tout nombre de rondes ≥ 2), ou de façon plus complexe et ainsi obtenir des bornes plus fines [166, 115]. Or mEDP sur deux rondes vaut trivialement* 0 *pour l'ensemble des chiffres que nous connaissons, cette donnée est donc inutilisable.*

Partant de l'idée développée dans [209], nous proposons une méthode pour minorer mEDP sur un nombre arbitraire de rondes en utilisant un nouveau modèle : les *Chiffres de Markov en Support*. Ce modèle, contrairement à celui des *Chiffres de Markov* présenté dans la section 3.1.3.3, n'est pas vérifié pour les algorithmes de chiffrements connus sous la simple hypothèse que les clés de rondes sont indépendantes et uniformément distribuées. Pourtant, il ouvre une nouvelle voie de recherche vers la sécurité prouvée face aux attaques par différentielles impossibles et tisse un nouveau lien entre la théorie des chaînes de Markov et l'étude théorique des algorithmes de chiffrement par blocs.

5.2 Vers la sécurité prouvée face aux attaques par différentielles impossibles

Nous présenterons d'abord le matériel et les notations nécessaires pour cette analyse, puis les preuves et enfin les résultats pour deux chiffres symétriques : CS-Cipher et AES.

5.2.1 Matériel et Notations

5.2.1.1 Quelques définitions et notations

L'étude faite dans ce chapitre est dédiée aux algorithmes de chiffrement qui ont une structure SPN ou PSN (voir la section 3.1.2.1). Il nous faut dans un premier temps introduire quelques définitions et notations pour manipuler simplement ces chiffrements.

Notations Tout au long de ce chapitre, l'algorithme de chiffrement sera noté $\mathbf{E}$, sa taille de bloc sera notée $\mathbf{n}$, et son nombre de rondes noté $\mathbf{R}$.

$\oplus$ dénote le *ou exclusif bit à bit*.

$\mathbf{wt}()$ dénote le poids de Hamming d'un vecteur d'éléments dans $GF(2)$

$\mathbf{\Delta(x)}$ dénote la différence par l'opération $\oplus$ entre deux vecteurs de bits x et x' lorsqu'il n'y pas d'ambiguïté sur x et x' , i.e. $x \oplus x' = \Delta(x)$.

$\mathbf{k_i}$ dénote la $i$ème clé de ronde.

$\mathbf{O_i(x)}$ dénote la sortie de la $i$ème ronde de l'algorithme de chiffrement $\mathbf{E}$ lorsque x est le message clair.

i-ronde Différentielle (α, β) dénote un couple de différences tel qu'il existe un couple de messages (x, x') pour lequel $\Delta(x) = \alpha$ et $\Delta(O_i(x)) = \beta$. Une R-ronde différentielle est simplement appelée une *Différentielle*.

i-ronde Caractéristique Ω_i dénote une *caractéristique différentielle* sur les i premières rondes de E.

Restrictions sur l'algorithme de chiffrement Afin de simplifier les preuves, nous allons fixer quelques propriétés particulières de l'algorithme de chiffrement cible E. Notons que pour un chiffre n'ayant pas ces propriétés, il est possible d'adapter les preuves pour son cas, en d'autres termes ces restrictions ne compromettent pas la généralité de la méthode.

Nous nous restreindrons à une forme plus simple appelée *chiffre clé-alternant* ("Key-alternating cipher" en anglais) et introduite par Joan Daemen et Vincent Rijmen dans de récents travaux [63]. Le schéma général de l'algorithme de chiffrement d'un tel chiffre clé-alternant est présenté sur la figure 5.1, il s'agit simplement de restreindre la fonction de mélange de clé à un ou exclusif bit à bit. C'est le cas, entre autres, de l'AES et de CS-Cipher.

Remarque 16 *les rondes du schéma 5.1 sont uniquement composées des fonctions de substitutions et de permutations.*

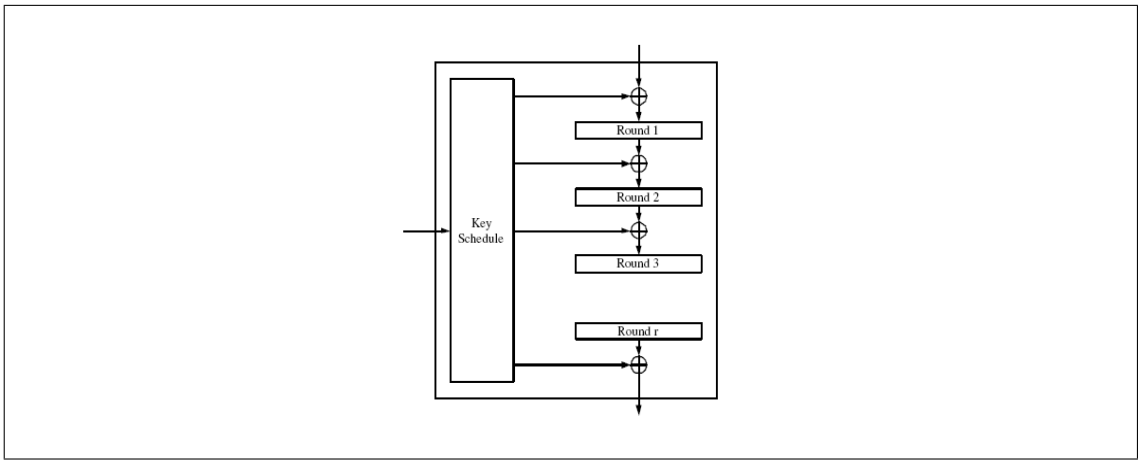

FIG. 5.1 – Schéma général d'un algorithme de chiffrement clé-alternant [63]

Les rondes de l'algorithme de chiffrement sont identiques et la fonction de génération de clé est supposée être optimale dans le sens où chaque clé de ronde est indépendante des autres et uniformément distribuée. Ces hypothèses sur les clés de rondes sont irréalisables mais souvent acceptées pour construire des preuves de résistance (voir dernières remarques de la section 3.1.3.3).

Remarque 17 ([63]) *Sous ces conditions sur les clés de rondes, un chiffre clé-alternant est un chiffre de Markov.*

La zone de confusion de l'algorithme de chiffrement est composé de $\boldsymbol{m}$ boîtes-S identiques. La boîte de substitution, notée $\mathbf{P}$, est une permutation de $\boldsymbol{l}$ bits, i.e., permutation de $GF(2)^l$ dans lui-même ($\boldsymbol{l \times m = n}$). Ainsi nous noterons $\mathbf{P^m}$ la permutation sur $(GF(2)^l)^m$ composée de la concaténation des m boîtes-S.

Notons DP_{max} la *probabilité différentielle maximale* de la boîte-S et DP_{min} la *probabilité différentielle minimale* de la boîte-S, tels que :

$$\begin{aligned} \mathrm{DP}_{max} &= \max_{\alpha \neq 0, \beta} \mathrm{DP}^P(\alpha, \beta) \\ \mathrm{DP}_{min} &= \min_{\substack{\alpha \neq 0, \beta \\ \mathrm{DP}^P(\alpha,\beta) \neq 0}} \mathrm{DP}^P(\alpha, \beta)) \end{aligned}$$

La zone de diffusion est constituée d'une application linéaire, notée $\mathcal{L}$, de $GF(2^l)^m$ dans lui-même.
Le schéma 5.2 (resp. 5.3) récapitule la structure d'une ronde de l'algorithme de chiffrement par bloc de type SPN (resp. PSN).

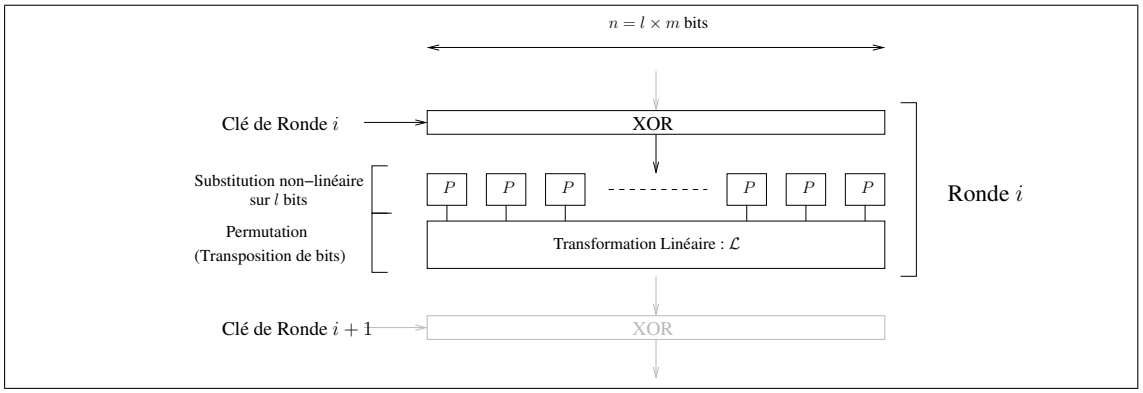

FIG. 5.2 – Structure SPN.

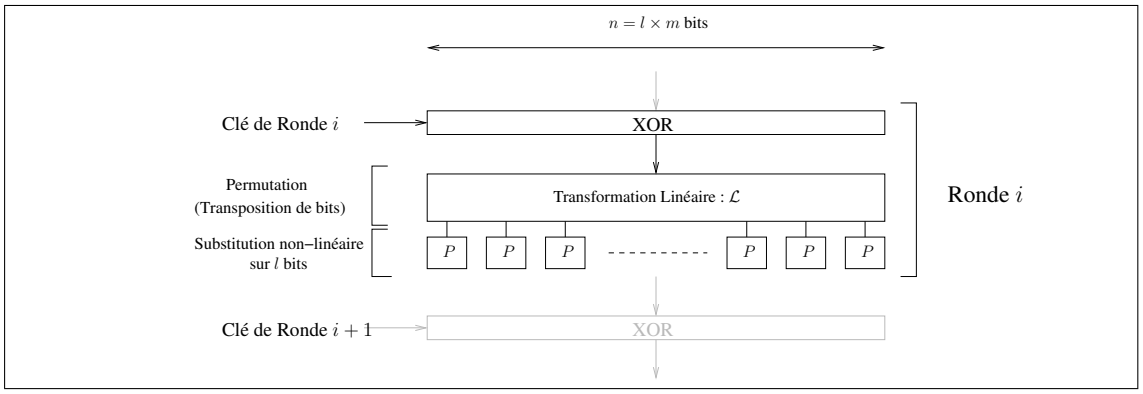

FIG. 5.3 – Structure PSN.

5.2.1.2 Agrégation forte de chaînes de Markov

Définition 19 (Chaînes de Markov Fortement Lumpables) *Soit une chaîne de Markov* $(X_0, X_1, \cdots)$ *définie sur un espace d'état* S *de taille* n $(S = (s_1, \cdots, s_n))$*, soit* $(S_1, \cdots, S_k)$ *une*

partition de S en k ensembles disjoints.
Soit le processus aléatoire défini par les variables aléatoires $(\hat{X}_0, \hat{X}_1, \cdots)$ sur l'espace d'état $(S_1, \cdots, S_k)$ tel que

$$\forall 1 \leq j, l \leq k, \forall i \geq 0, \Pr(\hat{X}_{i+1} = S_j | \hat{X}_i = S_l) = \Pr(X_{i+1} \in S_j | X_i \in S_l)$$

La chaîne de Markov $(X_0, X_1, \cdots)$ est dite fortement agrégeable *(strong lumpability en anglais) par rapport à la partition $(S_1, \cdots, S_k)$, si et seulement si, le processus aléatoire $(\hat{X}_0, \hat{X}_1, \cdots)$ est une chaîne de Markov, i.e.*

$$\Pr(X_{i+1} \in S_{r_{i+1}} | X_i \in S_{r_i}, X_{i-1} \in S_{r_{i-1}}, \cdots, X_0 \in S_{r_0}) = \Pr(X_{i+1} \in S_{r_{i+1}} | X_i \in S_{r_i})$$

Le théorème 4 donne une idée plus claire des contraintes que l'agrégation forte impose à une chaîne de Markov. Notons $p_{iS_j} = \sum_{s_k \in S_j} \Pr(X_{l+1} = s_k | X_l = s_i)$, la probabilité de passer d'un état s_i dans le sous-ensemble S_j en un pas de la chaîne de Markov originale.

Théorème 4 (provennant de [120]) *Une condition nécessaire et suffisante pour qu'une chaîne de Markov soit fortement agrégeable par rapport à la partition $(S_1, \cdots, S_k)$ est que pour tout couple S_i et S_j, p_{kS_j} ait la même valeur pour tout s_k dans S_i. Ces valeurs $\{\hat{p}_{ij}\}$ forment la matrice de transition de la chaîne agrégée.*

La condition de chaîne de Markov fortement agrégeable est très contraignante. Peu de chaînes de Markov utiles en pratique possèdent une partition intéressante de leur espace d'état satisfaisant cette propriété. Par ailleurs, trouver une partition de l'espace d'état d'une chaîne de Markov permettant une agrégation forte est un problème réputé difficile.
Néanmoins c'est un outil d'une très grande puissance car il permet de restreindre l'étude de la chaîne de Markov à chaque ensemble de la partition pris séparément (c'est l'*agrégation*) puis, de faire remonter le résultat sur l'ensemble de la chaîne (*désagrégation*), ainsi réduisant considérablement la complexité de l'étude sur un espace d'états trop grand.
Notons qu'il existe des méthodes d'approximations lorsque la chaîne n'est que partiellement agrégeable (ou encore appelée *presque décomposable*) [120, 41, 206].

L'intérêt d'apporter cette notion aux chiffres de Markov est évidente : les chaînes de Markov de ces chiffres ont un espace d'état gigantesque (2^n pour une taille de bloc de n), pouvoir les agréger permettrait de réduire la complexité de leur étude.

5.2.1.3 Chiffre de Markov en Support

Nous définissons ici un *chiffre de Markov en Support*.

Définition 20 (Fonction Support χ) *Appelée fonction caractéristique dans [209], χ est définie comme suit :*

$$\begin{array}{rccc} \chi: & GF(2^l)^m & \rightarrow & GF(2)^m \\ & (x_1, \ldots, x_m) & \rightarrow & (y_1, \ldots, y_m) \end{array}$$

tel que

$$y_i = \begin{cases} 0 \textit{ si le l\_uplet } x_i = 0, \\ 1 \textit{ sinon.} \end{cases}$$

Remarque 18 *$\chi(x)$, est souvent noté γ_x dans la littérature (entre autres [114, 166]). Nous adopterons donc cette notation par la suite.*

Remarque 19 *$\chi(O_i(x) \oplus O_i(x')) = \gamma_{\Delta(O_i(x))}$ représente directement le paterne des boîtes-S actives pour la $i$ème ronde (resp. pour la $(i-1)$ème ronde) d'une structure SPN (resp. d'une structure PSN).*
Nous avons donc $\gamma_{\Delta(x)} = \gamma_{P^m(\Delta(x))}$, i.e. il y a une différence non nulle en sortie de boîte-S si et seulement si il y a une différence non nulle en entrée.

Définition 21 (Chiffre de Markov en Support) *Un chiffre de Markov E est dit* Chiffre de Markov en Support *si et seulement si la chaîne de Markov qui lui est associée est fortement agrégeable par rapport au partitionnement de son espace d'état par la fonction support χ.*

Ce partitionnement de l'espace d'état $GF(2^l)^m$ par la fonction support χ est naturellement défini par les 2^m sous espaces $(S_0, \cdots, S_{2^m-1})$ tels que, $\forall x \in GF(2^l)^m, x \in S_{\gamma_x}$.

5.2.2 Bornes supérieures et inférieures de EDP

Dans cette section sont présentés les théorèmes principaux de ce chapitre. Pour les deux types de structure (SPN et PSN), deux théorèmes sont proposés, l'un pour le calcul de la borne supérieure de EDP, l'autre pour le calcul de la borne inférieure.

5.2.2.1 Théorèmes principaux

Lemme 4 *Soit un couple de messages clairs (x, x') tel que $(\Delta(x) = \Delta y_0, \Delta y_1, \cdots, \Delta y_R)$ soit la caractéristique différentielle produite par (x, x') sur l'ensemble de l'algorithme de chiffrement E.*
Pour tout $i \in \{0, \cdots, R-1\}$,

$$\begin{array}{lcll} \gamma_{\mathcal{L}^{-1}(\Delta y_{i+1})} & = & \gamma_{\Delta y_i} & \textit{Pour une structure SPN} \\ \gamma_{\Delta y_{i+1}} & = & \gamma_{\mathcal{L}(\Delta y_i)} & \textit{Pour une structure PSN} \end{array}$$

Preuve Nous présentons le cas SPN, le cas PSN se traite de la même façon.
Pour tout $i \in \{1, \cdots, R\}$, notons $y_i = O_i(x)$ et $y'_i = O_i(x')$. Soit $i \in \{0, \cdots, R-1\}$, $\mathcal{L}^{-1}(\Delta y_{i+1}) = \mathcal{L}^{-1}(y_{i+1}) \oplus \mathcal{L}^{-1}(y'_{i+1})$ puisque $\mathcal{L}$ est linéaire. Donc $\mathcal{L}^{-1}(\Delta y_{i+1})$ est strictement égale à la différence des données en sortie des boîtes-S de la $i$ème ronde (notée Δa_i). Or $\Delta a_i = P^m(k_i \oplus y_i) \oplus P^m(k_i \oplus y'_i)$ donc $\gamma_{\Delta a_i} = \gamma_{\Delta y_i}$ puisque la transformation par la fonction support χ est conservée par le passage à travers les boîtes-S et que trivialement $k_i \oplus y_i \oplus k_i \oplus y'_i = \Delta y_i$. □

Lemme 5 *Soit $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov du chiffre de Markov E, nous avons pour tout $i \in \{0, \cdots, R-1\}$,*
pour une structure SPN :

$$\Pr(X_{i+1} = x_{i+1} | X_i = x_i) \begin{cases} \leq (DP_{max})^{wt(\gamma_{x_i})} & \text{si } \gamma_{x_i} = \gamma_{\mathcal{L}^{-1}(x_{i+1})} \\ = 0 & \text{sinon} \end{cases}$$

pour une structure PSN :

$$\Pr(X_{i+1} = x_{i+1} | X_i = x_i) \begin{cases} \leq (DP_{max})^{wt(\gamma_{x_{i+1}})} & \text{si } \gamma_{x_{i+1}} = \gamma_{\mathcal{L}(x_i)} \\ = 0 & \text{sinon} \end{cases}$$

Preuve Nous présentons la preuve uniquement dans le cas SPN, le cas PSN étant similaire.
Le lemme 4 montre que si $\gamma_{x_i} \neq \gamma_{\mathcal{L}^{-1}(x_{i+1})}$ alors $\Pr(X_{i+1} = x_{i+1} | X_i = x_i) = 0$
Maintenant, si $\gamma_{x_i} = \gamma_{\mathcal{L}^{-1}(x_{i+1})}$, considérons la différence en sortie de boîtes-S, caractérisée par la variable aléatoire A. Comme X_{i+1} est complètement connu par la valeur prise par A ($\mathcal{L}(A) = X_{i+1}$),

$$\Pr(X_{i+1} = x_{i+1} | X_i = x_i) = \Pr(A = \mathcal{L}^{-1}(x_{i+1}) | X_i = x_i)$$

Ainsi, puisque les m boîtes-S reçoivent des entrées indépendantes les unes des autres, nous avons le résultat. □

Lemme 6 *Soit $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov du chiffre de Markov E, nous avons pour tout $i \in \{0, \cdots, R-1\}$,*
pour une structure SPN :

$$\textit{Si } \Pr(X_{i+1} = x_{i+1} | X_i = x_i) \neq 0 \textit{ alors } \Pr(X_{i+1} = x_{i+1} | X_i = x_i) \geq (DP_{min})^{wt(\gamma_{x_i})}$$

pour une structure PSN :

$$\textit{Si } \Pr(X_{i+1} = x_{i+1} | X_i = x_i) \neq 0 \textit{ alors } \Pr(X_{i+1} = x_{i+1} | X_i = x_i) \geq (DP_{min})^{wt(\gamma_{x_{i+1}})}$$

Preuve Similaire à la preuve du lemme 5 □

Le théorème 5 (resp. théorème 6), pour les structures SPN, sera appliqué au chiffre AES dans la section suivante afin de trouver une borne supérieure (resp. inférieure) à la probabilité d'occurrence d'une i-ronde différentielle.

Théorème 5 (Borne Supérieure pour Structure SPN) *Soit E un algorithme de chiffrement de Markov et $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov associée. Pour tout $i \in \{2, \cdots, R\}$,*

$$\begin{aligned} \Pr(X_i = x_i | X_0 = x_0) \quad \leq \quad & [DP_{max} \times (2^l - 1)]^{wt(\gamma_{x_0})} \\ & \times \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}, \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0}) \\ & \times \Pr(\chi(X_i) = \gamma_{x_i} | \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0}) \end{aligned}$$

Preuve Commençons par introduire X_1 dans la formule via la loi des probabilités totales :

$$\Pr(X_i = x_i | X_0 = x_0) = \sum_{x_1} \Pr(X_i = x_i | X_1 = x_1, X_0 = x_0) \Pr(X_1 = x_1 | X_0 = x_0)$$

Le lemme 5 nous donne :

$$\Pr(X_i = x_i | X_0 = x_0) \leq (\mathrm{DP}_{max})^{wt(\gamma_{x_0})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}^{-1}(x_1)} = \gamma_{x_0}}} \Pr(X_i = x_i | X_1 = x_1, X_0 = x_0)$$

Et comme $(X_0, X_1, \cdots, X_R)$ est une chaîne de Markov,

$$\Pr(X_i = x_i | X_0 = x_0) \leq (\mathrm{DP}_{max})^{wt(\gamma_{x_0})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}^{-1}(x_1)} = \gamma_{x_0}}} \Pr(X_i = x_i | X_1 = x_1)$$

Qui peut se ré-écrire sous la forme :

$$\Pr(X_i = x_i | X_0 = x_0) \leq (\mathrm{DP}_{max})^{wt(\gamma_{x_0})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}^{-1}(x_1)} = \gamma_{x_0}}} \frac{\Pr(X_i = x_i, X_1 = x_1)}{\Pr(X_1 = x_1)}$$

Or $\Pr(X_1 = x_1)$ est une constante pour tout x_1 et vaut $1/2^n$:

$$\Pr(X_i = x_i | X_0 = x_0) \leq 2^n (\mathrm{DP}_{max})^{wt(\gamma_{x_0})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}^{-1}(x_1)} = \gamma_{x_0}}} \Pr(X_i = x_i, X_1 = x_1)$$

En sommant, nous obtenons :

$$\Pr(X_i = x_i | X_0 = x_0) \leq 2^n (\mathrm{DP}_{max})^{wt(\gamma_{x_0})} \Pr(X_i = x_i, \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0})$$

Ainsi, en développant le dernier terme et en remarquant que
$\Pr(\chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0}) = \frac{(2^l - 1)^{wt(\gamma_{x_0})}}{2^n}$,

$$\Pr(X_i = x_i | X_0 = x_0) \leq [\mathrm{DP}_{max} \times (2^l - 1)]^{wt(\gamma_{x_0})} \Pr(X_i = x_i | \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0})$$

Enfin, en introduisant le terme $\chi(X_i)$ via la loi des probabilités totale,

$$\begin{aligned}\Pr(X_i = x_i | X_0 = x_0) \leq\ & [\mathrm{DP}_{max} \times (2^l - 1)]^{wt(\gamma_{x_0})} \times \\ & \sum_{\gamma} \Pr(X_i = x_i | \chi(X_i) = \gamma, \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0}) \Pr(\chi(X_i) = \gamma | \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0})\end{aligned}$$

Il suffit de remarquer que $\Pr(X_i = x_i | \chi(X_i) = \gamma, \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0}) \neq 0$ uniquement si $\gamma = \gamma_{x_i}$ pour obtenir le résultat cherché. □

Théorème 6 (Borne Inférieure pour Structure SPN) *Soit E un algorithme de chiffrement de Markov et $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov associée. Pour tout $i \in \{2, \cdots, R\}$,*

$$\begin{aligned}\Pr(X_i = x_i | X_0 = x_0) \quad \geq \quad & [(DP_{min})^2 \times (2^l - 1)]^{wt(\gamma_{x_0})} \\ & \times \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}, \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0}) \\ & \times \Pr(\chi(X_i) = \gamma_{x_i} | \chi(\mathcal{L}^{-1}(X_1)) = \gamma_{x_0})\end{aligned}$$

Preuve Commençons par introduire X_1 dans la formule via la loi des probabilités totales :

$$\Pr(X_i = x_i | X_0 = x_0) = \sum_{x_1} \Pr(X_i = x_i | X_1 = x_1, X_0 = x_0) \Pr(X_1 = x_1 | X_0 = x_0)$$

Le lemme 6 nous donne :

$$\Pr(X_i = x_i | X_0 = x_0) \geq (\text{DP}_{min})^{wt(\gamma_{x_0})} \sum_{\substack{x_1 t.q. \\ \Pr(X_1 = x_1 | X_0 = x_0) \neq 0}} \Pr(X_i = x_i | X_1 = x_1)$$

Notons $\begin{cases} Poss_{x_0} & = & \{x, \Pr(X_1 = x | X_0 = x_0) \neq 0\} \\ Supp_{x_0} & = & \{x, \gamma_{\mathcal{L}^{-1}(x)} = \gamma_{x_0}\} \end{cases}$
Nous allons évaluer $\sum_{x_1 \in Poss_{x_0}} \Pr(X_i = x_i | X_1 = x_1)$ en remarquant tout d'abord que $Poss_{x_0} \subset Supp_{x_0}$ (d'après le lemme 4).
De plus le cardinal de $Supp_{x_0}$ est $(2^l - 1)^{wt(\gamma_{x_0})}$ et le cardinal de $Poss_{x_0}$ vaut, au moins, $(\text{DP}_{min}(2^l - 1))^{wt(\gamma_{x_0})}$, nous avons donc :

$$Card(\{Poss_{x_0}\}) \geq (\text{DP}_{min})^{wt(\gamma_{x_0})} Card(\{Supp_{x_0}\})$$

Enfin, considérant $(X_0, X_1, \cdots, X_R)$ comme une chaîne de Markov :
$\forall (x, y), \Pr(X_i = y | X_1 = x)$ est indépendant du fait que $x \in Poss_{x_0}$ ou $x \in Supp_{x_0}$, nous obtenons donc :

$$\Pr(X_i = x_i | X_0 = x_0) \geq (\text{DP}^2_{min})^{wt(\gamma_{x_0})} \sum_{x_1 \in Supp_{x_0}} \Pr(X_i = x_i | X_1 = x_1)$$

La fin de la preuve est strictement similaire à la preuve du théorème 5 □

Les théorèmes 7 et 8 sont le pendant des théorèmes 5 et 6 pour les structures PSN, les preuves sont très proches. Le lecteur soucieux des détails pourra les trouver en annexe A de ce document. Ils seront utilisés dans la section suivante pour le chiffre CS-Cipher.

Théorème 7 (Borne Supérieure pour Structure PSN) *Soit E un algorithme de chiffrement de Markov et $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov associée. Pour tout $i \in \{2, \cdots, R\}$,*

$$\begin{aligned} \Pr(X_i = x_i | X_0 = x_0) \quad \leq \quad & [DP_{max} \times (2^l - 1)]^{wt(\gamma_{\mathcal{L}(x_0)})} \\ & \times \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}, \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \\ & \times \Pr(\chi(X_i) = \gamma_{x_i} | \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \end{aligned}$$

Théorème 8 (Borne Inférieure pour Structure PSN) *Soit E un algorithme de chiffrement de Markov et $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov associée. Pour tout $i \in \{2, \cdots, R\}$,*

$$\begin{aligned} \Pr(X_i = x_i | X_0 = x_0) \quad \geq \quad & [(DP_{min})^2 \times (2^l - 1)]^{wt(\gamma_{\mathcal{L}(x_0)})} \\ & \times \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}, \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \\ & \times \Pr(\chi(X_i) = \gamma_{x_i} | \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \end{aligned}$$

5.2.2.2 Évaluation

Notons que les théorèmes présentés dans la section précédente ne supposent pas l'hypothèse de chiffre de Markov en Support. Ils peuvent être vus comme une agrégation d'un chiffre de Markov : nous exprimons les bornes supérieures et inférieures recherchées en fonction de l'état transitoire de la chaîne de Markov agrégée.

L'hypothèse de chiffre de Markov fortement agrégeable va nous permettre de calculer exactement les bornes en nous permettant d'approcher les deux derniers termes des équations :
$\Pr(X_i = x_i|\chi(X_i) = \gamma_{x_i}, \chi(X_0)) = \gamma_{x_0})$ et $\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0})$.

$\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0})$**.** L'hypothèse de chiffre de Markov en support nous dit que $(\chi(X_0), \chi(X_1), \cdots, \chi(X_R))$ est une chaîne de Markov, nous en déduisons donc que

$$\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0}) = \sum_{\gamma_{x_1}, \cdots, \gamma_{x_{R-1}}} \prod_{k=0}^{i-1} \Pr(\chi(X_{k+1}) = \gamma_{x_{k+1}}|\chi(X_k) = \gamma_{x_k})$$

En fait, en considérant la matrice de transition de la chaîne de Markov $(\chi(X_0), \cdots, \chi(X_R))$, de taille $(2^m)^2$, le calcul de $\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0})$ corresponds à R produit matrices-vecteurs (de complexité $O((2^m)^2)$). Par ce calcul on obtient la répartition probabiliste de sortie : $\forall\gamma, \Pr(\chi(X_i) = \gamma|\chi(X_0) = \gamma_{x_0})$. Le faire pour tout état initial γ_{x_0} se fait donc avec une complexité $O(R2^{3m})$.

Remarque 20 *La complexité est en pratique moindre et est fonction de la fonction de diffusion $\mathcal{L}$: pour un élément $\gamma \in GF(2)^m$, le cardinal de l'ensemble $\{\gamma', \exists x \in GF(2)^n, (\gamma_x = \gamma')$ et $\gamma_{\mathcal{L}(x)} = \gamma\}$ peut-être très restreint, nous appellerons $RULES(\gamma)$ ce cardinal dans la suite.*

Remarque 21 *La* sécurité parfaite*, cette fois-ci au sens du support différentiel, peut être définie comme suit :*

$$\Pr_{X,X'}(\chi(X \oplus X') = \gamma_x|\chi(Y \oplus Y') = \gamma_y) = \Pr_{X,X'}(\chi(X \oplus X') = \gamma_x)$$

Où $Y = E(X)$, $Y' = E(X')$, X et X' ne représentent plus des différences.
Ainsi, en reprenant la remarque 14, nous avons une hiérarchie dans les notions de sécurité parfaites :

$$\Big(\Pr_X(X = x|Y = y) = \Pr_X(X = x)\Big)$$
$$\Rightarrow \Big(\Pr_{X,X'}(X \oplus X' = \alpha|Y \oplus Y = \beta) = \Pr_{X,X'}(X \oplus X' = \alpha)\Big)$$
$$\Rightarrow \Big(\Pr_{X,X'}(\chi(X \oplus X') = \gamma_x|\chi(Y \oplus Y') = \gamma_y) = \Pr_{X,X'}(\chi(X \oplus X') = \gamma_x)\Big)$$

$\Pr(X_i = x_i|\chi(X_i) = \gamma_{x_i}, \chi(X_0)) = \gamma_{x_0})$. Bien que $(X_0, \cdots, X_R)$ et sa forme agrégée $(\chi(X_0), \cdots, \chi(X_R))$ soient toutes deux des chaînes de Markov, il est en générale difficile d'évaluer une telle probabilité. Pourtant nous ferons ici une conjecture qui semble tout à fait acceptable :

Conjecture 1 *L'influence de l'événement* $(\chi(X_0)) = \gamma_{x_0})$ *sur l'événement* $(X_i = x_i|\chi(X_i) = \gamma_{x_i})$ *est plus faible que son influence sur* $(\chi(X_i) = \gamma_{x_i})$

En d'autres termes, la seule donnée du paterne des boîtes-S actives en entrée a plus d'influence sur le paterne des boîtes-S actives après i rondes que sur la valeur des différences de sortie des boîtes-S actives après i rondes. Bien que cette conjecture ne soit pas prouvée, elle semble d'autant plus vérifiée à mesure que l'influence de $(\chi(X_0)) = \gamma_{x_0})$ sur $(\chi(X_i) = \gamma_{x_i})$ est faible.

Sous ces différentes hypothèses, il est maintenant possible d'exprimer une borne supérieure et une borne inférieure pour la probabilité d'occurrence d'une i-ronde différentielle pour un chiffre de type SPN ou PSN. Notamment, si, pour un chiffre donné,
$\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0})$ est strictement positif pour toutes valeurs de γ_{x_i} et γ_{x_0} et que sa valeur est assez proche de $\Pr(\chi(X_i) = \gamma_{x_i})$ (i.e., l'influence de $(\chi(X_0) = \gamma_{x_0})$ est faible) alors pour tout couple (α, β) de différences, $\Pr(X_i = \beta|X_0 = \alpha) > 0$ et donc il n'existe pas de différentielles impossibles sur i rondes et plus.
Nous avons fait ces calculs pour deux chiffres : AES et CS-Cipher, les résultats sont présentés dans la section suivante.

5.2.3 Résultats pour AES et CS-Cipher

5.2.3.1 CS-Cipher

Le chiffre CS-Cipher ainsi que la version étudiée dans ce chapitre CSC^* sont décrits en détail dans le chapitre 7. Rappelons à présent les paramètres le concernant, nécessaires pour notre analyse.

$CSC*$ a été proposé par S. Vaudenay [219] pour l'étude de sécurité de CS-Cipher. C'est un chiffre à clé-alternant, et les clés de ronde sont indépendantes et uniformément distribuées. CSC^* est donc un chiffre de Markov. Les boîtes-S du chiffre CSC^* sont toutes identiques. Ce chiffre est donc un choix pertinent pour l'étude présentée dans les sections précédentes. D'autre part, conformément aux notations de la précédente section :

n	R	m	l	DP_{max}	DP_{min}	$RULE(\gamma)$
64	24	8	8	2^{-4}	2^{-7}	$\leq 3^4 = 81$

Calcul de $\Pr(\chi(\mathbf{X_i})|\chi(\mathbf{X_0}))$ Le calcul direct de $\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0})$ pour toutes valeurs de i, γ_{x_i} et γ_{x_0} nous permet d'assurer (sous l'hypothèse de chiffre de Markov en Support), $\forall i \geq 11, \forall \gamma_{x_i}, \gamma_{x_0}$,

$$\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0}) \begin{cases} \leq \Pr(\chi(X_i) = \gamma_{x_i}) + 2^{-64} \\ \geq \Pr(\chi(X_i) = \gamma_{x_i}) - 2^{-64} \end{cases}$$

Nous en déduisons directement, d'après la conjecture 1,
$\forall i \geq 11, \forall x_i, \gamma_{x_0},$

$$\Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}, \chi(X_0) = \gamma_{x_0}) \begin{cases} \leq \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}) + 2^{-64} \\ \geq \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}) - 2^{-64} \end{cases}$$

Il suffit alors de remarquer que

$$Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}) = (\tfrac{1}{2^l - 1})^{wt(\gamma_{x_i})}$$
$$Pr(\chi(X_i) = \gamma_{x_i}) = (\tfrac{2^l - 1}{2^l})^{wt(\gamma_{x_i})} \times (\tfrac{1}{2^l})^{m - wt(\gamma_{x_i})} = (2^l - 1)^{wt(\gamma_{x_i})} \times 2^{-l \cdot m}$$

et appliquer les théorèmes 7 et 8 pour obtenir :
$\forall i \geq 11, \forall x_i, x_0,$

$$\begin{aligned} \Pr(X_i = x_i | X_0 = x_0) \quad \geq \quad & [2^{-6} - 2^{-14}]^{wt(\gamma_{\mathcal{L}(x_0)})} \\ & \times 2^{-64} [(2^8 - 1)^{wt(\gamma_{x_i})} - 1] \\ & \times [(2^8 - 1)^{-wt(\gamma_{x_i})} - 2^{-64}] \end{aligned}$$

et

$$\begin{aligned} \Pr(X_i = x_i | X_0 = x_0) \quad \leq \quad & [2^4 - 2^{-4}]^{wt(\gamma_{\mathcal{L}(x_0)})} \\ & \times 2^{-64} [(2^8 - 1)^{wt(\gamma_{x_i})} - 1] \\ & \times [(2^8 - 1)^{-wt(\gamma_{x_i})} - 2^{-64}] \end{aligned}$$

Ce qui donne, grossièrement, après 11 rondes de CSC^* :
pour $wt(\gamma_{\mathcal{L}(x_0)}) = 8$

$$2^{-112} \leq \Pr(X_i = x_i | X_0 = x_0) \leq 2^{-32}$$

pour $wt(\gamma_{\mathcal{L}(x_0)}) = 1$

$$2^{-70} \leq \Pr(X_i = x_i | X_0 = x_0) \leq 2^{-60}$$

Ces résultats ont fait l'objet d'une publication dans les proceedings de la conférence MCO ("Modelling, Computation and Optimization in Information Systems and Management Sciences") en 2008 [187]

5.2.3.2 AES

Le chiffre AES [64] est de structure SPN avec toutes ses boîtes-S identiques. Conformément aux notations de la section précédente :

n	R	m	l	DP_{max}	DP_{min}	$RULE(\gamma)$
128	10	16	8	2^{-6}	2^{-7}	1192 (moyenne)

Calcul de $\Pr(\chi(\mathbf{X_i})|\chi(\mathbf{X_0}))$ Le calcul direct de $\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0})$ pour toutes valeurs de i, γ_{x_i} et γ_{x_0} nous permet d'assurer (sous l'hypothèse de chiffre de Markov en Support), $\forall i \geq 6, \forall \gamma_{x_i}, \gamma_{x_0}$,

$$\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0}) \begin{cases} \leq \Pr(\chi(X_i) = \gamma_{x_i}) + 2^{-128} \\ \geq \Pr(\chi(X_i) = \gamma_{x_i}) - 2^{-128} \end{cases}$$

Remarque 22 *Le calcul de* $\Pr(\chi(X_i) = \gamma_{x_i}|\chi(X_0) = \gamma_{x_0})$ *pour toutes valeurs de* i, γ_{x_i} *et* γ_{x_0} *a une complexité qui approche* $O(i \times 1192 \times (2^{16})^2)$*, il n'a pu être fait dans son entier. Nous nous sommes limité à un ensemble plus faible d'entrées (*γ_{x_0}*) testées. Les résultats montrent que les probabilités sont équivalentes lorsque le poids de Hamming de* γ_{x_0} *est fixé, nous en déduisons donc les résultats proposés ici.*

Nous en déduisons directement, d'après la conjecture 1,
$\forall i \geq 6, \forall x_i, \gamma_{x_0}$,

$$\Pr(X_i = x_i|\chi(X_i) = \gamma_{x_i}, \chi(X_0) = \gamma_{x_0}) \begin{cases} \leq \Pr(X_i = x_i|\chi(X_i) = \gamma_{x_i}) + 2^{-128} \\ \geq \Pr(X_i = x_i|\chi(X_i) = \gamma_{x_i}) - 2^{-128} \end{cases}$$

Il suffit alors d'ppliquer les théorèmes 5 et 6 pour obtenir :
$\forall i \geq 6, \forall x_i, x_0$,

$$\begin{aligned} \Pr(X_i = x_i|X_0 = x_0) \quad \geq \quad & [2^{-6} - 2^{-14}]^{wt(\gamma_{x_0})} \\ & \times 2^{-128}[(2^8 - 1)^{wt(\gamma_{x_i})} - 1] \\ & \times [(2^8 - 1)^{-wt(\gamma_{x_i})} - 2^{-128}] \end{aligned}$$

et

$$\begin{aligned} \Pr(X_i = x_i|X_0 = x_0) \quad \leq \quad & [2^2 - 2^{-6}]^{wt(\gamma_{x_0})} \\ & \times 2^{-128}[(2^8 - 1)^{wt(\gamma_{x_i})} - 1] \\ & \times [(2^8 - 1)^{-wt(\gamma_{x_i})} - 2^{-128}] \end{aligned}$$

Ce qui donne, grossièrement, après 6 rondes de l'AES :
pour $wt(\gamma_{x_0}) = 16$

$$2^{-224} \leq \Pr(X_i = x_i|X_0 = x_0) \leq 2^{-96}$$

pour $wt(\gamma_{x_0}) = 1$

$$2^{-134} \leq \Pr(X_i = x_i|X_0 = x_0) \leq 2^{-126}$$

5.2.3.3 Conclusion et améliorations

Le modèle établi pour les chiffres par bloc permet effectivement d'évaluer exactement les bornes inférieures et supérieures de la probabilité différentielle espérée (EDP), notons en effet que toutes les probabilités manipulées dans les sections précédentes sont évaluées en moyenne sur l'ensemble des clés de rondes uniformément distribuées et indépendantes. L'agrégation forte de chaînes de Markov est un outil très puissant et largement étudié dans la littérature, nous prouvons ici qu'il peut être d'une grande utilité dans l'étude de preuves de sécurité des chiffres de Markov. Bien que ce modèle ne soit pas juste au sens strict du terme, il est certainement possible d'en évaluer l'erreur (écart avec la réalité) et d'en déduire une erreur sur les résultats proposés ici, des techniques d'approximation allant dans ce sens existent déjà, la méthode itérative de Takahashi [206] en fait partie.

CS-Cipher 12 rondes de CSC^* correspondant à 4 rondes de CS-Cipher, nous pouvons en déduire que si les modèles de chiffre de Markov et de chiffre de Markov en support ne sont pas trop éloignés de la réalité, il n'existe pas de différentielles impossibles pour CS-Cipher à partir de 4 rondes.

AES De même, pour AES, nous pouvons en déduire que si les modèles de chiffre de Markov et de chiffre de Markov en support ne sont pas trop éloignés de la réalité, il n'existe pas de différentielles impossibles pour AES à partir de 6 rondes.

Améliorations. Les bornes trouvées sont très larges, un certain nombre d'améliorations peuvent sûrement y être apportées, notamment

- Introduire la distribution des probabilités différentielles sur les boîtes-S comme a été fait dans [166]
- Utiliser les meilleurs bornes sur 2 ou 4 rondes (pour la borne supérieure) comme cela est fait dans KMT2-DC [115]

Attaques par analyse de courant et approximations linéaires 6

Sommaire

6.1 Préliminaires 82
6.1.1 Modèle d'exécution d'un chiffre symétrique 82
6.1.1.1 Implémentation sur plate-forme physique 82
6.1.1.2 Fuite d'information : Valeurs Intermédiaires 83
6.1.2 Approximations linéaires 84
6.1.2.1 Attaques par cryptanalyse multi-linéaire 84
6.1.2.2 Approximations linéaires d'une fonction booléenne et décodage par liste de codes de Reed-Muller d'ordre 1 85
6.2 Attaques par analyse de consommation basées sur les approximations linéaires 86
6.2.1 Motivation 86
6.2.2 Une DPA Probabiliste 87
6.2.3 Attaques Multi-linéaires par Analyse de Courant (MLAC) 88
6.2.3.1 Description et Complexité 89
6.2.3.2 MLAC en pratique 90
6.2.3.3 Contre-mesures 91
6.2.4 Résultats d'attaques MLAC 91
6.2.5 Pour aller plus loin 92

Ce chapitre présente une nouvelle façon d'aborder les attaques par canaux auxiliaires en utilisant les notions de la cryptanalyse linéaire conventionnelle. Ainsi nous présenterons comment les approximations linéaires peuvent être utilisées dans une attaque par *analyse de courant différentielle* (DPA) classique. Puis nous introduirons une nouvelle forme d'attaques par analyse de courant ou analyse de radiations électromagnétiques à partir des outils de cryptanalyse multi-linéaire. Les notions de cryptographie et de théorie des codes utilisées dans ce chapitre ont été introduites dans le chapitre 3 (sections 3.1.2, 3.1.3.1, 3.1.4 et 3.2.2).

6.1 Préliminaires

Nous avons vu dans la section 3.1.2.1, comment sont définis les chiffres symétriques par blocs de structure SPN. Nous nous restreindrons à cette structure pour l'étude faite dans ce chapitre, néanmoins tous les résultats présentés ici peuvent être adaptés pour toute structure de chiffre symétrique itératif.
Dans le contexte des attaques par canaux auxiliaires, le chiffre symétrique n'est pas considéré uniquement comme un objet mathématique mais aussi comme une implémentation sur une plate-forme physique. En effet, la perturbation ou la simple observation de l'environnement d'exécution de l'algorithme de chiffrement peut apporter au cryptanalyste de nouvelles données sur l'état interne de l'algorithme durant son exécution.

Remarque 23 *Dans toute la suite du chapitre nous parlerons indifféremment d'attaques par analyse de courant ou de consommation sur un circuit pour désigner les attaques qui utilisent l'observation de la consommation du circuit ou des radiations électromagnétiques. Il est communément admis que ces deux types d'observation donnent sensiblement les mêmes informations sur l'état interne du circuit à l'exécution.*

Dans cette section nous présenterons le modèle d'implémentation d'un algorithme de chiffrement de structure SPN dont découlera la modélisation des *points de fuites* qu'un attaquant peut observer par la mesure de consommation ou de radiations électromagnétiques du circuit. Puis nous présenterons plus en détail la cryptanalyse multi-linéaire mentionnée très brièvement dans la section 3.1.3.1, comment générer de nombreuses approximations et comment monter une attaque efficace à partir de plusieurs approximations linéaires.

6.1.1 Modèle d'exécution d'un chiffre symétrique

Avant de présenter les attaques par analyse de consommation nous devons définir le modèle dans lequel ces attaques sont construites. Ce modèle ne fait pas d'hypothèses qui ne soient généralement acceptées dans la littérature et se veut donc assez proche de la réalité, la grande réussite des attaques par analyse de consommation en est la preuve.

6.1.1.1 Implémentation sur plate-forme physique

L'implémentation d'un chiffre symétrique sur une plate-forme physique, tel qu'un microcontrôleur (*implémentation logicielle*) ou qu'une carte dédiée (FPGA ou ASIC) (*implémentation matérielle*), peut prendre des formes très variées selon les spécifications de la plate-forme et les contraintes du projet (contraintes en débit, en place, en consommation, en sécurité).
Nous considérons que toute plate-forme physique est constituée de registres, de mémoire et de logique combinatoire. Durant un cycle d'horloge, les données placées sur les registres sont envoyées vers la mémoire (via un bus) ou vers la logique combinatoire, à la fin du cycle d'horloge de nouvelles données en entrée des registres y sont enregistrées. Comme vu dans la section 3.1.4.2, la consommation du circuit est dépendante des transitions de valeurs à un instant t et que ces transitions ne peuvent être précisément prédites qu'au niveau d'un registre ou d'un bus du fait de leur utilisation synchrone (voir remarque 5, section 3.1.4.2). En conséquent, à chaque front d'horloge,

l'observation de la consommation donne une information sur la donnée enregistrée dans un ou un ensemble de registres. Pour simplifier notre étude nous ne considérerons que les registres de données et l'exécution de l'algorithme de chiffrement sans la génération des clés de rondes (e.g., les clés de rondes ont déjà été générées et sont accessibles sur une mémoire protégée). Pour une implémentation particulière, la prise en compte de registres d'instructions (dans une implémentation logicielle) ou de l'algorithme de génération de clé ne peut rendre l'attaque que plus efficace (dans notre cas, ces opérations ajouteront du bruit à la mesure de consommation).
Ainsi, à chaque cycle d'horloge, une valeur intermédiaire de la fonction de chiffrement est placée sur un registre. Qui plus est, nous supposerons que l'implémentation logicielle ou matérielle de l'algorithme de chiffrement est connue, i.e., l'attaquant est supposé connaître les opérations exécutées à chaque cycle d'horloge.

6.1.1.2 Fuite d'information : Valeurs Intermédiaires

Soit un algorithme de chiffrement E représenté figure 6.1, de taille de blocs $\boldsymbol{n}$, implémenté sur une plate-forme physique dont les registres de données ont une taille $\boldsymbol{m}$. D'après la section précé-

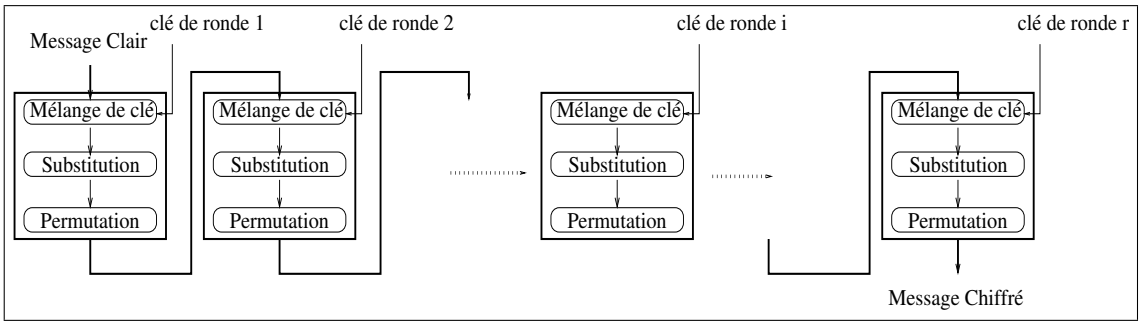

FIG. 6.1 – Chiffre Symétrique Itératif

dente, à chaque cycle d'horloge, $\boldsymbol{m}$ bits de valeurs intermédiaires sont placés sur un registre pour être traités par la logique combinatoire, puis être à nouveau placés sur un registre. Si l'exécution complète de l'algorithme de chiffrement est traitée en C cycles d'horloge nous pouvons conclure, sous notre modèle, que l'attaquant observant la consommation du circuit peut récupérer de l'information sur C valeurs de m bits. Dans toute la suite du chapitre, nous noterons $\{V_i\}_{1\leq i\leq C}$ ces *valeurs intermédiaires* considérées en C *points de fuite*. De plus, considérant le modèle de consommation en distance de Hamming (HD) (resp. en poids de Hamming (HW)) présenté dans la section 3.1.4.2, les valeurs intermédiaires $\{V_i\}_{1\leq i\leq C}$ représenteront des différences de valeurs intermédiaires prenant successivement place dans un registre (resp. des valeurs intermédiaires prenant place dans un registre). Dans la suite du chapitre, les modèles de consommation HD et HW peuvent être utilisés de façon interchangeable. Dans les deux cas, l'information fuite par l'observation de la consommation durant l'exécution d'un algorithme de chiffrement symétrique est l'ensemble $\{H(V_i)\}_{1\leq i\leq C}$ (où $H()$ est la fonction de poids de Hamming).

6.1.2 Approximations linéaires

Pour aborder les attaques par analyses de courant basées sur les approximations linéaires, il convient de définir plus en détail les techniques de cryptanalyse multi-linéaire et leur complexité.

6.1.2.1 Attaques par cryptanalyse multi-linéaire

La cryptanalyse linéaire, présentée dans la section 3.1.3.1, peut-être améliorée par l'utilisation de plusieurs approximations linéaires (idée introduite en 1994 par Kaliski et al. [107]). En 2004, Biryukov et al. [33] ont prouvé que la donnée de l approximations linéaires indépendantes (i.e. les masques sur les bits de clé sont indépendants) de biais respectifs $\epsilon_1, \epsilon_2, \cdots, \epsilon_l$, permet de monter une attaque de complexité en données $\frac{1}{\sum_{i=1}^{l} \epsilon_i^2}$. Dans un cadre plus général, Gerard et Tillich [85] ont fait le lien entre retrouver les bits de clés intervenant dans l approximations linéaires et le *décodage souple* (Soft-decoding en anglais) de codes correcteurs linéaires, améliorant ainsi les performances, tout en utilisant des approximations qui n'ont plus besoin d'être indépendantes deux à deux.

Approche simple Rappelons qu'une approximation linéaire d'un algorithme de chiffrement E, de taille de bloc n et taille de clé k, est définie par la donnée d'un triplet de masques $(\alpha, \gamma, \beta) \in \{0;1\}^n \times \{0;1\}^k \times \{0;1\}^n$ et d'un réel $\epsilon \in [-\frac{1}{2}, \frac{1}{2}]$ (le biais de l'approximation) tels que :

$$\Pr_{(X,K)}[\langle \alpha, X \rangle \oplus \langle \gamma, K \rangle \oplus \langle \beta, E(X,K) \rangle = 0] = 1/2 + \epsilon$$

où $\langle ., . \rangle$ est le produit scalaire de vecteurs d'éléments dans $GF(2)$ et $\oplus$ le *ou exclusif bit à bit* (ou encore l'addition dans $GF(2)^n$).
Considérons l approximations linéaires $\{(\alpha_i, \gamma_i, \beta_i), \epsilon_i\}_{1 \leq i \leq l}$ pour l'algorithme de chiffrement E, telles que $\{\gamma_i\}_{1 \leq i \leq l}$ font intervenir s bits de la clé secrète et forment une matrice de taille $l \times s$ de rang s. Nous obtenons alors un système d'équations linéaires (sur-défini si $l > s$) :

$$\left\{ \begin{array}{lcl|l} \langle \gamma_1, c \rangle & = & \langle \alpha_1, x_j \rangle \oplus \langle \beta_1, E(x_j, c) \rangle & p_1 = 1/2 + \epsilon_1 \\ \langle \gamma_2, c \rangle & = & \langle \alpha_2, x_j \rangle \oplus \langle \beta_2, E(x_j, c) \rangle & p_2 = 1/2 + \epsilon_2 \\ \cdots & & \cdots & \cdots \\ \langle \gamma_l, c \rangle & = & \langle \alpha_l, x_j \rangle \oplus \langle \beta_l, E(x_j, c) \rangle & p_l = 1/2 + \epsilon_l \end{array} \right.$$

où c est la clé secrète et $\{x_j\}_{0 \leq j \leq N}$ un ensemble de messages clairs connus.
Si N est assez grand, il est possible d'évaluer avec une grande probabilité la partie droite du système d'équations, de résoudre celui-ci et de retrouver les s bits de clé.

Décodage d'un code de Reed-Muller d'ordre 1 La résolution du système d'équations proposé peut être vue comme le décodage d'un code de Reed-Muller binaire d'ordre 1 : $\mathfrak{R}(1, s)$ (voir section 3.2.2). Rappelons que les codes $\mathfrak{R}(1, s)$ définissent l'ensemble des polynômes de degré 1 à s variables dans $GF(2)$ (i.e., fonctions affines à s variables définies dans $GF(2)$). L'algorithme de codage d'un code $\mathfrak{R}(1, s)$ consiste à évaluer une telle fonction affine sur les 2^s points possibles. Le mot de code obtenu est donc un élément de $GF(2)^{2^s}$. La distance minimale du code vaut, au

moins, 2^{s-1}. Il est donc possible de corriger un mot $\tilde{y}$ en un mot de code y si le nombre d'effacements e et le nombre de fautes t sont tels que $2t + e \leq 2^{s-1} - 1$ (voir section 3.2.1.1).
Ainsi, en admettant que la valeur de la clé secrète définie une fonction affine à s variables sur $GF(2)$, il suffit d'évaluer la partie droite du système d'équations en assez de messages clairs $\{x_j\}_{0 \leq j \leq N}$ pour que le nombre d'effacements (i.e., valeurs de γ_i correspondante à aucune approximation) et le nombre d'erreurs (dû aux biais des équations) donnent un mot $\tilde{y}$ corrigible.
Le décodage se fait alors efficacement par l'utilisation de la transformée de Hadamard rapide tel que présenté dans la section 3.2.2.1.

Remarque 24 *Pour optimiser au maximum les performances, un* décodage souple *est utilisé : Au lieu de faire un choix strict sur la valeur de la partie droite du système d'équation (1 ou 0), une valeur plus fine pondérée par le biais (grossièrement il faut faire plus confiance à une approximation avec un biais fort). La valeur utilisée en pratique sera* $(\#\{S_0^i\} - \#\{S_1^i\}) ln(\frac{1/2-\epsilon_i}{1/2+\epsilon_i})$ *pour la* i*ème équation où* $\#\{S_0^i\}$ *(resp.* $\#\{S_1^i\}$*) est le nombre de messages* x *qui ont conduit à l'évaluation de l'équation à 0 (resp. 1) (cf. [72] pour plus de détails).*

Nous voici donc avec une attaque efficace contre un chiffre symétrique dans le cas où nous possédons un grand nombre d'approximations linéaires de l'algorithme de chiffrement (avec des biais suffisant). La question que se pose maintenant est : Comment trouver de telles équations ? Nous avons vu, dans la section 3.1.3.1, comment construire de telles approximations en utilisant le lemme *Piling-Up* de Matsui, ici nous utiliserons un méthode plus récente, plus générique et qui ne fait pas appel à l'hypothèse d'indépendance des boîtes-S.

6.1.2.2 Approximations linéaires d'une fonction booléenne et décodage par liste de codes de Reed-Muller d'ordre 1

Les résultats auxquels nous nous référons ici sont issus du travail récent de R. Fourquet, P. Loidreau et C. Tavernier [77] qui se sont penchés sur la difficulté de trouver des approximations linéaires d'une fonction booléenne avec un biais fixé.

Il est montré dans [77] que trouver l'ensemble des approximations linéaires pour un masque β de sortie fixé d'un algorithme de chiffrement E de biais supérieur à un biais fixé ϵ est équivalent à décoder par liste dans un code de Reed-Muller binaire d'ordre 1.
Considérons un masque β et la fonction booléenne f_E qui à un couple $(x, c) \in \{0;1\}^n \times \{0;1\}^k$ associe la valeur $\langle \beta, E(x,c) \rangle \in \{0;1\}$. Trouver une approximation linéaire de E avec le masque de sortie β et de biais ϵ revient donc à trouver un couple de masques (α, γ) tel que $f_E(x,c) = \langle \alpha, x \rangle \oplus \langle \gamma, c \rangle$ pour une fraction $2^{nk} \times (1/2 + \epsilon)$ de l'ensemble des entrées (i.e. $\{0;1\}^n \times \{0;1\}^k$).
Il s'agit donc bien de trouver une fonction affine à $k + n$ variables définie sur $GF(2)$ qui coïncide avec f_E sur une sous-partie des entrées. Nous nous trouvons ainsi ramené à un problème de décodage dans un code de Reed-Muller d'ordre 1 pour un canal bruité. En pratique, le canal considéré est fortement bruité, car ϵ est tel que le nombre d'erreurs est largement supérieur à la limite de correction de ces codes ($d_{min} = 2^{kn-1}$). Il est donc nécessaire d'utiliser un décodage par liste. Le deuxième problème auquel il faut faire face est la taille de l'espace d'entrée qui interdit tout espoir d'estimer f_E en tout point. Il faut donc faire appel à un décodage probabiliste qui estime, avec une marge d'erreur, la fraction des entrées où f_E et l'approximation linéaire considérée coïncident.

Dans la section 3.2.2.2, nous avons vu un algorithme itératif de décodage par liste des codes de Reed-Muller binaires d'ordre 1 qui convient parfaitement à notre problème. C'est celui que nous utiliserons dans la suite. Notons que la complexité de cet algorithme pour la recherche d'approximation de biais $\geq \epsilon$ est de l'ordre de $O(1/\epsilon^2)$, qui correspond, en ordre de grandeur, à la complexité de l'attaque par cryptanalyse multi-linéaire présentée dans la section précédente.

Remarque 25 *Notons tout de même que cette recherche est faite à masque de sortie (β) fixé. Trouver le masque de sortie optimal pour une fonction E reste un problème ouvert et, dans le cas d'un chiffre symétrique, il est illusoire de tous les essayer (il en existe 2^n). Dans [77], les auteurs proposent d'utiliser les masques trouvés par Matsui pour l'étude du chiffre DES.*

Maintenant que les outils de cryptanalyse linéaire ont été définis et que le modèle de fuite d'une implémentation d'un chiffre symétrique sur une plate-forme physique a été fixé, nous pouvons présenter les attaques par analyse de consommation basées sur les approximations multi-linéaires.

6.2 Attaques par analyse de consommation basées sur les approximations linéaires

6.2.1 Motivation

Nous avons vu dans la section 3.1.4.2 les attaques par *analyse de consommation différentielle* (DPA) et les contre-mesures qui existent contre ce type d'attaque. Bien que très efficace sur des plate-formes physiques sans contre-mesure, la DPA souffre d'un inconvénient : Elle ne permet pas d'utiliser toute l'information contenue dans l'ensemble $\{V_i\}_{1\leq i\leq C}$. En effet, une attaque de type DPA repose sur l'évaluation d'une valeur intermédiaire (V_i ou une partie de V_i) sous une hypothèse de clé. Par des méthodes statistiques, il est alors possible de distinguer les fausses hypothèses de clé des bonnes. Soit V_i une valeur intermédiaire cible, si V_i est dépendante de k bits de clé (i.e. il faut faire une hypothèse sur k bits de la clé pour évaluer V_i), il faudra donc traiter $O(2^k)$ hypothèses. En pratique, k est limité à 32 pour que l'attaque soit possible. Ainsi, suivant la fonction de diffusion de l'algorithme de chiffrement considéré, une attaque DPA ne pourra pas cibler de valeurs intermédiaires après quelques rondes de chiffrement. Cette limitation permet par exemple de restreindre les contre-mesures (souvent coûteuses) aux quelques rondes initiales et finales de la fonction de chiffrement.
D'autre part, les méthodes statistiques supposent de réaliser l'attaque sur un échantillon de messages clairs (ou chiffrés) de taille non négligeable. L'utilisation combinée de l'information fuite en plusieurs points permettrait de diminuer la complexité de l'attaque (i.e., nombre de messages observés).

Plusieurs autres types d'attaques par analyse de courant proposent de dépasser ces limitations. Elle sont souvent basées sur la cryptanalyse conventionnelle (voir section 3.1.4.2), en particulier, la récente proposition d'attaques par canaux auxiliaires basées sur la cryptanalyse algébrique (Renauld et Standaert [178, 177]) permet d'envisager que la complexité d'une attaque par analyse de consommation soit réduite à son minimum (un seul message) en utilisant la totalité des points de fuite simultanément.

Remarque 26 *En plus d'un point, les attaques présentées dans [178, 177] sont liées aux attaques présentées dans la suite ce chapitre. Ainsi, lorsque nous travaillons ici sur des systèmes d'équations linéaires correspondant à des approximations de l'algorithme de chiffrement, les attaques algébriques s'intéressent aux systèmes d'équations représentant exactement l'algorithme de chiffrement (donc non linéaires). L'utilisation d'équations linéaires permet une résolution efficace du système considéré en comparaison à la résolution d'un système d'équations de plus haut degré, mais implique un plus grand nombre d'évaluations (i.e., nombre de couples clair-chiffrés connus) du fait du caractère probabiliste des équations. Ces deux approches sont finalement assez proches et permettront, dans beaucoup de cas, les mêmes constructions d'attaques (elles différeront évidemment dans les détails de construction et de résolution de systèmes d'équations).*

6.2.2 Une DPA Probabiliste

L'utilisation d'approximations linéaires dans le domaine des attaques par analyse de consommation s'adapte très naturellement aux attaques DPA : il suffit d'utiliser une approximation linéaire comme fonction de sélection. Cela permet de créer une attaque par DPA sur une valeur intermédiaire strictement dépendante de plus de 32 bits de clé. En effet une approximation permet d'évaluer (avec une probabilité d'erreur) une valeur intermédiaire V_i sans faire intervenir tous les bits de clé dont V_i est dépendante. On pourra donc appliquer une attaque par DPA plus profondément dans l'algorithme de chiffrement. L'attaquant aura ainsi un plus grand nombre de points de fuite à sa disposition.
Toutefois, modifier la fonction de sélection *exacte* d'une attaque par DPA classique pour une fonction de sélection *probabiliste* a nécessairement un impact sur la complexité de l'attaque. Nous allons voir comment.

Rappelons les notations introduites dans la section 3.1.4.2 pour la présentation des attaques par DPA. Nous nous limiterons au cas de distance des moyennes pour une fonction de sélection sur 1 bit, cette étude s'adapte de la même façon aux autres types de méthode statistique.
Une fonction de sélection sur un bit conduit, sous une hypothèse de clé $\hat{K}$ à deux ensembles de traces $S^0_{\hat{K}}$ et $S^1_{\hat{K}}$, représentant respectivement l'ensemble des traces pour lesquelles le bit ciblé a été évalué à 0 et les traces pour lesquelles le bit a été évalué à 1. La distance des moyennes s'écrit alors :

$$\Delta_{\hat{K}} = \mathbf{E}[I_K(X)|I_K(X) \in S^1_{\hat{K}}] - \mathbf{E}[I_K(X)|I_K(X) \in S^0_{\hat{K}}]$$

Où $I_K(x)$ représente la consommation du circuit lorsque la valeur intermédiaire (un seul bit ici) V_i est manipulée pour le message clair x. L'espérance est prise sur la variable aléatoire X, représentant les messages clairs uniformément distribués lors d'une attaque à clairs connus. Ici K est fixé, la valeur $\Delta_{\hat{K}}$ la plus grande sur l'ensemble des choix $\hat{K}$, implique $K = \hat{K}$.
Soit $\hat{K}$, une hypothèse de clé et K la clé secrète. Pour alléger les formules, nous supprimerons dans la suite les indices $\hat{K}$ et K. Nous avons donc :

$$\Delta = \mathbf{E}[I(X)|I(X) \in S^1] - \mathbf{E}[I(X)|I(X) \in S^0]$$

Considérons maintenant une fonction de sélection non exacte avec une probabilité de réussite $p = 1/2 + \epsilon$. Le choix de mettre la mesure de consommation $I(x)$ dans l'ensemble S^1 ou S^0 est

donc soumis à une probabilité d'erreur, notons $\widehat{S}^1$ et $\widehat{S}^1$ les ensembles choisis par la fonction de sélection probabiliste et gardons S^1 et S^0, les ensembles correspondant à l'évaluation exacte de la valeur intermédiaire. Nous avons alors une nouvelle différence des moyennes :

$$\widehat{\Delta} = \mathbf{E}[I(X)|I(X) \in \widehat{S}^1] - \mathbf{E}[I(X)|I(X) \in \widehat{S}^0]$$

Par la loi des probabilités totales et la linéarité de l'espérance, $\widehat{\Delta}$ peut s'écrire :

$$\begin{array}{rcl} \widehat{\Delta} & = & (p\mathbf{E}[I(X) \mid I(X) \in \widehat{S}^1, I(X) \in S^1] + (1-p)\mathbf{E}[I(X) \mid I(X) \in \widehat{S}^1, I(X) \in S^0]) \\ & - & (p\mathbf{E}[I(X) \mid I(X) \in \widehat{S}^0, I(X) \in S^0] + (1-p)\mathbf{E}[I(X) \mid I(X) \in \widehat{S}^0, I(X) \in S^1]) \end{array}$$

Notons que pour tout couple $(l, j) \in \{1; 2\}^2$, l'événement $(I(X) \in \widehat{S}^l)$ donne toujours moins d'information que $(I(X) \in S^j)$ sur la valeur moyenne de $I(X)$. Nous obtenons donc :

$$\begin{array}{rcl} \widehat{\Delta} & = & (p\mathbf{E}[I(X) \mid I(X) \in S^1] + (1-p)\mathbf{E}[I(X) \mid I(X) \in S^0]) \\ & - & (p\mathbf{E}[I(X) \mid I(X) \in S^0] + (1-p)\mathbf{E}[I(X) \mid I(X) \in S^1]) \end{array}$$

et alors

$$\begin{array}{rcl} \widehat{\Delta} & = & 2\epsilon(\mathbf{E}[I(X) \mid I(X) \in S^1] - \mathbf{E}[I(X) \mid I(X) \in S^0]) \\ & = & 2\epsilon\Delta \end{array}$$

Pour finir, la complexité de l'attaque est multipliée par un facteur $1/(2\epsilon)$ pour une fonction de sélection probabiliste de biais ϵ.

Remarque 27 *Cette attaque s'adapte bien lorsque plusieurs approximations linéaires d'une même valeur intermédiaire sont à disposition. Pour chaque trace, le choix de la mettre dans S^0 ou S^1 devra être fait autant de fois qu'il y a d'approximations. Les ensembles de traces contiendront donc plusieurs instances de chaque trace et les mesures de consommation devront être pondérées par le biais de la fonction de sélection associée.*

Remarque 28 *Cette attaque s'adapte très mal aux DPA multi-bits. En effet, les approximations étant des fonctions booléennes, il faut m approximations pour évaluer m bits de valeur intermédiaires. La fonction de sélection probabiliste constituée de ces m approximations aura donc la probabilité $p = \prod_{i=1}^{m} p_i$ (produit des probabilités des m approximations linéaires). Rapidement, une telle fonction de sélection devient inutilisable, ayant un biais trop faible.*

Cette dernière remarque limite grandement l'efficacité d'une attaque DPA probabiliste, les meilleures attaques DPA actuelles étant des attaques DPA multi-bits. Nous allons voir dans la prochaine section une façon bien plus efficace d'utiliser des approximations linéaires pour construire une attaque par analyse de courant.

6.2.3 Attaques Multi-linéaires par Analyse de Courant (MLAC)

En admettant que nous ayons un modèle de fuite assez proche de la réalité, il est possible d'utiliser directement des approximations linéaires de cette fonction de fuite. Ainsi, au lieu d'évaluer la valeur intermédiaire V_i pour $1 \leq i \leq C$ nous évaluerons directement la fuite d'information, i.e. $H(V_i)$ pour les modèles HW et HD (cf notations de la section 6.1.1.2).

Supposons donc dans un premier temps que l'observation de la consommation d'un circuit sur lequel est exécuté un chiffre symétrique (conformément au modèle défini dans la section 6.1.1) nous permet d'évaluer avec précision les valeurs $\{H(V_i)\}_{1\leq i\leq C}$. La prochaine section présente une attaque par analyse de consommations basée sur la cryptanalyse multi-linéaire (cf section 6.1.2.1) sous cette hypothèse. Ensuite nous verrons comment, en pratique, se rapprocher de cette hypothèse ou la contourner.

6.2.3.1 Description et Complexité

Soit V_i une valeur intermédiaire sur un point de fuite de l'exécution d'un chiffre symétrique. Supposons que nous ayons trouvé L approximations linéaires $\{(\alpha_l, \gamma_l, \beta_l^H), \epsilon_l\}_{1\leq l\leq L}$ de la fonction de fuite en V_i. Dans notre cas, la fonction de fuite associe $H(V_i(x, c))$ à un message x et à une clé c. C'est pourquoi nous avons :
$\forall 1 \leq l \leq L$,

$$\Pr_{(X,K)}[\langle \alpha_l, X\rangle \oplus \langle \gamma_l, K\rangle \oplus \langle \beta_l^H, H(V_i(X, K))\rangle = 0] = 1/2 + \epsilon_l$$

Notons que β_l^H est un masque du poids de Hamming de la valeur intermédiaire, donc sur $\log_2(m) + 1$ bits d'après les notations de la section 6.1.1.2.
Sous l'hypothèse que l'observation de la consommation au niveau du point de fuite correspondant à V_i permet d'évaluer avec précision $H(V_i)$, nous pouvons appliquer directement l'attaque par cryptanalyse multi-linéaire présentée dans la section 6.1.2.1.
Plus précisément, voici le détail de l'attaque :

Phase "hors ligne" : Trouver L approximations linéaires $\{(\alpha_l, \gamma_l, \beta_l^H), \epsilon_l\}_{1\leq l\leq L}$ faisant intervenir s bits de la clé secrète c. Puis,

1. Observer N exécutions de l'algorithme de chiffrement au point de fuite correspondant à la valeur intermédiaire V_i. A la fin de cette étape nous sommes en possession des valeurs $\{H(V_i(x_j, c))\}_{1\leq j\leq N}$ (pour les N messages clairs connus x_j et la clé secrète c).
2. Pour chaque approximation linéaire, évaluer la valeur de $\langle \gamma_l, c\rangle$ (0 ou 1) pour chacune des valeurs $\{H(V_i(x_j, c))\}_{1\leq j\leq N}$. Notons N_0^l le nombre d'évaluation à 0 et N_1^l le nombre d'évaluation à 1 de $\langle \gamma_l, c\rangle$ pour la $l$ème approximation.
3. Construire un vecteur $\widetilde{y}$ de taille 2^s à valeurs dans les réels tel que,

   $$\forall l \in \{1, \cdots, L\}, y[\gamma_l] = (N_0^l - N_1^l) ln(\frac{1/2 - \epsilon_i}{1/2 + \epsilon_i})$$

   Où γ_l est vu comme un élément de $GF(2^s)$. Les coordonnées de $\widetilde{y}$ qui restent vides sont misent à zéro (elles sont considérées comme des effacements).
4. Décoder $\widetilde{y}$ dans le code de Reed-Muller binaire du premier ordre par *décodage souple*. Ceci revient à trouver la fonction affine à s variables $y()$ définie sur $GF(2)$ qui maximise $y \otimes \widetilde{y} = \sum_{x\in\{0;1\}^s}(-1)^{y(x)}\widetilde{y}[x]$. Une transformée de Fourier rapide permettra d'évaluer $y \otimes \widetilde{y}$ pour tout y avec une complexité en temps $O(s2^s)$ et en données $O(2^s)$.

Remarque 29 *L'algorithme ne considère qu'une seule valeur intermédiaire V_i. Il peut être appliqué à plusieurs valeurs intermédiaires simultanément. Chaque approximation sera alors définie par le point de fuite auquel elle correspond.*

La phase "hors ligne" de l'attaque qui consiste à construire les approximations linéaires $\{(\alpha_l, \gamma_l, \beta_l^H), \epsilon_l\}_{1 \leq l \leq L}$ est faite à l'aide de l'algorithme proposé dans [77] comme présenté dans la section 6.1.2.2. A noter que la complexité de la recherche d'approximations est généralement plus grande que celle de l'attaque en elle-même. Pourtant, pour un algorithme de chiffrement donné, cette génération n'est à faire qu'une unique fois, les approximations peuvent alors être utilisées sur plusieurs implémentations différentes.

Remarque 30 *Notons ici que l'ensemble des masques de sortie est maintenant très restreint ($m+1$ éléments). Contrairement au cas d'approximations linéaires d'un algorithme de chiffrement (cf remarque 25) il sera donc possible de chercher des approximations pour tous les masques de sortie possibles.*

6.2.3.2 MLAC en pratique

En pratique, l'observation de la consommation d'un circuit exécutant un chiffre symétrique, même sous notre modèle d'implémentation, ne permet pas d'évaluer à coup sûr le poids de Hamming de la valeur intermédiaire V_i choisie. Même si la mesure de courant était parfaite, le bruit généré par les opérations en cours mais n'entrant pas directement dans le calcul de V_i rend la chose impossible. Différents moyens de s'en approcher ou de contourner ce problème peuvent être proposés :

1. La solution la plus simple, néanmoins la moins efficace, consiste à décider de la valeur de $H(V_i(x, c))$ par une analyse statistique sur l'ensemble des N traces utilisées pour l'attaque. Ainsi, à partir des N valeurs de consommation $\{I(x_j, c)\}_{1 \leq j \leq N}$ au point de fuite correspondant à la valeur intermédiaire V_i, et en considérant que le poids de Hamming de $V_i(X, c)$ suit une loi normale lorsque la variable aléatoire X suit une distribution uniforme, il est possible d'évaluer (avec une certaine probabilité d'erreur) la valeur de $H(V_i(x, c))$. Dans une forme très simple, cette solution a été expérimentée sur des traces très propres (peu de bruit) provenant du dpa-contest (voir section 6.2.4) et donne de bons résultats. Il est vraisemblable que l'ajout de bruit rende cette attaque beaucoup plus coûteuse.
2. Une solution plus efficace serait d'utiliser l'hypothèse de la *carte d'entraînement* (comme proposé pour les attaques Template [52]). Il s'agit de supposer que l'attaquant a accès à une implémentation du chiffre strictement identique à l'implémentation ciblée pour l'attaque. Ainsi, cela lui permet de construire une bibliothèque de données, rapprochant chaque valeur possible $H(V_i)$ avec une valeur de consommation, et ceci en tout point $i \in \{1; \cdots; C\}$. Cette phase de pré-calcul permet d'estimer, avec une précision plus grande, la valeur de $H(V_i(x, c))$ à partir de $I(x, c)$. Cette solution est utilisée dans les travaux de Renauld et Standaert pour les attaques par canaux auxiliaires algébriques [178, 177].
3. Enfin, toujours dans l'hypothèse de la *carte d'entraînement*, probablement la solution la plus efficace est de construire directement les approximations linéaires par l'observation

de la consommation de la carte d'entraînement. Ceci est possible grâce à l'algorithme de recherche d'approximations linéaires qui travaille sur une fonction booléenne en *boîte noire* (i.e. sans se préoccuper de la structure de la fonction). Ainsi, l'idée est d'appliquer cet algorithme directement sur la fonction $I(X, K)$ en un point de fuite choisi. Cette solution présente un avantage considérable, elle permet de se libérer de tout modèle de fuite. Il n'est plus nécessaire de forcer un lien entre $I(X, K)$ et $V_i(X, K)$, les approximations lieront directement la consommation du circuit en un point donné avec les entrées (ou sorties) de l'algorithme de chiffrement.
Plus intéressant encore, on remarque que, le choix du point de fuite où monter l'attaque n'est plus très important, il suffit de choisir le moment sur l'ensemble des traces qui conduit aux meilleures approximations, pour en déduire le point de fuite le plus intéressant pour l'attaque. Lors de l'attaque il faut alors récupérer les mesures au même moment et travailler directement sur les valeurs de consommation mesurées.

6.2.3.3 Contre-mesures

Avant de décrire les expérimentations et résultats sur le sujet, il paraît important d'aborder le problème des contre-mesures qui peuvent être appliquées à une plate-forme physique pour éviter ce type d'attaques.
Les contre-mesures contre la DPA (présentées dans la section 3.1.4.2) sont tout-à-fait pertinentes contre les autres attaques par analyse de consommation. Notons tout de même que lorsque les attaques algébriques par canaux cachés peuvent, sous certaines conditions, attaquer une implémentation dotée d'une contre-mesure par masquage additif (voir [178, 177] pour plus de détails), il est possible de faire de même dans le contexte des attaques MLAC (cf remarque 26).

6.2.4 Résultats d'attaques MLAC

Voici les résultats obtenus en attaquant une implémentation matérielle du chiffre symétrique DES. Les traces de consommations sont publiques et ont été utilisées lors du *concours DPA* proposé par le groupe de recherche VLSI du département COMELEC de l'école d'ingénieur française Telecom ParisTech (voir `http://www.dpacontest.org/`). Ce concours a pour but d'évaluer les meilleures attaques par analyses de courant sur un jeu de traces commun à tous les participants. Notons ici que le concours a été un grand succès puisque de nombreuses propositions d'attaques y ont été soumises, les meilleurs d'entre elles permettent de trouver la clé secrète en moins de 100 mesures. Pourtant les meilleurs attaques utilisent un sous-ensemble préchoisi de traces, permettant d'améliorer les performances. Ce n'est évidement pas possible dans le cadre d'une attaque réelle, mais cela donne une idée de la complexité en meilleurs cas des attaques DPA sur ce type de traces. Les expériences que nous avons réalisées ont été faite sur des sous-ensembles de traces choisies aléatoirement parmi l'ensemble des traces disponibles (plus de 80000). Les résultats présentés sur la figure 6.2 et dans le tableau 6.1 sont des résultats moyens sur 1000 sous-ensembles de taille fixée de traces.

Description des résultats L'implémentation utilisée pour le concours dpa est une implémentation matérielle du chiffre DES tel que le registre LR de données contient 64 bits (taille de bloc

du chiffre DES) et chaque ronde est exécutée en un cycle d'horloge, le chiffre DES en comporte 16. Ainsi, avec les notations de ce chapitre, les valeurs intermédiaires $\{V_i\}_{1\leq i\leq C}$ sont sur 64 bits, sont au nombre de 16 et représentent une différence entre une entrée et une sortie de ronde (i.e. nous nous plaçons dans le modèle HD). Les détails de la mise en place de l'attaque ainsi que des exemples d'approximations linéaires utilisées pour ces attaques peuvent être trouvés dans l'annexe B de ce document.
La recherche d'approximations linéaires sous ces contraintes a abouti à quatre ensembles d'équations linéaires. Ils correspondent respectivement à la valeur intermédiaire manipulée en sortie de première ronde, de deuxième ronde, de 15ème ronde et 16ème ronde.
La figure 6.2 représente l'évolution du nombre de bits de clé retrouvés en moyenne par rapport à la complexité de l'attaque (en nombre de traces). Ceci pour chaque point de fuite considéré, mais aussi en appliquant l'attaque simultanément sur plusieurs points de fuites (marqués "$1||15||16$" et "$1||2||15||16$"). Les extrémités des barres verticales représentent les valeurs maximales et minimales atteintes sur l'ensemble des 1000 tests (les intervalles de confiances, invisibles sur les courbes sont inférieurs à 1% dans tous les cas).

Remarque 31 *L'attaque en sortie de deuxième ronde est moins efficace que sur les autres points de fuite. Ceci n'est pas dû à une complexité algébrique supérieure (qui diminuerait les biais des approximations linéaires) puisque la complexité algébrique après deux rondes est strictement identique à celle avant la 15ème ronde. En fait, il y a deux raisons à cela : D'une part le choix du point de fuite (Quel moment choisir sur l'ensemble d'une trace ?) n'est peut-être pas le meilleur choix qui puisse être (bien que nous n'en ayons pas trouvé de plus pertinent pour cette valeur intermédiaire). D'autre part il apparaît que les traces de consommation sont plus bruitées en début de chiffrement qu'en fin de chiffrement (les figures B.2 et B.3 qui se trouvent en annexe B le montrent clairement). La conjonction de ces deux circonstances rend la complexité d'attaque plus forte après deux rondes. En augmentant le nombre d'approximations nous arrivons à rattraper un peu cet écart.*

Le tableau 6.1 récapitule les résultats les plus importants pour chaque attaque et donne le nombre de d'équations linéaires trouvées pour chaque point de fuite. La première colonne, désigne le choix d'une attaque sur un point de fuite unique ou simultanément sur plusieurs points de fuites. La colonne "rondes" désigne les points de fuites (ou valeurs intermédiaires) visés par l'attaque. Les colonnes intitulées "# équ. linéaires", "# bits de clés" et "# traces" contiennent respectivement le nombre d'équations linéaires utilisées, le nombre bits de clé retrouvés en moyenne et la complexité de l'attaque en nombre de traces. Notons qu'une attaque permet en moyenne de retrouver 40 bits de clés avec 500 traces pour une attaque simple. La complexité descend à 100 traces pour le même nombre de bits de clé lorsque plusieurs points sont utilisés dans chaque trace.

6.2.5 Pour aller plus loin

Nous avons évoqué dans la remarque 26 la similitude entre l'approche présentée dans ce chapitre et les attaques par canaux cachés basées sur la cryptanalyse algébrique due à Renauld et Standaert [178, 177]. Il est intéressant de souligner ici que ces deux approches encadrent un champ

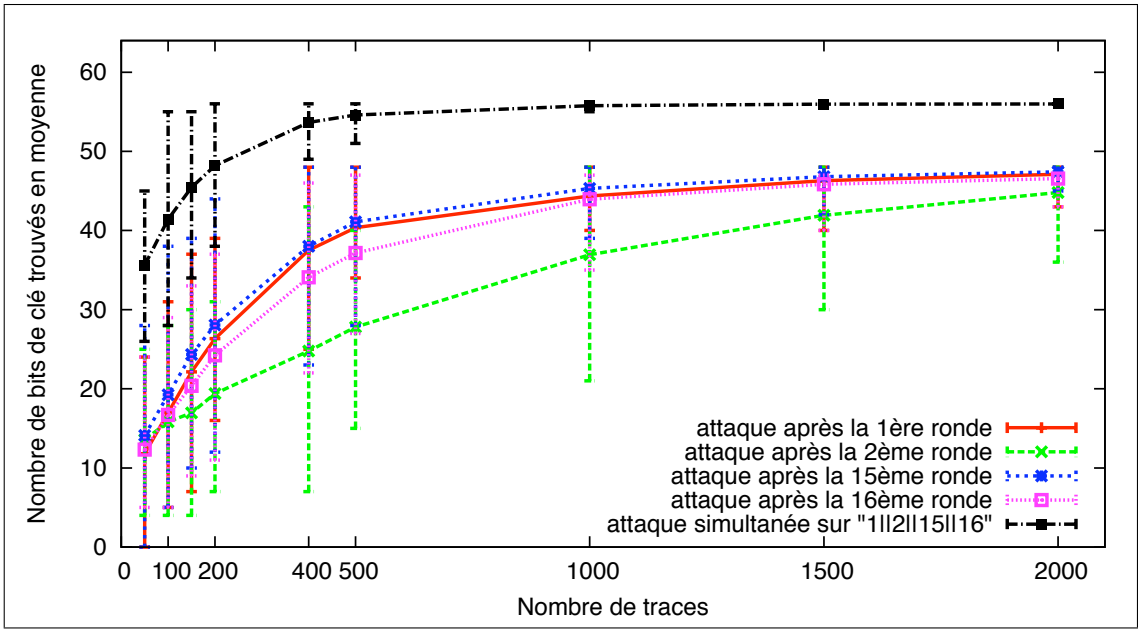

FIG. 6.2 – Résultat d'attaque MLAC sur les traces du concours DPA

DES	rondes	# équ. linéaires	# bits de clés	# traces
Valeur intermédiaire unique	1	533	40	500
	2	1452	27	500
	15	488	41	500
	16	446	37	500
Plusieurs Valeurs intermédiaires	1||15||16	1467	36	100
	1||15||16	1467	53	500
	1||2||15||16	2919	41	100
	1||2||15||16	2919	54	500

TAB. 6.1 – Résultat d'attaque MLAC sur les traces du concours DPA : résumé

de recherche encore vierge qui recèle peut-être la méthode d'attaque par analyse de courant la plus puissante. Ainsi l'utilisation d'approximations de plus haut degré permettra d'approximer plus profondément encore le chiffre symétrique considéré et donc de construire des attaques à la fois plus efficaces (en terme de nombre de traces) et qui touchent plus de valeurs intermédiaires. D'autre part, en faisant augmenter le degré des équations, on se rapproche du système considéré dans [178, 177]. Une autre façon d'augmenter l'efficacité de ces attaques serait de les combiner, par exemple en intégrant des approximations linéaires au système d'équations algébriques afin d'aider à sa résolution.

CS-Cipher 7

Sommaire

7.1 Présentation 95
7.1.1 CS-Cipher 95

Le sujet de recherche sur le "dimensionnement et intégration d'un chiffre symétrique dans le contexte d'un système d'information distribué de grande taille" a été appliqué à un chiffre symétrique particulier : le chiffre CS-Cipher. Plusieurs raisons sont à l'origine de ce choix :

- Le chiffre a été conçu sous la direction de l'entreprise Communication et Systèmes, celle-ci est donc assurée de que le chiffre ne contient pas une "porte dérobée" potentiellement introduite par les constructeurs du chiffre.
- Le chiffre ne présente pas de faiblesse connue.
- La structure interne du chiffre permet facilement les modifications et extensions.

Dans la première section sera présenté le chiffre original CS-Cipher. Le travail de dimensionnement et d'intégration du chiffre n'est pas présenté dans cette version publique de la thèse, les recherches ainsi que les résultats de ces travaux sont la propriété de la société Communication et Système.

7.1 Présentation

Le chiffre CS-Cipher a été conçu par Jacques Stern et Serge Vaudenay en 1998 [205] dans le cadre d'un partenariat avec la Compagnie des Signaux (aujourd'hui *Communication et Systèmes*). Il fut proposé au projet international NESSIE ("New European Schemes for Signatures, Integrity and Encryption"[1]) où il ne passe pas au deuxième tour bien que n'ayant aucune faiblesse connue [2] (certainement pour des questions de performance).
Nous nous contentons ici de présenter le chiffre CS-Cipher

7.1.1 CS-Cipher

Nous nous contentons ici de décrire brièvement le chiffre CS-Cipher en reprenant les notations et la description originale de [205].

[1] `https://www.cosic.esat.kuleuven.be/nessie/`

Notations Dans toute la suite, nous noterons,
- $\oplus$ est l'opération *ou exclusif* bit-à-bit ou addition dans le corps $\mathbb{F}_2^n$.
- $\wedge$ est l'opération *et* bit-à-bit.
- $||$ est la concaténation de chaînes de bits.
- $R_l(x)$ est rotation circulaire à gauche des bits de x.
- $\overline{x}$ est le passage au complémentaire de x : $\overline{x} = x \oplus (11\cdots111)_2$.
- $(x_0||x_1||\cdots||x_{n-1})_{16}$ représente un entier $x \in \mathbb{F}_2^{8n}$ en hexadécimal par bloc que 8 bits.
- Soit f une fonction de $\mathbb{F}_2^n$ dans lui-même, f^p est la fonction de $\mathbb{F}_2^{pn}$ dans lui-même telle que $\forall x = (x_0||x_1||\cdots||x_{p-1})_{16} \in \mathbb{F}_2^{pn}, f^p(x) = (f(x_0)||f(x_1)||\cdots||f(x_{p-1}))_{16}$

CS-Cipher est un *chiffre clé-alternant*, tel que l'on peut le voir sur la figure 7.1. Le tableau 7.1 contient les paramètres globaux de CS-Cipher.

TAB. 7.1 – CS-Cipher : Paramètres Globaux

Paramètres	Valeur
Taille de bloc	64 bits
Taille de clé	128 bits
Nombre de rondes	8

Génération des clés de ronde La fonction de génération de clés de rondes, visible sur la partie droite sur la figure 7.1, a une structure de Feistel, elle prend en entrée une clé sur 128 bits et génère 9 clés de rondes de 64 bits. Comme le veut la structure clé-alternant de CS-Cipher, une clé de ronde sera utilisée pour masquer (par *ou exclusif* bit-à-bit) le bloc de donnée à l'entrée de chaque ronde et la dernière clé de ronde sera utilisé pour masquer la sortie de la dernière ronde.
La génération de clé, à peine moins coûteuse que la fonction de chiffrement (8 rondes), est assez complexe pour rendre impossible (aussi loin que notre étude nous a amené) les techniques de cryptanalyse par clés corrélées.

Les constantes $\{c^i\}_{0\leq i\leq 8}$ sont générées à partir de la permutation P. P, définie plus bas, est un endomorphisme de $\mathbb{F}_2^8$, et
$\forall i \in \{0;\cdots,8\}$,

$$c^i = (P(8i)||P(8i+1)||P(8i+2)||P(8i+3)||P(8i+4)||P(8i+5)||P(8i+6)||P(8i+7))_{16}$$

où $P(x)$ est considéré comme un vecteur de 8 bits (octet). La fonction F_{c^i} est représentée sur la figure 7.2, pour tout $x \in \mathbb{F}_2^{64}, F_{c^i}(x) = T(P^8(x \oplus c^i))$. Soit $x \in \mathbb{F}_2^{64}$ dans sa représentation par octets $x = (x_0||\cdots||x_7)_{16}$, chacun des octets est un vecteur de 8 bits noté $\forall i \in \{0,\cdots,7\}, x_i = (x_i^0||\cdots||x_i^7)_2$, alors T est une permutation sur les bits de x :

$$T(x) = (x_0^0||x_1^0||\cdots||x_7^0||x_0^1||\cdots||x_0^7||\cdots||x_7^7)_2$$

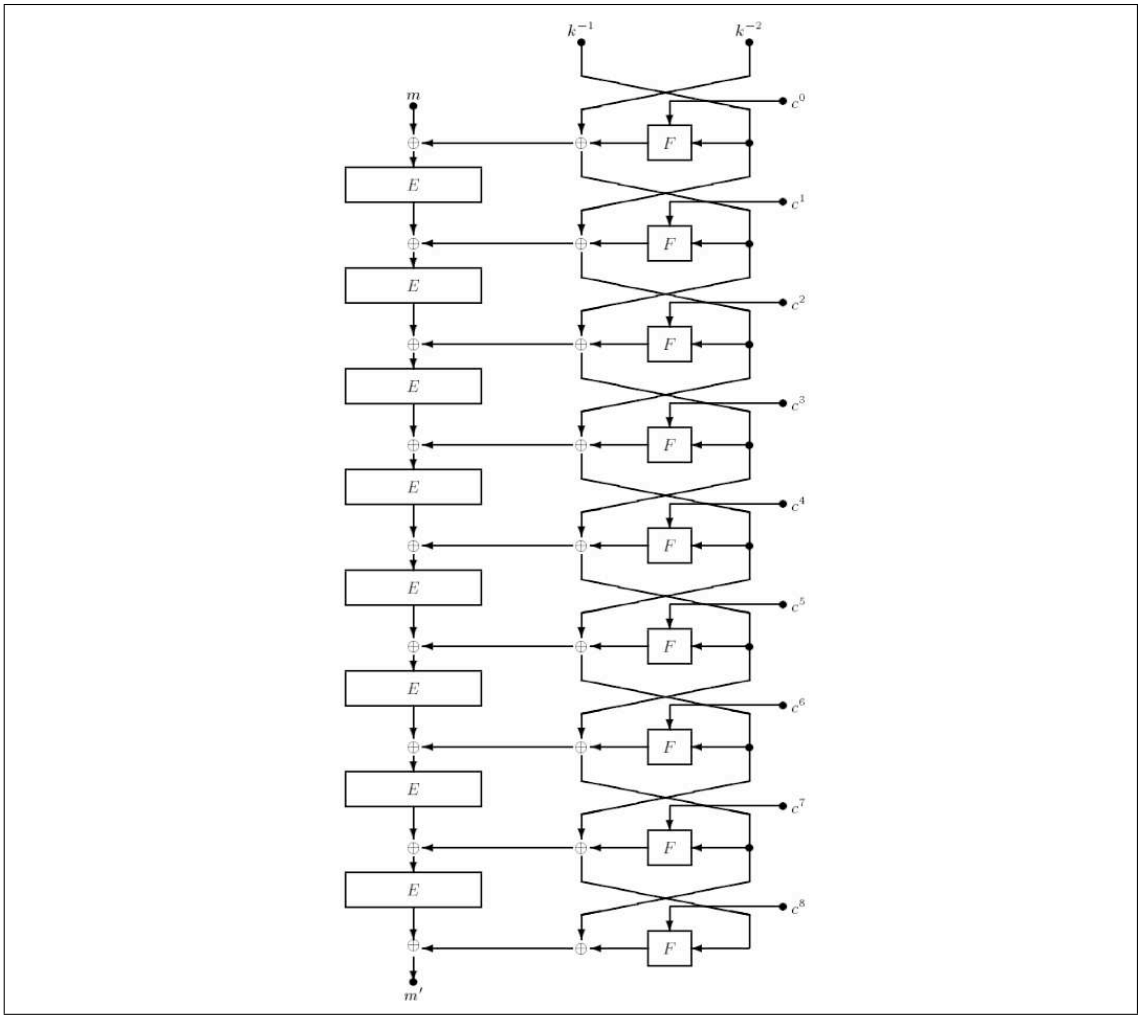

FIG. 7.1 – Schéma global de CS-Cipher (Tirée de [205])

Fonction de chiffrement Chaque ronde E étant composée de trois sous-rondes de structure PSN (pour "Permutation Substitution Network" en anglais), la figure 7.3 décrit une ronde de l'algorithme de chiffrement.

Les constantes c et c' sont les mêmes pour chaque ronde, elle sont générées à partir des décimales de e

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = 2, \texttt{b7e151628aed2a6abf7158809cf4f3c762e7160f} \cdots_{16}$$

La fonction M est définie par $\forall x \in \mathbb{F}_2^{16}, M(x) = P^2(\mu(x))$ (figure 7.4). telle que

– P, endomorphisme de $\mathbb{F}_2^8$, est la boîte de substitution de CS-Cipher, elle a été construite sur un réseau de Feistel à partir de deux endomorphismes de $\mathbb{F}_2^4$ (voir figure 7.5). Cette

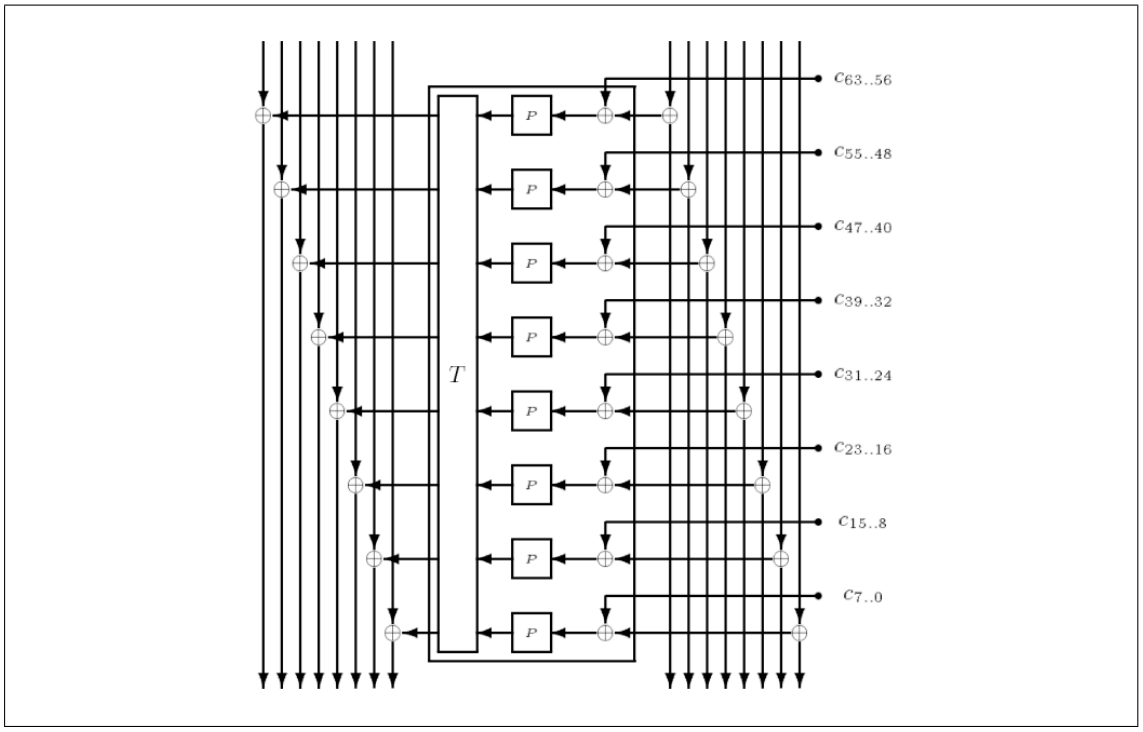

FIG. 7.2 – Fonction F_{c^i} dans la génération de clés de rondes (Tirée de [205])

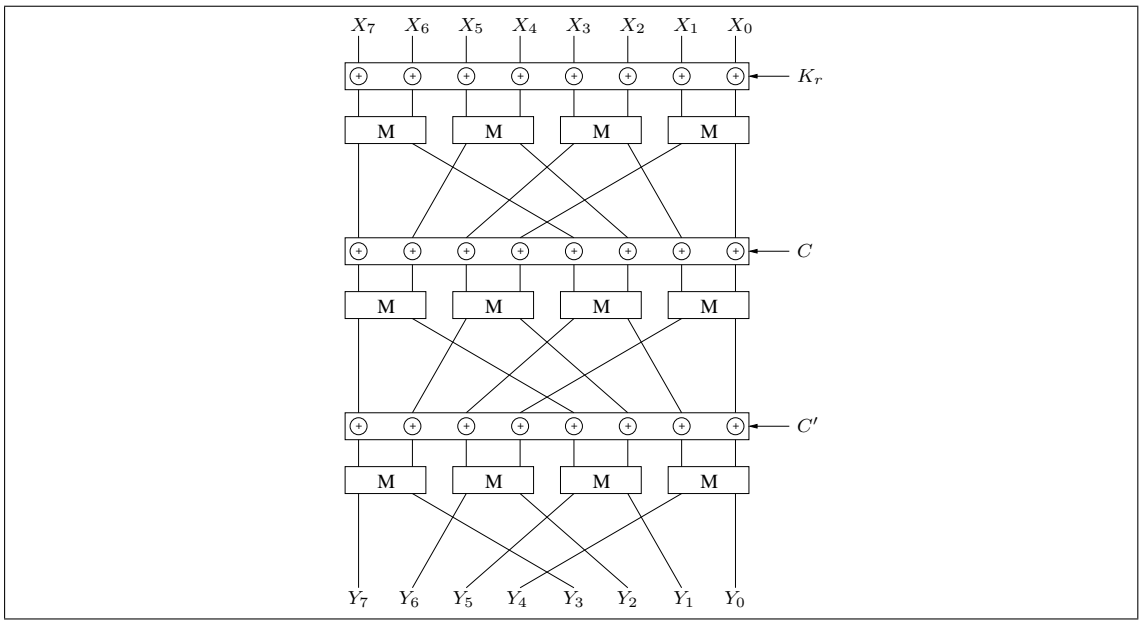

FIG. 7.3 – Fonction de Ronde E de CS-Cipher

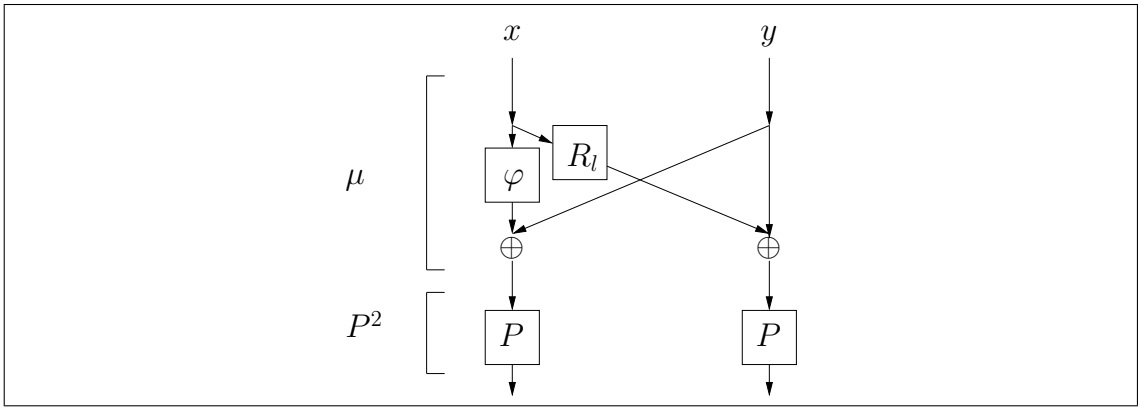

FIG. 7.4 – Fonction M de CS-Cipher

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$f(x)$	f	d	b	b	7	5	7	7	e	d	a	b	e	d	e	f

TAB. 7.2 – Fonction f

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$g(x)$	a	6	0	2	b	e	1	8	d	4	5	3	f	c	7	9

TAB. 7.3 – Fonction g

construction et le choix des fonctions f et g permet à la fois une implémentation matérielle efficace et une bonne non-linéarité à la fonction P (caractéristique essentielle d'une boîte-S). Les tableaux 7.2 et 7.3 décrivent les fonctions f et g respectivement, notons que f peut s'écrire aussi $f(x) = \overline{x} \wedge \overline{R_l(x)}$.

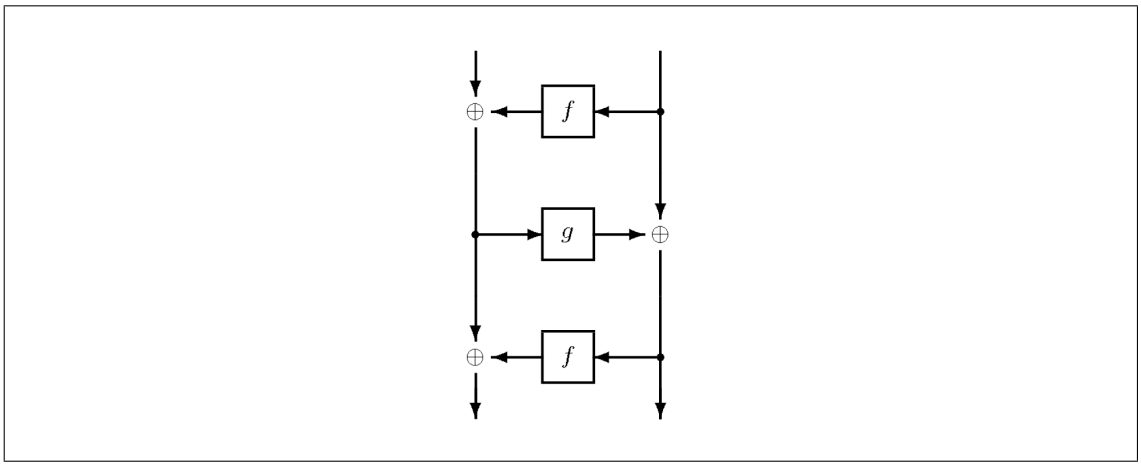

FIG. 7.5 – Boîte de Substitution de CS-Cipher (P)

– μ est une $(2,2)$-*multipermutation*, définie par :

$$\forall (a,b) \in (\mathbb{F}_2^8)^2, \mu(a,b) = (\varphi(a) \oplus b, R_l(a) \oplus b)$$

où φ est définie par $\varphi(x) = (R_l(x) \wedge 0x55) \oplus x$.
La notion de multipermutation a été introduite par Schnorr et Vaudenay [197] et est particulièrement intéressante dans la construction de chiffrements symétrique [217]. On la retrouve dans beaucoup de chiffres actuels, AES par exemple. Une $(2,2)$-multipermutation définie sur $\mathbb{F}_2^8$ peut être vue comme une permutation sur $\mathbb{F}_2^{16}$ telle que fixer la première (resp. seconde) moitié de l'entrée définit deux permutations sur $\mathbb{F}_2^8$ consistant aux deux moitiés de la sortie lorsqu'on fait varier la seconde (resp. première) moitié de l'entrée. Pour une présentation formelle des multipermutations, se référer à la thèse de doctorat de Vaudenay [198].

Dans toute la suite nous noterons $\mathscr{L}$ la fonction linéaire de diffusion suivant un schéma de transformée de Fourrier rapide, schéma dit en *ailes de papillon* (noté FFT dans la suite pour "Fast Fourrier Transform" en anglais). Pour tout $x = (x_0||\cdots||x_7)_{16}$,

$$\mathscr{L}(x) = (x_0, x_2, x_4, x_6, x_1, x_3, x_5, x_7)_{16}$$

Il s'agit de la dernière étape de chaque sous-ronde de CS-Cipher.

Remarque 32 *Nous avons dit que les sous rondes de CS-Cipher suivent une structure PSN, pour le voir il suffit de se rendre compte que les boîtes de permutations P peuvent être placées, sans changer la fonction de chiffrement, après la transformation linéaire $\mathscr{L}$.*

A la suite de l'article [205] définissant le chiffre CS-Cipher, Serge Vaudenay étudie la sécurité du chiffre par rapport à la cryptanalyse linéaire et différentielle dans [219]. Pour l'analyse de sécurité du chiffre (résistance aux attaques), l'auteur définit un chiffre, noté CSC^*, sur lequel l'étude est simplifiée. Dans un deuxième temps les résultats sur CSC^* sont étendus à CS-Cipher. Comme nous l'avons vu, les rondes de CS-Cipher sont elles-mêmes composées de 3 sous-rondes : l'entrée de la première sous-ronde étant masquée par une clé de ronde et les entrées des deuxièmes et troisièmes rondes étant masquées par les constantes c et c' définies plus haut. Les rondes de CSC^* sont directement les sous-rondes de CS-Cipher et comprend donc 24 rondes. Ainsi les constantes c et c' sont remplacées par des clés de rondes, le schéma 7.6 reprend la structure de CSC^*. Ce chiffre étant construit pour l'analyse de sécurité de la fonction de chiffrement et non de la génération de rondes, celle-ci est supprimée : chaque clé de ronde est supposée indépendante des autres et uniformément distribuée, hypothèse classique pour l'étude de résistance face aux cryptanalyses linéaires, différentielles et dérivées, cf. section 3.1.3.3.

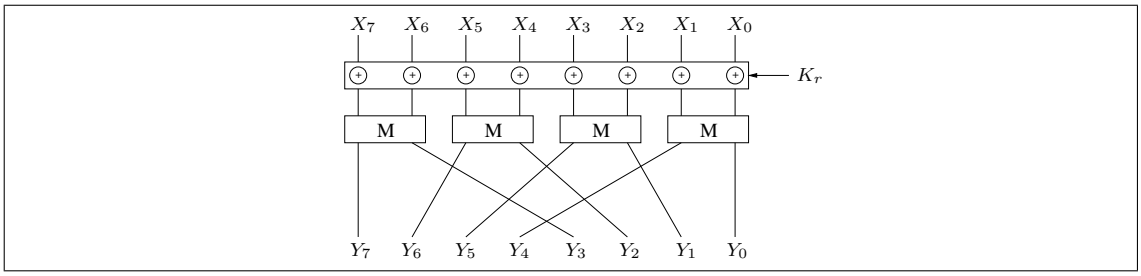

FIG. 7.6 – Fonction de Ronde de CSC^*

III

Tolérance aux fautes, certification de résultats

Tolérance aux Fautes 8

Sommaire

8.1 La tolérance aux fautes : Principes de base et Taxinomie 105
8.1.1 Classes de fautes 106
8.1.2 Classes de défaillances 106
8.1.3 Techniques de tolérance aux fautes 106
8.2 Tolérance aux fautes dans les systèmes distribués 108
8.2.1 Systèmes distribués 109
8.2.2 Modèles de Fautes 109
8.2.3 Techniques de tolérance aux fautes 110
8.2.4 Cas des systèmes distribués de grande taille 111

Dans ce chapitre sont introduits les grands principes de la tolérance aux fautes pour les systèmes distribués. Nous avons vu dans le chapitre 2, que la tolérance aux fautes est un des points essentiels de la *sûreté de fonctionnement*. Dans une première partie nous présenterons les notions nécessaires à l'étude de la tolérance aux fautes, notamment nous détaillerons les différents types de fautes ou défaillances qui peuvent menacer un système et les techniques de tolérance aux fautes visant à s'en prémunir. Une deuxième section examinera les modèles et caractéristiques particulières des systèmes distribués dans le contexte de tolérance aux fautes.

8.1 La tolérance aux fautes : Principes de base et Taxinomie

Avant de présenter les différentes techniques existantes de tolérance aux fautes pour les systèmes distribués, nous allons définir formellement les différents composants nécessaires à l'étude de la tolérance aux fautes en suivant la taxinomie proposée par Jean-Claude Laprie et al. [9, 12, 13].

Les techniques de *tolérance aux fautes* appliquées à un système rassemblent l'ensemble des moyens visant à éviter les *défaillances* du système. Une défaillance correspond à un comportement inacceptable du système, elle est donc définie par rapport à la description fonctionnelle du système et non par rapport aux spécifications qui peuvent elles-même comporter des erreurs. Un état erroné du système (simplement appelé *erreur*) est la cause d'une défaillance, i.e. l'état du système n'appartient plus à l'ensemble des états corrects. Une erreur est à son tour causée par

l'occurrence d'une *faute*. Le schéma 8.1, emprunté à [9], reprend la chaîne fondamentales de causalité des *entraves* à la sûreté de fonctionnement.

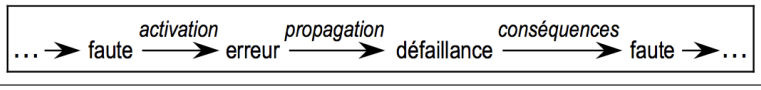

FIG. 8.1 – Chaîne fondamentale des entraves à la sûreté de fonctionnement ([9])

8.1.1 Classes de fautes

Les fautes menaçant un système sont catégorisées dans [9] suivant 7 critères complémentaires et non exclusifs :

Phase de création *Phase du cycle de vie du système où la faute apparaît.*

Frontières du système *Localisation de la faute, elle peut-être interne au système ou externe et se propager à l'intérieur du système.*

Cause Phénoménologique *Origine de la faute : humaine ou non.*

Dimension *La faute affecte le matériel ou le logiciel.*

Intention *Les fautes peuvent être malveillantes ou non.*

Capacité *Suivant que la faute provient d'un choix délibéré, d'un accident ou d'une incompétence.*

Persistance *Durée de vie de la faute.*

8.1.2 Classes de défaillances

De même que pour les fautes, le défaillances sont classées dans [9] suivant 4 critères complémentaires et non exclusifs :

Domaine *Défaillance en valeur : Le système rend un service incorrect. Défaillance temporelle : Le système ne délivre pas le service dans une durée convenable.*

Détectabilité *Le service incorrect est détecté comme tel ou non.*

Cohérence *L'apparition de la défaillance suit une loi déterministe ou non (auquel cas elle est appelée* byzantine*).*

Conséquences *Gravité de la défaillance.*

8.1.3 Techniques de tolérance aux fautes

Les techniques de tolérance aux fautes s'organisent autour de deux grandes opérations : la *détection d'erreurs*, qui a pour rôle de rechercher la présence d'une erreur à l'origine d'une défaillance, et le *rétablissement du système*, qui s'attache à reconstruire un état correct du système (*traitement de l'erreur*) et à éviter l'apparition de la faute dans le futur (*traitement de la faute*).

Détection d'erreur La détection d'erreur peut être faite en cours de fonctionnement du système (*détection concomitante*) ou lors de suspension de service (*détection préemptive*). Les principales techniques de détection concomitantes sont les suivantes :

- **Codes détecteurs d'erreur.** Rendre l'état d'un système redondant permet de détecter si il est erroné ou non, le nombre d'erreurs pouvant être détectées se calcule en fonction de la distance du code (voir section 3.2 pour une introduction aux codes correcteurs). La redondance peut-être matérielle ou temporelle. Le cas trivial de ce type de détection est le *doublement et comparaison* (spatial ou temporel). Bien que ce ne soit pas le cas en général pour la détection d'erreurs par codes, la solution dédoublement présente l'avantage d'être à la fois très simple et applicable à tous types d'application.
- **Tests de programmes durant l'exécution.** Cette technique permet de vérifier l'exécution correct d'un programme (ou sous-programme) du système. Ainsi le *testeur* de programme, introduit par Boyer et Moore en 1981 [44], consiste simplement à vérifier qu'un programme se comporte bien par rapport à un ensemble d'entrées prédéfinies en comparant les sorties aux sorties pré-calculées. Ce schéma, très simple à réaliser, ne permet pas de détecter les erreurs qui n'apparaissent que pour un sous-ensemble d'entrées du programme, par contre il s'applique à tous types de programme.
- **Vérification de résultats.** La vérification de résultats, introduit par Blum et Kannan en 1989 [37] consiste à construire un programme *vérifieur* (*simple checker* en anglais) capable de certifier, efficacement et avec une probabilité d'erreur faible, la validité d'un résultat. Le résultat du programme est vérifié en utilisant une post-condition (condition assurée par le résultat lorsqu'il est correct). Cette technique, plus robuste que le test de programme, est plus contraignante puisqu'un vérifieur doit être développé pour chaque programme en fonction de sa post-condition. De plus, tout les programmes ne jouissent pas nécessairement de post-conditions efficaces (i.e., dont le coût de calcul est faible par rapport à l'exécution du programme initial). Suivant la même idée, Ergün et al.[75] proposent le *spot-checking*, les vérifieurs sont plus efficaces (plus rapides) mais ont une probabilité de se tromper plus grande.
- **Contrôles temporels et d'exécution.** Ces techniques nécessitent l'utilisation de programmes "chien de garde" (*watchdog* en anglais). Ces programmes ont pour rôle de vérifier, durant l'exécution, que certaines contraintes temporelles et sur le flot d'instructions exécutées par le système sont respectées. Largement utilisée, cette méthode est applicable à tout type d'applications.
- **Contrôles de vraisemblance.** Vérifications que les données respectent des caractéristiques intrinsèques aux fonctionnalités du système.
- **Contrôles de données structurées.** Vérifications de l'intégrité sémantique ou structurelle des données durant l'exécution.

Rétablissement du système Le rétablissement du système consiste à reconstruire un état correct du système, i.e., corriger l'erreur. Trois grandes familles de rétablissement de système sont utilisées :

- **La Reprise**. Cette première technique, très largement utilisée, consiste à sauvegarder régulièrement l'état du système sur une *mémoire stable*, c'est à dire une zone de stockage qui assure l'intégrité et la disponibilité de l'information. En cas de signal de détection d'erreur,

le système est réinitialisé avec une sauvegarde antérieure d'un état correct. Cette technique, très générique, s'adapte à tous types d'application (lorsque le temps de sauvegarde n'en compromet pas les contraintes temps-réel) mais impose un certain nombre de contraintes sur l'architecture du système. De plus, il est important de savoir détecter les erreurs rapidement pour ne pas sauvegarder d'états corrompus.

- **La Poursuite**. Les techniques de poursuite consistent à récupérer un état acceptable si ce n'est correct du système et à continuer l'exécution sur cet état imparfait mais satisfaisant les critères essentiels (en terme de sécurité par exemple) du système. L'exécution se fera alors de façon dégradée mais garantira la terminaison dans un état sûr. Il est à noter que cette solution est dépendante de l'application concernée, elle ne peut être appliquée de façon générique comme l'est la reprise ou certaines formes de compensation.
- **La compensation**. La compensation consiste à avoir un état du système suffisamment redondant pour pouvoir corriger les erreurs éventuelles. Voici les principales techniques de compensation :
  - *Détection d'erreurs et compensation.* Par exemple sur la base de duplication de composants auto-testables. Un composant vérifie régulièrement son exécution (par exemple par les techniques exposées dans le paragraphe précédent). Lorsque qu'une erreur dans son exécution est avérée, il est déconnecté et la main est passée à un de ses clones.
  - *Codes correcteurs d'erreurs.* De même que pour la détection d'erreur par codes détecteurs d'erreurs, les codes correcteurs d'erreurs permettent de corriger les erreurs lorsque l'état du système est suffisamment redondant (voir section 3.2). Il faudra une distance de code plus grande que pour la détection. Par exemple, le cas trivial de code correcteur est le vote majoritaire entre copies d'un même composant, il nécessite trois copies pour corriger une erreur alors qu'il en fallait seulement deux pour la détecter (cf *dédoublement et comparaison* pour la détection d'erreurs). Un exemple classique de ce type de réplication est le TMR ("Triple Modular Redundancy" [11]).
    Parmi ces solutions, notons aussi les *ABFT* (pour "Algorithm-based fault tolerance" en anglais) qui seront étudiés dans le chapitre suivant.
  - *Programmes auto-correcteurs.* Dans la lignée de la *vérification de résultats* pour la détection d'erreurs, Lipton [141], puis Blum, Luby et Rubinfield [38] proposent les *programmes auto-correcteurs*. Ils correspondent à une construction de "wrappers", autour du programme initial P non-tolérant aux fautes, permettant d'assurer une tolérance aux fautes par appels multiples de P.

Maintenant que les bases de la tolérance aux fautes ont été établies, nous allons nous porter sur son étude dans les systèmes répartis ou distribués, cible préférée des travaux de recherche présentés ici.

8.2 Tolérance aux fautes dans les systèmes distribués

Nous commencerons par définir le modèle d'exécution d'un système distribué. Sur la base de ce modèle, nous verrons comment les techniques présentées dans la section précédente sont appliquées.

8.2.1 Systèmes distribués

Les systèmes distribués ou répartis sont, conformément à la définition donnée au chapitre 2, tout les systèmes composés d'un ensemble de *processus* communiquants par *messages* aux moyens de *canaux*. Tout comme dans [9], nous ferons l'hypothèse que les défaillances des processus sont indépendantes. Cette hypothèse permet de simplifier l'étude de la tolérance aux fautes, elle peut être invalidée si, par exemple, deux processus s'exécutent sur la même machine défectueuse ou corrompue. Dans ce cas il suffit de regrouper les processus s'exécutant sur la même machine pour retrouver le modèle de départ. Ce modèle est habituellement appelé le *modèle logique* du système distribué, par opposition au *modèle physique* qui est constitué des *processeurs* et canaux de communications inter-processeurs.

La question du *modèle temporel* est fondamentale dans les systèmes distribués, elle l'est aussi dans le domaine de la tolérance aux fautes. Ainsi, la détection d'un arrêt de processus se fait généralement par mesure du temps de réponse : au delà d'une certaine durée, si le processus n'a pas rendu son service, il est considéré comme arrêté et ses communications coupées. L'étude de systèmes complètement asynchrones, (sans aucune notion de temps) ne sera pas développée ici, du fait de leur incapacité à décider si un processus est arrêté ou non, la tolérance aux fautes pour ce type de système est bien plus ardue que dans le cas synchrone. Notons, par exemple que le *consensus* entre processus est impossible dans de tels systèmes (Fischer, Lynch, Paterson [76]). Le consensus entre processus consiste à faire le choix d'une valeur commune entre tout les processus corrects, c'est un des problèmes fondamentaux de la tolérance aux fautes dans les systèmes distribués.
Notons enfin que les modèles classiques supposent que les canaux de communication sont fiables. Cette supposition se justifie en pratique par l'utilisation de protocoles de communication faisant appel à des codes correcteurs pour transmettre les données et à des messages de validation ("acknowledgement" en anglais).

Ce modèle présente plusieurs avantages significatifs pour l'étude de la tolérance aux fautes, notamment de construire un modèle de faute au niveau des processus : les processus sont étudiés en *boîtes noires*.

8.2.2 Modèles de Fautes

Nous définirons 5 types de défaillances au niveau processus, pour chacun d'entre eux, la défaillance peut être temporaire ou permanente :

Arrêt Souvent appelé *panne franche* dans la littérature (*fail stop* en anglais). C'est le cas le plus simple, et lorsque c'est possible on essaie de s'y ramener (par exemple en "tuant" un processus pour lequel une erreur a été détectée).

Omission Le système perd des messages entrant ou sortant d'un processus. Cette défaillance n'est pas prise en compte dans notre modèle de canaux de communication fiable.

Temporelle Le processus ne répond pas dans les limites de temps décrites dans les spécifications du système.

Valeurs Le processus renvoie une donnée incorrecte.

Arbitraire Souvent appelée *byzantine*. Le processus peut faire n'importe quoi, de façon incohérente. Cette approche permet souvent de modéliser un comportement malveillant et le "pire cas". L'intérêt est d'étudier les conditions les plus défavorables, ainsi une technique de tolérance aux fautes pour le modèle byzantin sera valide pour tout les autres modèles de défaillance.

8.2.3 Techniques de tolérance aux fautes

Il n'est pas possible de construire une méthode de tolérance aux fautes valable pour tout cas d'étude, elle sera forcément dépendante des paramètres spécifiques du système ainsi que de son environnement d'exécution. L'étude du contexte d'utilisation et du système (matériel et logiciel) permet de choisir un modèle de faute ainsi qu'un modèle d'occurrence de fautes : Combien de processus peuvent être défectueux ou corrompus, à quelle fréquence (pour des défaillances temporaires) et avec quel type de panne (modèle de fautes).

Modèle par Arrêt La plus grande majorité des solutions de tolérance aux fautes pour les systèmes distribués a été proposée pour le modèle de faute "Arrêt". Elles se répartissent en 3 catégories :

Par Mémoire Stable Nécessite l'utilisation d'une *mémoire stable*. Rappelons qu'une mémoire stable est un serveur de stockage de données préservant l'intégrité et la disponibilité des données de façon sûre. Elle permet de sauvegarder régulièrement l'état du système et de restituer un état global cohérent lorsqu'une erreur a été détectée (pour plus de détails sur ces techniques voir [73]). Dans la littérature deux formes de reprise par mémoire stable sont proposées : La *reprise par journalisation* et la *reprise par sauvegarde*.

Par Réplication de processus La réplication consiste à dupliquer les processus en plusieurs copies identiques qui s'exécuteront sur plusieurs processeurs. Il en existe trois formes principales [226]. Il suffit de $N + 1$ copies de processus pour tolérer N pannes par arrêt.

- *Réplication active.* Chaque copie exécute le même traitement, le résultat final est donné par la première copie qui termine le travail.
- *Réplication passive.* Une copie est désignée pour être le leader, elle seule effectuera le traitement. En cas d'arrêt de la copie leader, une autre copie prend sa place.
- *Réplication semi-active.* Combinaison des deux stratégies précédentes : toutes les copies effectuent le traitement, mais seule la copie leader renvoie le résultat. En cas d'arrêt de la copie leader, une copie secondaire prend sa place.

Par Réplication de données La réplication de données a un rôle complémentaire à la réplication de processus, elle est utilisée pour assurer à la fois l'accessibilité des données lorsque certaines données sont possédées par des processus ou processeurs défaillants et la performance en lecture des données puisqu'il est généralement plus efficace de communiquer avec un processeur proche. Cette technique est très largement utilisée depuis les systèmes multi-processeurs jusqu'aux larges réseaux pair-à-pair.

Modèle par Valeur Le modèle de défaillance par valeur a été moins étudié, ces défaillances ne concernaient jusqu'à récemment que quelques cas exceptionnels de calcul dans l'espace ou

haute atmosphère (du fait de rayons cosmiques par exemple). Les architectures modernes utilisent des composants de taille si réduite et en telle quantité que les fautes par valeurs ne sont plus négligeables. Par exemple BlueGene/L d'IBM comportant, 128000 noeuds, doit récupérer une erreur par valeur dans son cache toutes les 4-6 heures en moyenne [229].

Remarque 33 *Notons que les techniques de tolérance aux fautes pour le modèle "Arrêt" s'appliquent directement au modèle de défaillance par valeur si l'on suppose que les processus possèdent des méthodes de détection d'erreurs. En effet, ils seront alors supposés arrêtés lorsqu'une erreur est détectée. De même les techniques pour le modèle de défaillance par valeur s'adaptent de façon transparente au modèle de défaillance par arrêt, il suffit de considérer qu'un processus arrêté renvoie des résultats aléatoire.*

Les méthodes de réplication de processus suivi de vote par majorité sont certainement les plus utilisées pour les systèmes critiques (dans l'aéronautique par exemple). L'avantage certain de ces techniques est qu'elles sont indépendantes de l'application cible tout en assurant un temps d'exécution indépendant de l'occurrence d'une faute (contrairement aux techniques de sauvegarde par exemple), caractéristique particulièrement appréciée pour les systèmes sous contraintes temps-réel. Le grand désavantage de la réplication est le coût de l'implémentation : il faut doubler le nombre de processeurs pour détecter une erreur, tripler pour corriger une erreur.

Pour les machines multi-processeurs, la reprise par sauvegarde est aussi très utilisée, elle est assez efficace à mettre en oeuvre de par le caractère homogène de la machine (composée de N processeurs identiques) qui permet d'assurer une sauvegarde cohérente de façon simple [50, 130, 74]. Néanmoins, lorsque les tâches du calcul manipulent des données trop volumineuses, le coût des sauvegardes peut devenir prohibitif.
Pour ce type de machines, les ABFTs ont été largement étudiés et de nombreuses applications ont été développées sous ce principe de tolérance aux fautes par code correcteur. Nous verrons ces méthodes plus en détail dans le chapitre 9 et étudierons leur utilisation dans les systèmes distribués de grande taille.

8.2.4 Cas des systèmes distribués de grande taille

Nous avons vu dans le chapitre 2 que nous entendons par systèmes distribués de grande taille des systèmes de types *réseaux de machines* (*Network Of Workstations* en anglais). Ces systèmes, peuvent atteindre de très grandes tailles en nombre de machines et comporter des machines très différentes (au niveau matériel et logiciel), des exemples ont été proposés dans le chapitre 2.

La taille et le contexte d'utilisation de ces systèmes (systèmes ouverts) rendent à la fois les techniques de tolérance aux fautes primordiales et très difficiles à mettre en oeuvre. La plupart des techniques utilisées en pratique reposent sur la réplication. Notons tout de même :

Reprise par sauvegarde Pour les grilles en comportement non malicieux [104, 23]

Challenges Pour les comportements malicieux sur grilles ouvertes ou réseaux pairs-à-pairs, l'idée est d'utiliser des techniques de *vérification de résultat* ou *spot-checking* pour évaluer les noeuds du réseaux. Lorsqu'un noeud défectueux ou corrompu est démasqué, il peut être déconnecté (*blacklisting* en anglais) [225, 194].

Certification de résultats en présence d'attaques massives Pour les comportements malicieux en grand nombre, Varrette et al. [216, 186] proposent de les détecter en réexécutant aléatoirement certaines tâches de l'algorithme ; ce qui permet de certifier que le nombre de fautes n'est pas trop grand. Une technique de tolérance aux fautes interne à l'algorithme permet alors de les corriger et donc de certifier le résultat avec une probabilité d'erreur quantifiée.

Réplication et vote Dans le cas de comportements malicieux sur réseaux pair-à-pair, de nombreuses recherches se sont intéressées aux problèmes classiques du *consensus* ou *accord byzantin* [125, 113]. La résolution de ces problèmes permet de construire des techniques de tolérance aux fautes par réplication et vote de majorité.

Algorithmes auto-tolérants aux fautes sur système de calcul global. Application au produit de matrices 9

Sommaire

9.1 Notations et premières définitions 114
9.1.1 Coût de calcul sur plate-forme parallèle 114
9.1.2 Fonction d'impact et capacité de tolérance aux fautes 116
9.2 Tolérance aux fautes dans les systèmes de calcul globaux 117
9.2.1 Description générale du système de calcul global et analyse de la problématique 117
9.2.1.1 Modèles de fautes 118
9.2.1.2 Certification de résultat 119
9.2.1.3 Version itérative du produit de matrices 121
9.2.2 Schéma général du calcul tolérant aux fautes 121
9.2.2.1 Algorithmes auto-tolérants aux fautes 122
9.2.2.2 Schéma générique 123
9.2.3 Choix de codes correcteurs et influence sur R 125
9.2.3.1 Réplication pure 125
9.2.3.2 Reed-Solomon et codes par interpolation 126
9.2.3.3 Low Density Parity Check Codes 127
9.2.3.4 Synthèse 128
9.3 Application au produit de matrices et premiers résultats 130
9.3.1 Algorithmes auto-tolérants aux fautes 130
9.3.2 Utilisation de codes systématiques ou creux et conséquence sur U . . . 133
9.4 Produit de matrices et codes linéaires (ABFT) 134
9.4.1 ABFT : Définition et état de l'art 134
9.4.1.1 Applications 135
9.4.1.2 Méthodes 136
9.4.2 Application au produit de matrices 137
9.5 Produit de matrices et codes par résidus 138
9.5.1 Schéma général et état de l'art 139
9.5.1.1 RNS et PRNS 139

9.5.1.2 RRNS et RPRNS 140
9.5.1.3 Algorithmes efficaces de codage et décodage des codes RRNS et RPRNS 141
9.5.2 Codes par résidus pour le produit de matrices sur des systèmes de calcul globaux 142
9.6 Synthèse des résultats 144
9.6.1 Récapitulation des résultats 144
9.6.2 Application de l'algorithme ABFT pour le produit de matrices sur réseau pair-à-pair 144

Ce chapitre est consacré à une méthode de tolérance aux fautes sur systèmes distribués de grandes tailles dédiés au calcul (appelés *systèmes de calcul globaux* dans la suite). Il s'agit d'étudier la construction d'algorithmes auto-tolérants aux fautes et leur capacité à détecter et corriger les erreurs sous un modèle de faute adapté aux systèmes que nous étudions. Cette étude sera appliquée au calcul du produit de matrices de grandes dimensions, un cas d'étude classique qui trouve de nombreuses applications dans les calculs scientifiques.
Dans une première section nous présenterons un modèle pour les systèmes de calcul globaux et les modèles de fautes étudiés. Nous définirons alors les algorithmes auto-tolérants aux fautes et proposerons un algorithme générique de tolérance aux fautes avec certification de résultats. Dans les trois sections suivantes nous utiliserons plusieurs méthodes pour améliorer le coût des ces algorithmes dans le cas du produit de matrices et obtenir des algorithmes auto-tolérants efficaces. Enfin nous présenterons comment un des algorithmes pourrait être mis en place sur une plate-forme de calcul pair-à-pair.

9.1 Notations et premières définitions

Il est question dans ce chapitre de comparaison d'algorithmes tolérants aux fautes sur plate-formes parallèles de calcul, afin de rendre le discours cohérent nous présentons dans cette section les hypothèses sous lesquels ces comparaisons sont faites et la caractérisation du coût d'un algorithme.

9.1.1 Coût de calcul sur plate-forme parallèle

Comparaison d'algorithmes Soit un algorithme $\mathscr{A}$ à exécuter sur une plate-forme I composée de p processeurs et ayant une vitesse de calcul totale Π_I (nombre d'instructions exécutées par seconde). Nous noterons :

- W le *travail* total de $\mathscr{A}$: le nombre total d'instructions de $\mathscr{A}$.
- D la *profondeur* de $\mathscr{A}$: le nombre maximal d'instructions en relation de précédente, i.e. chemin critique de l'algorithme parallèle sur un nombre infini de processeurs.

D'après le théorème 6 de [21], le temps d'exécution de $\mathscr{A}$ sur I, satisfait, *avec une grande probabilité* :

$$T_I \leq \frac{1}{\Pi_I}(W + O(pD))$$

Dans toute la suite, nous ne voudrons généralement pas faire d'hypothèses sur le nombre de machines que comporte le système de calcul global et nous seront principalement intéressés par une comparaison asymptotique des algorithmes. Nous comparerons donc deux algorithmes $\mathscr{A}$ et $\mathscr{A}'$ suivant leur coût respectif $\mathfrak{C}_{\mathscr{A}}$ et $\mathfrak{C}_{\mathscr{A}'}$ de la façon suivante :

$$\mathfrak{C}_{\mathscr{A}} \leq \mathfrak{C}_{\mathscr{A}'} \quad \text{ssi} \quad \left\{ \begin{array}{rcl} W_{\mathscr{A}} & = & O(W_{\mathscr{A}'}) \\ D_{\mathscr{A}} & = & O(D_{\mathscr{A}'}) \end{array} \right.$$

Ainsi, $\leq$ est une relation d'ordre partiel, deux algorithmes $\mathscr{A}$ et $\mathscr{A}'$ ne seront pas comparables si, par exemple,

$$\begin{array}{rcl} W_{\mathscr{A}'} & = & o(W_{\mathscr{A}}) \\ D_{\mathscr{A}} & = & o(D_{\mathscr{A}'}) \end{array}$$

Convention de complexité Dans toute la suite nous considérerons des applications composées d'opérations arithmétiques dans un ensemble E fini à q éléments. Nous considérerons que les éléments de E peuvent s'écrire sur $\lceil \log q \rceil$ bits. Dans le cas le plus général, nous supposerons que les opérations arithmétiques sur des valeurs dans E nécessitent un travail $W_{op} = \omega_q$ et une profondeur $D_{op} = \delta_q$. Les valeurs de ω_q et δ_q sont dépendantes de beaucoup de paramètres, suivant la structure de l'ensemble E, sa taille, et l'opération dont il est effectivement question (addition, multiplication, inverse, ...). On pourra trouver en annexe C une récapitulation des valeurs effectives de ω_q et δ_q, que l'on peut trouver dans le littérature, en fonction des différents cas. De plus nous ferons l'hypothèse suivante pour tout algorithme : un algorithme $\mathscr{A}$ composé de N opérations arithmétiques dans l'ensemble E et une profondeur de M opérations aura une complexité

$$\begin{array}{rcl} W_{\mathscr{A}} & = & N\omega_q \\ D_{\mathscr{A}} & = & M\delta_q \end{array}$$

Remarque 34 *A titre d'exemple et parce qu'il sera particulièrement intéressant dans ce chapitre, dans le corps de Galois à q éléments ($q = p^m$), nous pourrons supposer que $\omega_q = \widetilde{O}(\log q)$ et $\delta_q = \widetilde{O}(\log\log q)$. On Rappelle que $\widetilde{O}(x) = O(x \log^{O(1)} x)$.*

Remarque 35 *Nous imposons à E d'être fini pour généraliser l'écriture (tailles de stockage, complexités) et souligner le fait que la complexité d'une opération sur des éléments de E est dépendante de leur taille : en fixant une taille à E, nous imposons aussi une taille maximale aux éléments de E ($\log q$ bits). Ceci permet de choisir systématiquement une complexité aux opérations dans E (ω_q et δ_q), dépendante, d'une façon ou d'une autre, de q. Dans le cas où les opérations se font dans un ensemble infini, on peut considérer que E en est un sous-ensemble fini contenant, au moins, les éléments manipulés dans la totalité du calcul.*

9.1.2 Fonction d'impact et capacité de tolérance aux fautes

La tolérance aux fautes sur systèmes distribués a été présentée dans le chapitre précédent, nous avons notamment fait l'hypothèse de nous restreindre à l'étude des défaillances au niveau processus (*modèle logique*) au lieu de faire une étude, très dépendante de la plate-forme de calcul, au niveau processeurs (*modèle physique*).

Soit un algorithme $\mathscr{A}$ composé de T tâches, ayant une donnée d'entrée x et renvoyant une donnée de sortie y. Nous pouvons toujours écrire $y = (y_1, \cdots, y_k) \in E^k$, pour un choix de l'ensemble E adapté, par exemple : $E = \{0; 1\}$ et $(y_1, \cdots, y_k)$ est la représentation binaire de y ou encore $E = \{\text{sorties possibles de } \mathscr{A}\}$ et $k = 1$. L'étude de la tolérance aux fautes de $\mathscr{A}$ nécessite de savoir évaluer l'influence sur y d'une tâche défaillante de $\mathscr{A}$. Nous définissons la notion d'*impact* pour un algorithme $\mathscr{A}$ par rapport à une représentation de la sortie $y = (y_1, \cdots, y_k)$.

Définition 22 (Impact de fautes) *L'impact sur $\mathscr{A}$ de t tâches défaillantes (noté dans la suite $Impact(\mathscr{A}, t)$) est le nombre maximal de composantes fausses de la sortie $y = (y_1, \cdots, y_k)$ de $\mathscr{A}$ lorsque t tâches de $\mathscr{A}$ sont affectées par une faute. Nous dirons d'un algorithme $\mathscr{A}$, tel que $Impact(\mathscr{A}, 1) = s$, qu'il est "d'impact s".*

Soient deux algorithmes $\mathscr{A}$ et $\mathscr{A}'$ pour un même calcul, $\mathscr{A}$ tolère plus de fautes que $\mathscr{A}'$ si il existe $t > 0$ tel que $Impact(\mathscr{A}, t) = 0$ et $Impact(\mathscr{A}', t) > 0$.

L'impact de fautes est directement lié au graphe des dépendances entre les tâches d'un algorithme $\mathscr{A}$, il donne une sorte de mesure de la capacité d'un algorithme à tolérer les pannes.

Illustrons cette idée par l'exemple du produit de matrices. Il existe de multiples algorithmes pour calculer le produit de deux matrices carrées de dimensions k^2 : $C = A \times B$. Soit $\mathscr{A}$ l'algorithme naïf qui calcule de façon indépendante chaque composante de la matrice résultat ($C_{i,j} = \sum_{l=1}^{k} A_{i,l} B_{l,j}$) ($W_{\mathscr{A}} = O(k^3)$). Soit $\mathscr{A}'$, l'algorithme de Strassen ($W'_{\mathscr{A}} = O(k^{2.807})$) qui optimise le calcul en supprimant une multiplication dans le calcul d'une matrice de 2×2 blocs. Considérons la matrice résultat C comme un élément de E^{k^2} où E est l'ensemble de définition des matrices.
Il est facile de voir que $\mathscr{A}$ est d'impact 1 ($Impact(\mathscr{A}, 1) = 1$), une tâche a donc un faible impact sur le résultat C. Par contre, l'algorithme de Strassen introduit beaucoup de dépendance entre les tâches, si une erreur apparaît tout au début du calcul, elle peut se propager sur l'ensemble du résultat, on a alors $Impact(\mathscr{A}, 1) = k^2$. [A Expliciter]

Restriction Dans la suite du chapitre, de façon à clarifier l'étude, nous nous contenterons de travailler sur des algorithmes auto-tolérants aux fautes d'impact 1. Cela correspond à des algorithmes que l'on peut décomposer en k groupes entièrement indépendants de tâches $(f_1, \cdots, f_k)$. De telle façon que chaque composante de la sortie $(y_1, \cdots, y_k)$, est calculée de façon indépendante par un groupe de tâche.

Remarque 36 *Cette restriction, peut être relaxée dans le cal général : l'étude de tolérance aux fautes que nous allons voir peut se généraliser lorsque $Impact(A, 1) < k$.*

D'autre part, les systèmes que nous étudions (e.g. systèmes pair-à-pair) ne permettent généralement pas, à l'heure actuelle, d'exécuter des algorithmes avec de fortes dépendances entre les tâches et donc un ordonnancement complexe. Les applications sont donc souvent décomposables en tâches indépendantes.

9.2 Tolérance aux fautes dans les systèmes de calcul globaux

9.2.1 Description générale du système de calcul global et analyse de la problématique

Rappelons le problème que nous proposons de résoudre : comment améliorer la confiance dans le résultat d'un programme s'exécutant sur une plate-forme ouverte de calcul distribué de grande taille telle qu'une grille ou un réseau pair-à-pair ?

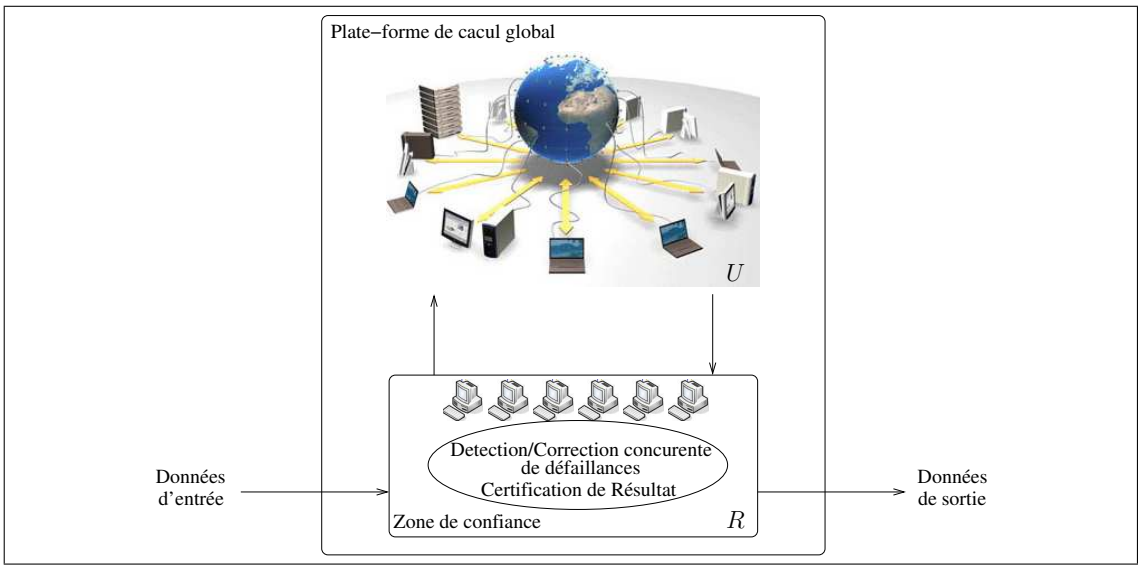

FIG. 9.1 – Calcul sur système distribué de grande taille

La figure 9.1 décrit très généralement système de calcul global et introduit la notion de *zones de confiance* à l'intérieur même du système. En effet, notre étude de la confiance dans le résultat est conduite sous l'hypothèse qu'une sous-partie du système cible peut être considérée comme fiable : les calculs exécutés et les données stockées sont corrects. Nous pourrons par exemple considérer qu'il s'agit de machines entièrement contrôlées par l'utilisateur du système et pour lesquelles ont été mis en oeuvre les différents moyens de la sûreté de fonctionnement. Une telle infrastructure fiable sera vraisemblablement très coûteuse (financièrement) et ne peut constituer une part significative du système de calcul global. Dorénavant, nous la noterons R (pour "Reliable") et lui associerons une puissance de calcul Π_R, de même nous noterons U (pour "Unreliable") le reste du système et Π_U sa puissance de calcul.
Il s'agit donc de maximiser la confiance dans le résultat en minimisant le surcoût dû à la tolérance aux fautes. Avec la définition du coût d'un algorithme donnée dans la section précédente, nous

considérerons le problème de la façon suivante : un algorithme tolérant les fautes $\mathscr{A}_F$ sur un système de calcul global $\{U, R\}$ a un vecteur de coût $(\mathfrak{C}_U, \mathfrak{C}_R)$ où $\mathfrak{C}_U$ (resp. $\mathfrak{C}_R$) est le coût de $\mathscr{A}_F$ sur les ressources U (resp. R). L'algorithme tolérant aux fautes $\mathscr{A}_F$ est optimal pour le système $\{U, R\}$ si il n'existe pas un autre algorithme, tolérant le même nombre de fautes, de vecteur de coût $(\mathfrak{C}'_U, \mathfrak{C}'_R)$, tel que $\mathfrak{C}'_U \leq \mathfrak{C}_U$ et $\mathfrak{C}'_R \leq \mathfrak{C}_R$.

Remarque 37 *L'hypothèse qu'une partie du système est digne de confiance, ou qu'une partie des calculs est exempt d'erreur est en réalité indispensable pour étudier le problème de la tolérance aux fautes. Car sans la supposition que les données d'entrée de l'algorithme sont correctes et accessibles et que les données de sortie sont correctement reçues, il est toujours possible de construire un résultat faux indétectable avec une unique erreur.*

Nous avons vu dans le chapitre 2 quelques exemples de tels systèmes de calcul globaux : grilles de calcul, "grilles de bureau" et les systèmes pair-à-pair.

Bien que très différents au niveau de l'architecture, ces systèmes peuvent se confondre autour d'un modèle de fautes particulièrement contraignant. Ainsi, que l'on considère une grille de calcul ou un réseau pair-à-pair, les noeuds du système sont administrés par différentes personnes, les droits d'un utilisateur sur les noeuds sont différents et souvent très limités. Lorsqu'un utilisateur soumet un calcul sur un tel système, la confiance qu'il a dans certains noeuds (ou groupes de noeuds) est très faible : e.g. les logiciels peuvent ne pas être à jour et posséder des failles de sécurité, certaines machines peuvent être en mauvais état ou soumises à des intempéries, l'administrateur même d'un noeud peut avoir comme but de fausser le calcul pour des raisons diverses (voir chapitre 2).

D'autre part, les solutions existantes (voir section 8.2.4 du chapitre précédent) présentent un certain nombre de désavantages : La *réplication* est coûteuse et corrige un taux d'erreurs fixe. La *reprise par sauvegarde* est difficile à mettre en place dans le cas de processus s'exécutant sur des machines hétérogènes et entraînera une surcharge des canaux de communication pour des applications travaillant sur de grandes tailles de données. Enfin les techniques de *challenges* sont inefficaces contre des modèles d'attaques organisées où les noeuds malicieux ne forgent des résultats erronés qu'aux moments opportuns (i.e. en fonction de la politique de challenge choisie).

9.2.1.1 Modèles de fautes

Les modèles de fautes sont de même nature que ce qui a été vu dans le chapitre précédent pour les systèmes distribués (voir section 8.2.2). Nous considérerons deux cas : les fautes *aléatoires* et les fautes *malicieuses*.

– Dans le cas de fautes *aléatoires* la probabilité d'occurrence d'une faute est indépendante de l'application et des données et suit une distribution fixée sur l'ensemble du système. Dans le cas le plus général la distribution est uniforme, mais elle peut être plus précise, dépendante de l'architecture du système ou calculée à partir d'études statistiques sur le système en fonctionnement. Par exemple, dans un réseau pair à pair, la *durée de vie moyenne* d'un pair est une donnée souvent utilisée pour l'étude et la caractérisation de tels réseaux : il s'agit d'évaluer le temps moyen qu'un utilisateur passe sur le réseau ; on estime ainsi, la

probabilité qu'un pair donné se déconnecte à un instant donné sachant le temps qu'il a déjà passé dans le système.
– Dans le cas *malicieux*, l'apparition d'une faute est dépendante d'un choix fait par un ou un ensemble de noeuds corrompus (collusion) suivant une politique d'attaque inconnue. En général, on peut considérer que le choix de l'apparition ou non d'une faute, ainsi que de sa nature, est fait de façon à être la plus nocive pour le calcul (on se place en *pire cas*). Ici, l'apparition d'une faute ne peut être modélisée, il s'agit d'un cas de fautes byzantines, c'est le nombre de machines corrompues maximum qui est estimé.

D'autre part, les fautes peuvent être de deux types : fautes par arrêt ou fautes par valeur. Dans le cadre de la théorie des codes correcteurs, elles correspondent respectivement à des effacements et à des erreurs. Par exemple, la déconnexion d'un utilisateur dans un réseau pair-à-pair correspondra à une panne par arrêt (i.e. un effacement).

L'objectif de notre approche est d'arriver à avoir une confiance quantifiée dans le résultat (i.e. une probabilité de se tromper arbitrairement faible, choisie par l'utilisateur). Lorsque l'estimation du nombre de fautes est exacte, par rapport à un modèle de fautes *aléatoires* ou *malicieuses*, la construction (telle que présentée dans ce qui suit) d'un algorithme auto-tolérant aux fautes adapté à ce nombre de fautes est toujours possible. En pratique, une telle estimation peut être mise en défaut, il convient donc de vérifier la validité de cette estimation en vérifiant, de façon concurrente au calcul ([186]) ou a posteriori comme nous allons le voir, que le nombre d'erreurs dans le résultat ne dépasse pas le seuil estimé.

9.2.1.2 Certification de résultat

La certification de résultat ne fait pas partie de l'algorithme auto-tolérant aux fautes, elle permet de s'assurer que les hypothèses sur les occurrences et la nature des fautes ou sur la capacité des attaquants sont correctes. Dans un système totalement ouvert, tel qu'un système pair-à-pair, où les ressources ne sont pas sensées être bien protégées et tout simplement ne sont pas dignes de confiance, il est très difficile de mettre des bornes au taux d'erreurs. Il est donc important, lorsque c'est possible, de certifier, a posteriori, que le résultat est correct.

Définition 23 (Post-condition probabiliste) *Soit A un problème et Best le meilleur algorithme connu résolvant A (on note $\mathfrak{C}$(Best) le coût de Best).*
Alors A possède une post-condition probabiliste si et seulement si il existe un algorithme V et un couple de valeurs $(p_V, p_F) \in]0,1]^2$, tels que $\mathfrak{C}(V) \ll \mathfrak{C}$(Best) et pour tout x de l'ensemble des entrées de A, tout y de l'ensemble des sorties de A,

$$\Pr(V(x,y) = Vrai \mid A(x) = y) \geq p_V$$
$$\Pr(V(x,y) = Faux \mid A(x) \neq y) \geq p_F$$

Exemple pour le produit de matrices L'algorithme 5 décrit une certification par *post-condition probabiliste* pour le produit de matrices dans un anneau unitaire intègre $\mathbb{K}$.

Théorème 9 *l'algorithme 5 est un algorithme de certification probabiliste pour le produit de matrices avec $p_V = 1$ et $p_F \geq 1 - \frac{1}{\#\{S\}}$.*

Algorithme 5: Certification par post-condition pour le produit de matrices

Entrée : Matrices A, B et C à valeurs dans $\mathbb{K}$
Sortie : $Vrai$ si $C = A \times B$, $Faux$ sinon

Tirer aléatoirement un vecteur u à valeurs dans S, un sous-ensemble fini de $\mathbb{K}$ ou une extension de $\mathbb{K}$ (de façon à assurer que le cardinal $\#\{S\}$ de S soit grand (e.g. 2^{32}))
$u' \leftarrow B \cdot u$;
$u'' \leftarrow A \cdot u'$;
$v \leftarrow C \cdot u$;
$w \leftarrow u'' - v$;
si $w = 0$ **alors**
  retourner $Vrai$
sinon
  retourner $Faux$

Preuve Nous supposerons que les matrices A, B et C sont carrées de dimensions n. Le coût de l'algorithme est de 3 produits matrice-vecteur ($O(n^2)$), donc plus efficace que le produit de matrices ($O(n^\omega)$).

Si le calcul de certification est effectué sans erreur,

- Si $C = A \times B$ alors l'algorithme 5 renvoie $Vrai$ avec une probabilité 1, on a bien $w = A \cdot (B \cdot u) - C \cdot u = 0$ et donc $p_V = 1$.
- Si $C \neq A \times B$, alors soit le vecteur de polynômes multivariés $W(X_1, X_2, \cdots, X_n)$ tel que :

$$\begin{aligned} W(X_1, X_2, \cdots, X_n) &= (C - A \times B) \cdot \begin{pmatrix} X_1 \\ X_2 \\ \cdot \\ \cdot \\ X_n \end{pmatrix} \\ &= \begin{pmatrix} W_1(X_1, \cdots, X_n) \\ W_2(X_1, \cdots, X_n) \\ \cdots \\ \cdots \\ W_n(X_1, \cdots, X_n) \end{pmatrix} \end{aligned}$$

W est un vecteur de n polynômes de degré 1 en n variables. Comme $C - A \times B \neq 0$, alors $\exists i, W_i(X_1, \cdots, X_n) \neq 0$. D'après le lemme de Swartz-Zippel, en tirant au hasard un vecteur u à valeurs dans S, $\Pr(W_i(u) \neq 0) \geq \frac{1}{\#\{S\}}$. On a donc bien $p_F \geq 1 - \frac{1}{\#\{S\}}$. □

Il est facile de voir qu'au moins une partie de la certification doit nécessairement être exécutée sur des machines sûres : dans le cas contraire, sachant que le résultat de certification est une valeur booléenne (Vrai ou Faux), il faut se tourner vers les techniques d'*accord byzantin* [136, 125, 113] et donc faire une hypothèse sur le nombre de ressources corrompues (alors que la certification était justement là pour s'affranchir d'une telle hypothèse).
L'utilisation d'une post-condition nous permet d'avoir une confiance très grande dans le résultat,

seulement elle n'assure en rien la tolérance aux fautes. Ainsi, la seule certification n'est pas d'un grand secours lorsque le système a systématiquement des erreurs, même en faible proportions. Dans un tel cas, puisque la certification n'est jamais passée, le système serait en *dénis de service*.

9.2.1.3 Version itérative du produit de matrices

Le caractère volatile des ressources dans les systèmes distribués que nous étudions a été maintes fois souligné dans ce document, qu'il soit dû à un comportement malicieux ou simplement à l'architecture du système (e.g., réseaux pair-à-pair). Une construction récemment développée par Chen [53] et Bosilca et al. [43] utilise une version du produit de matrice particulièrement intéressante dans ce contexte.
L'idée est d'effectuer un produit de matrices itératif assurant, à chaque itération, le calcul d'une matrice résultat intermédiaire. Plus précisément, considérons le produit de matrices sous sa forme dite de *produit externe* ("outer product" en anglais). Soient A et B deux matrices carrées de dimensions $(k \times k)$ et notons $\{A_i^c\}_{1 \leq i \leq k}$ (resp. $\{B_i^l\}_{1 \leq i \leq k}$) les colonnes de A (resp. les lignes de B). Le produit $C = A \times B$ peut alors être calculé par produit externe sous la forme $C = \sum_{i=1}^{k} A_i^c \times B_i^l$. Ce produit, décrit dans l'algorithme 6, se ramène à k^3 produits de coefficients et à sommer k matrices de dimensions $k \times k$.

Algorithme 6: Produit de matrices par produit externe

Entrée : Matrices A, B
Sortie : C

$C_0 \leftarrow$ matrice nulle;
pour $i = 1 \cdots k$ **faire**
  $C_i = C_{i-1} + A_i^c \times B_i^l$;
retourner C_k

Les colonnes de A et lignes de B peuvent être remplacées par des blocs de colonnes et de lignes de même taille, i.e., à chaque itération, A_i^c contient nb colonnes de A et B_i^l contient nb lignes de B. La valeur nb est arbitrairement choisie dans $\{1, \cdots, k\}$, elle peut même être modifiée à chaque itération.
Ce produit de matrices permet une granularité plus fine dans la tolérance aux fautes, chaque matrice résultat intermédiaire peut être calculée en utilisant un des algorithmes tolérants aux fautes que nous présentons ici, et, par exemple, être certifiée pour constituer, en cours d'exécution, un état stable stocké sur les ressources R du système.

9.2.2 Schéma général du calcul tolérant aux fautes

Nous proposons d'étudier la construction d'algorithmes tolérants les fautes et donnons dès à présent le schéma général d'un tel algorithme s'exécutant sur le système de calcul global. A partir de ce schéma, nous comparerons l'utilisation de codes correcteurs efficaces par rapport à une méthode classique de réplications de tâches.

9.2.2.1 Algorithmes auto-tolérants aux fautes

Définition 24 (Algorithme auto-tolérant aux fautes) *Un algorithme sera dit auto-tolérant aux fautes si il renvoie la donnée de sortie du calcul sous forme redondante, i.e. encodée.*

Notations Dans toute la suite de ce chapitre nous noterons :
- f l'application à exécuter sur le système de calcul global, x les données d'entrée de l'application et y les données de sortie : $y = f(x)$. y peut être représentée sous la forme d'un vecteur $(y_1, \cdots, y_k)$ d'éléments dans un ensemble fini E.
- $\mathcal{C}$ est un code correcteur de dimension k, de taille de mot de code n et de distance minimale d sur l'alphabet E, i.e. de la forme $(n, k, d)_E$.
- φ la fonction de codage de $\mathcal{C}$. $\varphi : E^k \rightarrow \mathcal{C}$ est une fonction bijective qui associe à un vecteur $y = (y_1, \cdots, y_k)$ de E^k un mot de code $\varphi(y)$. On note $\varphi_i(y)$ la composante i de $\varphi(y)$ pour $i = 1 \cdots n$.
- τ est le nombre de symboles de redondance de $\mathcal{C}$: $\tau = n - k$.
- $\mathscr{A}_f^{\mathcal{C}}$ un algorithme auto-tolérant aux fautes d'impact 1 pour le calcul f. La sortie z de $\mathscr{A}_f^{\mathcal{C}}$ peut être représentée par un vecteur $(z_1, \cdots, z_n)$ d'éléments de E telle que $z = \varphi(y)$. Comme $\mathscr{A}_f^{\mathcal{C}}$ est d'impact 1, il peut être décomposé en n sous-tâches $(f_1, \cdots, f_n)$ indépendantes telles que $\forall i, f_i(x) = z_i$.

Proposition 1 *Pour toute application f dont la sortie y a une taille bornée que l'on peut représenter comme un vecteur de taille k à éléments dans un ensemble fini E, pour tout $n \geq k$ et tout code correcteur $\mathcal{C}$ de la forme $(n, k, d)_E$, il est possible de construire un algorithme auto-tolérant $\mathscr{A}_f^{\mathcal{C}}$ d'impact 1.*

Preuve C'est en fait le principe de base de la réplication de tâches : le calcul f est répliqué n fois. Chaque réplication de f a une donnée de sortie $(y_1, \cdots, y_k)$, qui peut être encodée dans le code $\mathcal{C}$ en z : pour $i = 1 \cdots n, z_i = \varphi_i(y_1, \cdots, y_k)$.
Nous obtenons donc aisément un algorithme auto-tolérant aux fautes d'impact 1, $\mathscr{A}_f^{\mathcal{C}} = (f_1, \cdots, f_n)$, tel que f_i renvoie la i-ème composante z_i de la sortie encodée de f : $f_i(x) = \varphi_i(f(x))$. □

La proposition 1 nous permet d'étudier le choix de codage sans nous soucier de l'algorithme auto-tolérant sous-jacent, nous savons qu'il est toujours possible d'en construire un. Ainsi, dans cette section, nous regarderons la construction d'algorithmes tolérants les fautes dans un système de calcul global $\{R, U\}$ sans donner plus de précision sur l'application f et étudierons l'influence du choix d'un code correcteur sur le coût de l'algorithme et sur la tolérance aux fautes. La preuve de la proposition 1 est constructive, en ce sens qu'elle donne une méthode de construction d'un algorithme auto-tolérant aux fautes d'impact 1 quelque soit l'application f et le code $\mathcal{C}$. Si l'on reste dans le cas général nous ne savons pas construire un algorithme plus efficace de manière systématique. Cependant, nous verrons dans les sections suivantes comment optimiser une telle construction pour le produit de matrices.

9.2.2.2 Schéma générique

L'algorithme 7 (illustré sur la figure 9.2) présente une version générique, sur la base de $\mathscr{A}_f^{\mathcal{C}}$, d'un algorithme tolérant les fautes avec certification probabiliste de résultat.

Algorithme 7: Algorithme générique tolérant aux fautes avec certification de résultat

Entrée : L'application f, les données d'entrée x, l'algorithme $\mathscr{A}_f^{\mathcal{C}}$, le code $\mathcal{C}$.

Sortie : $(Vrai, y = f(x))$ si succès, $(Faux, \cdot)$ sinon

Sur les ressources R :
1. Proposer un calcul ($f(x)$) sous forme de l'algorithme $\mathscr{A}_f^{\mathcal{C}}$ aux machines U;
Sur les ressources U :
2. Exécuter chaque groupe de tâche f_i ($0 < i \leq n$) de $\mathscr{A}_f^{\mathcal{C}}$ et renvoyer le résultat redondant $(\tilde{z}_1, \tilde{z}_2, \cdots, \tilde{z}_n)$ dans la zone de confiance;
Sur les ressources R :
3. Décoder $(\tilde{z}_1, \tilde{z}_2, \cdots, \tilde{z}_n)$: trouver le mot de code $(\bar{z}_1, \bar{z}_2, \cdots, \bar{z}_n)$ le plus proche dans le code $\mathcal{C}$ (les composantes $\tilde{z}_i$ non reçues dû à des pannes sont considérées comme des effacements);
4. Certifier :
pour $l = 1 \cdots N$ **faire**
  Tirer aléatoirement un indice $i \in \{1, \cdots, n\}$;
  Calculer $z_i = f_i(x)$;
  si $z_i \neq \bar{z}_i$ **alors**
    Abandonner le calcul : **retourner** $((Faux, \cdot))$;
Récupérer la sortie $y = (y_1, \cdots, y_k) = \varphi^{-1}((\bar{z}_1, \cdots, \bar{z}_n))$;
retourner $(Vrai, y)$;

Remarque 38 *Dans le cas où la réponse de l'algorithme 7 est $(Faux, \cdot)$, on peut éventuellement décider de relancer le calcul en entier ou en partie. Il est aussi parfois possible de proposer de nouvelles tâches pour augmenter la redondance du résultat ($n \leftarrow n + f$) (par extension du code correcteur), une telle solution permet de ne pas perdre les résultats précédents.*

Théorème 10 *L'algorithme 7 permet de corriger t fautes par valeurs et s fautes par arrêt si $s + 2t < d$ (d est la distance minimale du code $\mathcal{C}$ de $\mathscr{A}_f^{\mathcal{C}}$). De plus l'algorithme de certification vérifie :*

- $p_V = 1$
- $p_F \geq 1 - (\frac{n-d}{n})^N$

Preuve Par hypothèse, l'algorithme $\mathscr{A}_f^{\mathcal{C}}$ satisfait :

- $Impact(\mathscr{A}_f^{\mathcal{C}}, 1) = 1$ pour une représentation de la sortie $(z_1, \cdots, z_n)$.
- La sortie $(z_1, \cdots, z_n)$ de $\mathscr{A}_f^{\mathcal{C}}$ est le *mot de code* de taille n obtenu par codage de la sortie $y = (y_1, \cdots, y_k)$ de $f(x)$ dans un code de distance minimale d.

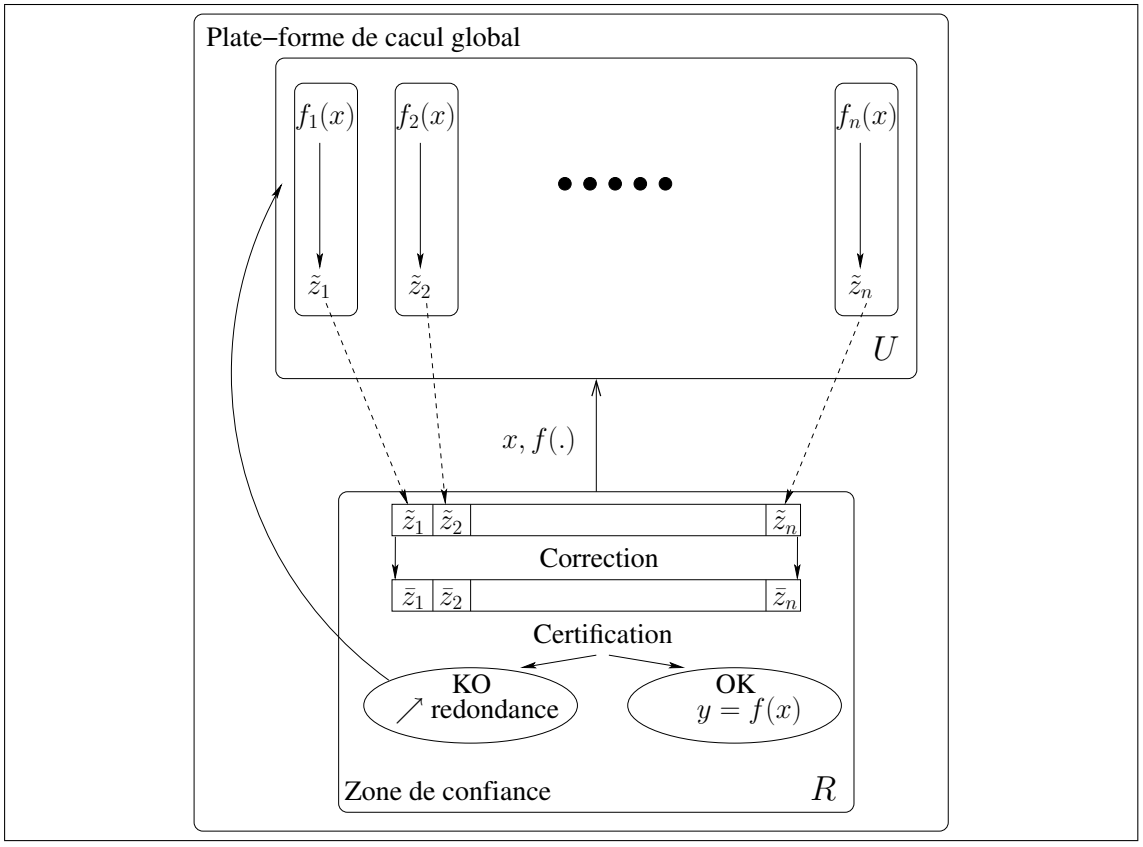

FIG. 9.2 – Schéma général d'un algorithme tolérant les fautes sur un système de calcul global $\{R, U\}$

Or nous savons que pour un code correcteur de distance minimale d, il est possible de corriger de façon unique un mot de code comportant t composantes erronées et s composantes effacées si $s + 2t \leq d - 1$ (voir la section 3.2 du chapitre 3).
Comme $Impact(\mathscr{A}_f^{\mathcal{C}}, 1) = 1$, nous en déduisons bien que l'algorithme 7 permet de corriger t fautes par valeurs et s fautes par arrêt si $s + 2t < d$.

Supposons maintenant que le mot de code $(\bar{z}_1, \cdots, \bar{z}_n)$ ne contienne pas d'erreur ($\forall 0 < i \leq n, z_i = \bar{z}_i$), alors chaque test de l'algorithme de certification sera réussi avec une probabilité 1 (ces tests sont exécutés sur les ressources fiables), nous avons donc bien

$$p_V = \Pr(\text{retourner } Vrai \mid \mathscr{A}_f^{\mathcal{C}}(x) = (\bar{z}_1, \cdots, \bar{z}_n)) = 1$$

Si $(\bar{z}_1, \cdots, \bar{z}_n)$ est erroné, d'après la définition de la distance minimale d'un code, nous savons qu'au moins d composantes sont fausses. En re-calculant une composante tirée au hasard dans le mot de code, la probabilité de choisir une composante erronée (et donc de mettre en évidence l'erreur), vaut $p = \frac{n-d}{n}$. Au bout de N re-calculs différents, nous avons une probabilité $p_N =$

$\prod_{i=0}^{N-1} \frac{n-d-i}{n-i} \leq (\frac{n-d}{n})^N$ de ne pas avoir re-calculé de composante fausse. Enfin, nous avons bien

$$p_F = \Pr(\text{retourner } Faux \mid \mathscr{A}_f^{\mathcal{C}}(x) \neq (\bar{z}_1, \cdots, \bar{z}_n)) \geq 1 - (\frac{n-d}{n})^N$$

□

L'algorithme 7 est le point de départ de notre étude d'algorithmes tolérants aux fautes, il est intéressant de remarquer ici que l'algorithme de certification probabiliste générique proposé est dépendant du code (ou structure de la redondance) de l'algorithme auto-tolérant aux fautes utilisé : plus le code aura une distance minimale grande, plus l'algorithme de certification sera efficace. Dans le cas du produit de matrice, l'algorithme 5 de certification est généralement plus efficace et c'est celui que nous utiliserons pour ce cas d'étude.

Nous allons tout d'abord étudier, sans spécifier le calcul à exécuter, l'influence d'un choix de code correcteur sur le coût de calcul et de stockage sur les ressources R. Puis nous allons proposer et comparer, pour le produit de matrices, des algorithmes auto-tolérants aux fautes d'impact 1.

9.2.3 Choix de codes correcteurs et influence sur R

La première solution que nous allons voir est le codage par réplication pure, avec un décodage par vote de majorité. Nous verrons ensuite deux codes correcteurs linéaires : les codes de Reed-Solomon et les codes LDPC (voir section 3.2.1.2 pour une présentation des codes linéaires et des codes de Reed-Solomon et LDPC).

9.2.3.1 Réplication pure

La réplication rentre naturellement dans le schéma général de tolérance aux fautes. Elle est très largement utilisée dans les systèmes que nous étudions parce qu'elle permet de construire aisément un algorithme auto-tolérant aux fautes d'impact 1 et ce pour n'importe quelle application. On peut la considérer comme un code correcteur de la forme $(n, k, d)_E$ tel que $n = k \cdot d$: chaque composante de la sortie y est répliquée d fois ; pour le décodage, un vote de majorité est fait pour chacune des composantes de y. E est un ensemble quelconque de taille finie q, ses éléments sont représentés sur $\log q$ bits.

Proposition 2 *Pour un code par réplication $(dk, k, d)_E$, le coût du décodage sur les ressources R est de $W = k \cdot O(d)\omega_q$ et $D = \delta_q$. La taille de stockage nécessaire sur les ressources fiables est d'au moins $kd \log q$ bits.*

Preuve Chaque composante $y_1, \cdots, y_k$ est répliquée d fois, la distance minimale du code est trivialement d. Le décodage par vote de majorité consiste à compter l'occurrence majoritaire d'un élément parmi d éléments (à faire pour chaque composante). Il est nécessaire de stoker les éléments distincts parmi les d réplications de façon à faire le vote : en pire cas $d \log q$ bits. Si l'on considère que le décodage est faux lorsqu'il y a plus de $\lfloor \frac{d}{2} \rfloor$ erreurs par valeurs, alors on peut limiter, en pire cas, la taille de stockage à $\lceil \frac{d}{2} \rceil \log q$ bits puisque si il y a plus d'éléments distincts

alors le nombre d'erreurs dépasse forcément la distance minimale. □

Tolérance aux fautes D'après le théorème 10 et la proposition 2, l'algorithme par réplication pure $((dk, k, d)_E)$ tolère t fautes par valeur et s fautes par arrêt si $s + 2t < d$.

Dans le cas de fautes malicieuses, les tâches corrompues chercherons à modifier les calculs de façon cohérente pour qu'une fausse valeur soit majoritaire lors du vote. Dans le cas de fautes aléatoires, la tolérance est plus difficile à évaluer, si des erreurs apparaissent dans les calculs, elles ont peu de chance d'être cohérentes les unes avec les autres, i.e. qu'apparaisse la même valeur sur la même composante. Dans ce cas, la tolérance aux fautes est dépendante de la taille des coefficients y_i, de la taille du vecteur (k) et de la distribution des erreurs. Dans le cas général, nous admettrons que la probabilité d'occurrence d'une erreur non détectée est négligeable (noté d'une étoile dans le tableau 9.2).

Conclusions La réplication n'est pas réputée être un code correcteur efficace, en effet, sa distance minimale est loin d'être optimale. Nous allons voir que l'utilisation de codes correcteurs plus complexes nous permet d'augmenter la capacité de correction pour un coût de calcul sur R équivalent.

Remarquons tout de même que le choix de k et de l'ensemble E a des conséquences notables sur R : pour une sortie y fixée, réduire la valeur de k (et donc augmenter la taille de q) permet de réduire sensiblement la taille de stockage et le coût de décodage au détriment de la capacité de tolérance aux fautes aléatoires, par contre, en pire cas (cas malicieux), la tolérance aux fautes reste la même.

9.2.3.2 Reed-Solomon et codes par interpolation

Les codes de Reed-Solomon (notés RS dans la suite, section 3.2.3) et plus généralement les codes par interpolation de polynômes univariés (section 3.2.4) ont une *distance minimale* maximale : ils ont une capacité de correction maximale en pire cas (fautes malicieuses) tout en proposant un algorithme de décodage quasi-linéaire. Ces codes sont de la forme $(n, k, n - k + 1)_E$ où E peut être le corps fini à q éléments $\mathbb{F}_q$ (dans le cas de codes de Reed-Solomon) ou plus généralement un sous-ensemble fini d'un corps commutatif (dans le cas de codes par interpolation). Par définition de ces codes, il est nécessaire d'avoir $n \leq q - 1$.

Proposition 3 *Pour un code de type interpolation polynomiale $(n, k, n - k + 1)_E$, le coût du décodage sur les ressources R est de $W = O(n \log^2 n)\omega_q$ et $D = O(\log^3 n)\delta_q$. La taille de stockage nécessaire sur les ressources fiables est de $n \log q$ bits.*

Preuve Le travail W et la taille de stockage sur R se déduisent directement de l'étude des codes de Reed-Solomon faite en section 3.2.3. La profondeur D pour l'algorithme de décodage proposé dans la section 3.2.3 (i.e. évaluation/interpolation de polynôme et algorithme d'Euclide) est décrit dans l'ouvrage de Bini et Pan [32] :

- Évaluation/Interpolation :
  - $D = O(\log^2 n)\delta_q$ pour un travail total $W = O(n \log^2 n)\omega_q$ dans le cas général.

- $D = O(\log n)\delta_q$ pour un travail total $W = O(n \log n)\omega_q$ dans le cas d'une puissance n-ème de l'unité (RS).

– Trouver le PGCD de deux polynômes :
  - Algorithme d'Euclide rapide : $D = O(\log^3 n)\delta_q$ pour un travail total $W = O(n^2 \log n)\omega_q$.
  - Résolution d'un système linéaire $D = O(n \log n)\delta_q$ pour un travail total $W = O(n \log^2 n)\omega_q$.

Ainsi, l'algorithme d'Euclide impose au décodage sa complexité, deux algorithmes sont avancés : le premier est plus efficace pour une exécution séquentielle (W plus faible), le deuxième est plus efficace pour une exécution parallèle lorsque le nombre de processeurs est suffisant (D plus faible). Nous considérons donc que suivant les caractéristiques de la plate-forme R, l'un ou l'autre des algorithmes sera utilisé, sans compter qu'il est peut-être possible de construire un algorithme prenant l'avantage de chacun des algorithmes par composition de portions séquentielles et parallèles. □

Tolérance aux fautes D'après le théorème 10, l'algorithme 7 avec un code de type $((n, k, n - k + 1)_E)$ tolère t fautes par valeurs et s fautes par arrêt (au niveau processus), si $s + 2t < n - k + 1$.

Dans le cas de fautes malicieuses, pour un algorithme de décodage un peu plus coûteux sur les ressources R et une taille de stockage identique (en fixant le nombre de symboles de redondance $\tau = n - k$), les codes par interpolation permettent de corriger beaucoup plus d'erreurs : la distance minimale est de $n - k + 1$ au lieu de $\frac{n}{k}$.
Dans le cas de fautes aléatoires, pour les mêmes raisons que précédemment, la tolérance est plus difficile à évaluer, nous supposerons qu'elle est équivalente au cas de réplication pure pour le même taux de redondance. Les codes RS (avec $n = q - 1$) sont des codes parfaits, dans ce cas uniquement, le nombre de fautes corrigées dans le cas malicieux et aléatoire est exactement le même : $2t + s < n - k + 1$.

Conclusions Comme nous l'avions pressenti, l'utilisation de codes correcteurs efficaces tels que les codes par interpolation polynomiale donne de bons résultats, notons tout de même que le coût de l'algorithme de décodage sur les ressources R est sensiblement augmenté et que, pour utiliser ces codes, des conditions sur la nature de l'ensemble E et sa taille doivent être assurées. D'autre part, la capacité à tolérer des fautes est bien plus précise qu'avec l'usage de réplication pure, ainsi il est possible de tolérer un taux d'erreurs choisi, e.g. 10% des composantes $(z_1, \cdots, z_n)$: $n = k + \frac{2n}{10}$ et donc $n = \frac{10}{8}k$. Par opposition un code par réplication pure ne permet de corriger qu'un taux d'erreurs de $\frac{1}{2k}$.

9.2.3.3 Low Density Parity Check Codes

Les codes LDPC (section 3.2.5) sont des codes linéaires de distance minimale faible en général, ils ont pourtant une capacité de correction proche de l'optimum lorsque les erreurs sont aléatoires. Dû à leur faible distance minimale, il est généralement possible de trouver un mot de code de poids faible et ainsi, en modifiant soigneusement peu de composantes d'un mot de code, rendre le décodage incorrect. Ces codes sont de la forme $(n, k, d)_E$ où E est le corps fini à q éléments $\mathbb{F}_q$.

Proposition 4 *Pour un code de type LDPC $(n, k, d)_E$, le coût du décodage sur les ressources R est de $W = O(n)\omega_q$. La taille de stockage nécessaire sur les ressources fiables est de $n \log q$ bits.*

Preuve Le travail W et la taille de stockage sur R se déduisent directement de la présentation des codes LDPC faite en section 3.2.5. L'évaluation de la profondeur D du calcul parallèle est très dépendant de l'algorithme de décodage utilisé et nous n'avons pas trouvé de résultat général sur ce sujet. Wang, Chen et Wu [223] développent un algorithme de décodage parallèle de type "propagation de croyance", sa profondeur est de l'ordre de $s\delta_q$ où s est le nombre maximal de composantes non nulles sur une colonne de la matrice de parité du code LDPC. □

Tolérance aux fautes D'après le théorème 10, l'algorithme 7 par codes LDPC ($(n, k, d)_E$) tolère t fautes par valeurs et s fautes par arrêt (au niveau processus), si $s + 2t < d$.

Le cas des codes LDPC est un peu particulier car il n'existe pas, dans le cas général, d'algorithme efficace pour trouver la distance minimale d'un tel code (ou encore trouver un mot de code de poids minimal) : ce problème peut se réduire à un problème NP-Complet. Par contre, cette distance n'a pas de raison d'être grande et si elle est faible il est généralement possible de trouver un mot de code de poids faible. Nous renvoyons le lecteur vers les travaux de Kim et al. [123] sur la recherche de mots de codes de poids faible dans les codes LDPC.

Dans le cas de fautes aléatoires, certains codes LDPC ont la particularité d'avoir un taux de correction proche de l'optimal, nous aurons donc la meilleure correction possible avec ces codes, nous donnons dans le tableau 9.2, deux résultats heuristiques pour des codes LDPC sur canal symétrique binaire ($E = \mathbb{F}_2$).

Conclusions Les codes LDPC ont l'avantage de n'ajouter qu'un surcoût algorithmique très faible par rapport à la réplication tout en satisfaisant au moins les même capacités de tolérance aux fautes. De ce point de vue, l'utilisation de ces codes est presque toujours plus intéressante que la réplication pure. Par contre, il faut travailler dans $\mathbb{F}_q$ et ces codes seront particulièrement efficaces lorsque la taille de q est faible.

9.2.3.4 Synthèse

Le tableau 9.1 récapitule la complexité algorithmique sur les ressources R pour les codes correcteurs proposés. Le tableau 9.2 considère uniquement la capacité à tolérer les fautes en fonction du modèle.

Les codes sont de la forme $(n, k, d)_E$ avec $d = n - k + 1$ pour les codes par interpolation polynomiale, $d = \frac{n}{k}$ pour le code par réplication pure et d est petit devant n pour les codes LDPC.

Le taille de stockage nécessaire sur les ressources R (i.e. mémoire fiable) est d'au moins $n \log q$ bits.

Remarque 39 *Il est intéressant de remarquer ici que, pour une application f fixée, la donnée de sortie y peut être représentée suivant plusieurs "découpages" $(y_1, \cdots, y_k)$ dans différents en-*

	Zone de Confiance (R) Décodage
Réplication	$W = O(n)\omega_q$ $D = \delta_q$
Reed-S.	$W = O(n \log^2 n)\omega_q$ $D = O(\log^3 n)\delta_q$
LDPC	$W = O(n)\omega_q$ $D = O(??)\delta_q$

TAB. 9.1 – Complexité des algorithmes tolérants aux fautes sur R en fonction du code correcteur

	Fautes aléatoires	
	pannes par arrêt	pannes par valeur
Réplication	$\star$	$\star$
Reed-S.	$\star$ ($\geq n - k$)	$\star$ ($\geq \lfloor \frac{n}{2k} \rfloor$)
LDPC	49% de n confiance $1 - 10^{-4}$ $n = 2k = 0.25 \times 10^7$ [167]	15% de n confiance $1 - 10^{-6}$ $n = 2k = 0.5 \times 10^6$ [181]
	Fautes malicieuses	
	pannes par arrêt	pannes par valeur
Réplication	$\frac{n}{k}$	$\lfloor \frac{n}{2k} \rfloor$
Reed-S.	$n - k$	$\lfloor \frac{n-k}{2} \rfloor$
LDPC	$d - 1$	$\lfloor \frac{d-1}{2} \rfloor$

TAB. 9.2 – Capacité de détection/correction des algorithmes tolérants aux fautes en fonction du code correcteur

sembles E. Le choix de la taille k (et donc de la taille de E) a un impacte sur la complexité du décodage. Nous verrons dans la section suivante, trois algorithmes auto-tolérants aux fautes pour le produit de matrices, faisant varier ces valeurs et donc le coût de l'algorithme sur les ressources R.

Nous avons vu que, si l'on considère acquis la donnée d'un algorithme auto-tolérant aux fautes d'impact 1 pour un code correcteur considéré, alors il est intéressant d'utiliser des codes correcteurs efficaces tels que les codes de Reed-Solomon ou codes LDPC pour tolérer les fautes sur un système de calcul global. Ces codes donnent à la fois une meilleur capacité à tolérer les fautes que la réplication pure, tout en offrant une meilleur maîtrise sur le choix du taux de fautes que l'on veut corriger.

Cette étude en boîte-noire vis-à-vis de l'application nous a permis de proposer un algorithme complet tolérant aux fautes avec certification de résultat. La proposition 1 et l'algorithme 7 seraient donc suffisants si l'on considère que la puissance de calcul Π_U sur les ressources U est non bornée et que le nombre de fautes sur U est borné : ainsi même si l'algorithme auto-tolérant aux fautes construit de façon naïve (cf. preuve de la proposition 1) est bien plus coûteux pour un code correcteur complexe que par simple réplication, tant que le nombre de fautes n'augmente pas avec le travail, les résultats sont pertinents.

Cette hypothèse étant peu réaliste, nous devons maintenant d'étudier la construction d'algorithmes auto-tolérants aux fautes d'impact 1 efficaces pour les codes correcteurs proposés. Nous proposons de faire cette étude sur l'application du produit de matrices.

9.3 Application au produit de matrices et premiers résultats

Nous allons tout d'abord proposer trois constructions immédiates d'algorithmes auto-tolérants aux fautes d'impact 1 pour le produit de matrices, nous en déduirons les coûts de ces algorithmes sur les ressources U (indépendant du codage) et sur les ressources R dans le cas des codes correcteurs RS et LDPC. Dans les deux sections suivantes nous verrons comment construire des algorithmes mieux adaptés aux codes correcteurs à l'aide de deux méthodes classiques.

Notations Dans toute la suite de ce chapitre nous noterons :
- A et B deux matrices carrées de dimensions $(k \times k)$ définies dans un anneau unitaire.
- C le résultat du produit usuel des matrices A et B, C est donc une matrice carrée de dimension $(k \times k)$.
- $\mathscr{A}_{simple}$ est l'algorithme naïf du produit de matrices (d'impact 1). Il a une complexité de $W = O(k^3)$ opérations sur des éléments de E et une profondeur $D = O(\log k)$.
- $\mathscr{A}_\omega$ est un algorithme de produit de matrices (d'impact k^2). Il a une complexité de $W = O(k^\omega)$ opérations avec $(2 \geq \omega < 3)$ et un profondeur $D = O(\log k)$.

9.3.1 Algorithmes auto-tolérants aux fautes

La fonction f est donc maintenant le produit de deux matrices carrées de dimension $(k \times k)$, sa sortie est donc un vecteur de k^2 éléments dans l'ensemble E, nous considérerons comme précédemment que E est de taille finie q.
Nous en déduisons trois algorithmes, jouant sur trois représentations différentes de la matrice C :

Algorithme 8 Un vecteur de k^2 éléments de E qui peut être calculé efficacement à l'aide $\mathscr{A}_\omega$. La sortie $(z_1, \cdots, z_n)$ est la représentation codée de C dans un code $(n, k^2, d)_E$.

Algorithme 9 k vecteurs de k éléments de E qui peuvent être calculés de façon complètement indépendante (avec $\mathscr{A}_{simple}$).
La sortie $((z_1^1, \cdots, z_n^1), \cdots (z_1^k, \cdots, z_n^k))$ est la représentation des k colonnes de C, chacune encodée dans un code $(n, k, d)_E$.

Algorithme 10 k^2 coefficients (vecteurs de 1 élément de E) qui peuvent être calculés de façon complètement indépendante (avec $\mathscr{A}_{simple}$). La sortie $((z_1^1, \cdots, z_n^1), \cdots (z_1^{k^2}, \cdots, z_n^{k^2}))$ est la représentation des k^2 composantes de C, chacune encodée dans un code $(n, v, d)_I$. I est un ensemble fini à u éléments tel que l'on peut écrire $E = I^v$ (typiquement lorsque E est une extension de corps fini).

Algorithme 8: Algorithme auto-tolérant aux fautes par codage de la matrice.

Entrée : Deux matrices A et B. Un code $\mathcal{C} : (n, k^2, d)_E$

Sortie : C encodée dans le code $\mathcal{C}$.

Sur les ressources R :

Rendre accessible en lecture aux ressources U les matrices A et B stockées sur les ressources R;

Sur les ressources U :

Exécuter, de façon indépendante pour $0 < i \leq n$, f_i :
- $C_i = A \times B$;
- C_i est un vecteur de k^2 éléments de E, l'encoder dans $\mathcal{C} : \varphi(C_i)$;
- Écrire la i-ème composante du mot de code $(\varphi_i(C_i))$ dans la zone de confiance;

Sur les ressources R :

Récupération $(\varphi_1(C_1), \cdots, \varphi_n(C_n))$ (permettra de retrouver la matrice C);

Chacun de ces algorithmes se base sur une représentation particulière de la matrice résultat C, cette différence a des conséquences sur le coût de décodage sur les ressources R, le tableau 9.3 reprend, pour chaque cas, les coûts sur R en fonction du code correcteur et le coût de l'algorithme auto-tolérant sur U. L'algorithme dit de référence sur le tableau récapitulatif correspond au produit de matrices directement exécuté sur les machines fiables (algorithme intolérant aux fautes $\mathscr{A}_\omega$) ; l'algorithme utilisant la réplication pure correspond à la réplication d fois de l'algorithme $\mathscr{A}_\omega$ sur les machines U et à un décodage par vote de majorité sur chaque composante des d matrices résultantes.

Dans leur forme actuelle, les algorithmes sont applicables à tout calcul ayant une sortie sur $k^2 \log q$ bits, nous avons vu qu'il existe un degré de liberté dans la répartition de la charge de travail entre les machines sûres et le reste du système. Ainsi, au prix d'un surcoût significatif de travail sur les machines non-sûres, le coût de l'algorithme sur la partie fiable du système peut être diminuée, jusqu'à atteindre une complexité algorithmique et une taille de stockage proche

Algorithme 9: Algorithme tolérant aux fautes par codage des colonnes de la matrice.

Entrée : Deux matrices A et B. Un code $\mathcal{C} : (n, k, d)_E$

Sortie : C dont chaque colonne est encodée dans le code $\mathcal{C}$.

Sur les ressources R :

Rendre accessible en lecture aux ressources U les matrices A et B stockées sur les ressources R.

Sur les ressources U :

Exécuter, de façon indépendante pour $0 < i \leq n$, f_i :
- $C_i = A \times B$;
- Chaque colonne de C_i est un vecteur de k éléments de E, les encoder dans $\mathcal{C}$: $(\varphi(C^c_{i,1}), \cdots, \varphi(C^c_{i,k}))$;
- Écrire la i-ème composante de chaque mot de code $(\varphi_i(C^c_{i,j}))$ dans la zone de confiance;

Sur les ressources R :

Récupération, pour tout $0 < j \leq k$, de $(\varphi_1(C^c_{1,j}), \cdots, \varphi_n(C^c_{n,j}))$ (permettra de retrouver la j-ème colonne de la matrice C);

Algorithme 10: Algorithme tolérant aux fautes par codage des composantes de la matrice.

Entrée : Deux matrices A et B. Un code $\mathcal{C} : (n, v, d)_I$

Sortie : C dont chaque composante est encodée dans $\mathcal{C}$.

Sur les ressources R :

Rendre accessible en lecture aux ressources U les matrices A et B stockées sur les ressources R;

Sur les ressources U :

Exécuter, de façon indépendante pour $0 < i \leq n$, f_i :
- $C_i = A \times B$;
- Chaque composante de C_i est un vecteur de v éléments de I, les encoder dans $\mathcal{C}$: $(\varphi(C_i[1,1]), \cdots, \varphi(C_s[k,k]))$;
- Écrire la i-ème composante de chaque mot de code $(\varphi_i(C_i[i,j]))$ dans la zone de confiance;

Sur les ressources R :

Récupération, pour tout $0 < i \leq k$ et $0 < j \leq k$, de $(\varphi_1(C_1[i,j]), \cdots, \varphi_n(C_n[i,j]))$ (permettra de retrouver composante $C[i,j]$ de la matrice C);

de l'optimum (algorithme 8). Nous allons voir maintenant comment, tout en conservant un coût constant sur les machines fiables et une tolérance aux fautes constante, il est parfois possible de réduire le coût de l'algorithme sur les machines non-sûres.

		Zone de Confiance (R) Décodage	Zone non sûre (U) Produit de matrices
Stockage $k^2 \log q$	Référence	$W = O(k^\omega)\omega_q$ $D = O(\log k)\delta_q$	0 0
$\mathcal{C}$=$(dk^2, k^2, d)_E$ Stock. $dk^2 \log q$	Réplication	$W = k^2 \cdot O(d)\omega_q$ $D = \delta_q$	$W = d \cdot O(k^\omega)\omega_q$ $D = O(\log k)\delta_q$
Alg. 8 $\mathcal{C}$=$(n, k^2, d)_E$ $n = k^2 + \tau$ Stock. $(k^2+\tau)\log q$	Reed-S.	$W = O((k^2+\tau)\log^2(k^2+\tau))\omega_q$ $D = O(\log^3(k^2+\tau))\delta_q$	$W = (k^2+\tau) \cdot O(k^\omega)\omega_q$ $D = O(\log k)\delta_q$
	LDPC	$W = O((k^2+\tau))\omega_q$ $D = O(??)\delta_q$	
Alg. 9 $\mathcal{C}$=$(n, k, d)_E$ $n = k + \tau$ Stock. $k(k+\tau)\log q$	Reed-S.	$W = k \cdot O((k+\tau)\log^2(k+\tau))\omega_q$ $D = O(\log^3(k+\tau))\delta_q$	$W = (k+\tau) \cdot O(k^\omega)\omega_q$ $D = O(\log k)\delta_q$
	LDPC	$W = k \cdot O((k+\tau))\omega_q$ $D = O(??)\delta_q$	
Alg. 10 $\mathcal{C}$=$(n, v, d)_I$ $n = v + \tau$ Stock. $k^2(v+\tau)\log u$	Reed-S.	$W = k^2 \cdot O((v+\tau)\log^2(v+\tau))\omega_u$ $D = O(\log^3(v+\tau))\delta_u$	$W = (v+\tau) \cdot O(k^\omega)\omega_q$ $D = O(\log k)\delta_q$
	LDPC	$W = k^2 \cdot O((v+\tau))\omega_u$ $D = O(??)\delta_u$	

TAB. 9.3 – Complexité des algorithmes auto-tolérants aux fautes

9.3.2 Utilisation de codes systématiques ou creux et conséquence sur U

La première remarque que l'on peut faire, en considérant l'algorithme 8 sur les ressources U non-fiables, est que l'utilisation de codes sous forme systématique améliore, sans aucun effort, la complexité algorithmique. Tels que définis dans le chapitre 3 (définition 14), les codes systématiques se contentent d'ajouter au message à encoder des symboles de redondance en laissant inchangé le message lui-même. Cela signifie, pour l'algorithme 8 utilisant un code sous forme systématique, que les k^2 premières tâches $(f_1, \cdots, f_{k^2})$ renvoient chacune, une des k^2 composantes de la matrice résultat C. Or le calcul d'une composante de la matrice $C = A \times B$ se fait en $O(k)$ opérations (produit scalaire de deux vecteurs de taille k).

En utilisant un code sous forme systématique dans l'algorithme 8, cette simple remarque permet de passer d'une complexité $W = O((k^2+\tau)k^\omega)\omega_q$ (cf. tableau 9.3) à $W = O((k^3+\tau k^\omega))\omega_q$.

De la même façon, dans l'algorithme 9 qui encode les colonnes de la matrice résultat, en utilisant un code sous forme systématique, les tâches $(f_1, \cdots, f_k)$ calculent une des ligne de la matrice résultat C, ainsi, au lieu d'une complexité, sur les ressources U, de $W = O((k+\tau)k^\omega)\omega_q$ (cf.

tableau 9.3) la complexité passe à $W = O(k^3 + \tau k^\omega)\omega_q$.

Pour l'algorithme 10 sur les composantes de la matrice, l'utilisation d'un code systématique ne change pas le fait que chaque groupe de ressource doit faire un produit de matrice complet et n'a donc pas d'intérêt particulier.

Codes linéaires creux Nous avons vu que les codes LDPC avaient la particularité d'avoir une matrice de parité creuse. Et nous savons (cf. section 3.2.5 du chapitre 3) qu'il existe certains codes LDPC, les "Low Density Generator Matrix" (LDGM) qui ont à la fois, une matrice de parité creuse et une matrice génératrice creuse.
L'utilisation de codes avec matrice génératrice creuse permet d'optimiser le calcul les algorithmes 8 et 9 comme le faisait les codes sous forme systématique. En effet, si l'on utilise un code dont la matrice génératrice ne contient pas plus de s composantes non nulles sur ses colonnes (i.e. une composante d'un mot de code est la combinaison linéaire d'au plus s symboles du message), alors pour chaque tâche f_i, il suffit de calculer,

- s composantes de la matrice C pour l'algorithme 8 : $W = O(s(k^2 + \tau)k)\omega_q$
- s lignes de la matrice C pour l'algorithme 9 : $W = O(s(k + \tau)k^2)\omega_q$

Cette solution est plus contraignante que l'utilisation de codes systématiques, puisque le choix du code correcteur est plus limité, par contre, elle permet d'éviter une différence trop grande dans la charge de travail des groupes de tâches $(f_1, \cdots, f_n)$.

Extensions vers d'autres applications L'utilisation de codes sous forme systématique ou de codes linéaires creux n'est possible que lorsque le résultat du calcul se découpe en composantes pouvant être calculées indépendamment les unes des autres (i.e. algorithme d'impact 1), le cas du produit de matrices s'y prête particulièrement bien, ce ne sera pas le cas en général.

9.4 Produit de matrices et codes linéaires (ABFT)

Nous considérons dans cette section une méthode classique de construction d'algorithmes auto-tolérants aux fautes : les ABFT (pour "Algorithm Based Fault Tolerance" en anglais). Après un bref état de l'art de ce domaine, nous verrons comment appliquer cette méthode dans notre contexte ; le produit de matrices étant l'application privilégiée de ces techniques, nous rassemblerons les constructions existantes autour d'un schéma général simple, qu'il faudra ensuite l'adapter à notre contexte de système de calcul global.

A notre connaissance, il s'agit de la première étude de construction d'ABFT utilisant des codes de type LDPC et destinés à des modèles de fautes massives ou byzantines.

9.4.1 ABFT : Définition et état de l'art

Définition La définition la plus répandue d'ABFT est celle donnée par l'article original de Huang et Abraham [100] :

> "La technique d'ABFT consiste à encoder les données d'entrées du système et à modifier l'algorithme pour s'adapter aux données codées et produire des données de sortie encodées".

Sur un système composé de plusieurs processeurs, la répartition des tâches (ou processus) est faite de façon à ce qu'un processeur défectueux ne puisse affecter qu'une faible partie des données [156]. Cette définition est très proche de celle que nous avions donnée aux algorithmes autotolérants aux fautes, la différence est justement le sujet de cette section : travailler sur les entrées encodées permet d'alléger le calcul sur les ressources non-fiables. En effet, si au moins une partie des tâches $f_1, \cdots, f_n$ étaient jusqu'ici obligées de contenir le produit de matrice en entier (dans le cas d'un code dense), c'est parce que certains symboles du mot de code sont dépendants d'un grand nombre (si ce n'est la totalité) des symboles du message original : Il faut donc avoir calculer l'ensemble du message (résultat du produit de matrice) pour pouvoir calculer un seul symbole du mot de code.
Lorsque c'est possible, les constructions ABFT détournent le problème en encodant l'entrée du calcul (ici les matrices A et B) et en exécutant l'algorithme de telle sorte que le codage se conserve à travers le calcul et permette donc de récupérer une sortie encodée. Si l'algorithme modifié, pour travailler sur les entrées encodées, peut être décomposé en tâches indépendantes (i.e. d'impact 1 d'après notre définition), nous obtenons un algorithme tolérant aux fautes efficace.

En effet, les constructions d'ABFT ont prouvé leur grande efficacité pour certaines classes d'applications. Par contre, ces techniques sont très dépendantes de la structure de l'application et n'ont peuvent être appliquées à n'importe quel calcul. Cette technique de tolérance aux fautes a été très largement étudiée et utilisée depuis le papier de Huang et Abraham. Divers façons de la mettre en oeuvre ont été proposées, nous verrons un bref aperçu de ces techniques dans la section suivante.

9.4.1.1 Applications

L'article introductif de l'ABFT ([100]) a pour objet l'implémentation matérielle d'un algorithme tolérant une faute pour le calcul de produit de matrices. Si les premières études ont prolongé ce premier résultat sur la conception de machines dédiées pour d'autres opérations matricielles et pour la transformée de Fourier rapide [56], le concept a rapidement évolué vers les machines multiprocesseurs non-dédiées et la correction de fautes multiples.

Comme il a été dit plus haut, la construction d'ABFT demande un travail important de transformation de l'algorithme intolérant aux fautes. Les chercheurs se sont donc penchés sur les algorithmes les plus utilisés dans les domaines du calcul scientifique et du traitement de signal. Les plus importantes sont la transformée de Fourier rapide [105, 213, 224], le tri [57], les opérations matricielles (multiplication, inversion, décomposition LU et QR) [106, 143, 173, 122, 228], équations de Laplace, de Poisson [190, 154] ou encore les équations différentielles partielles [193]. Par ailleurs, les travaux de Aharonov et al.[3], utilisent l'ABFT pour construire des portes logiques tolérantes aux fautes pour les ordinateurs quantiques.

Cas du produit de matrices définies dans un corps fini Soit G la matrice génératrice d'un *code linéaire* $\mathcal{C}$ défini sur $\mathbb{F}_q$ considérons deux matrices A et B définies sur $\mathbb{F}_q$ telles que $C = A \times B$ dans $\mathbb{F}_q$ et ${}^TG \times A$ correspond à l'encodage de chaque colonne de la matrice A. Nous avons alors ${}^TG \times A \times B = ({}^TG \times A) \times B = {}^TG \times (A \times B)$ par associativité du produit de matrices. Ainsi, indépendamment de l'algorithme utilisé pour le produit de matrices, nous obtenons un ABFT : le codage des données d'entrée (${}^TG \times A$) implique le codage des données de sortie (${}^TG \times C$). La modification de l'algorithme consiste ni plus ni moins à opérer le produit de matrices avec une opérande (matrice A) de plus grande dimension. La i-ème ligne de la matrice encodée résultat, correspond exactement à la sortie d'une tâche f_i dans l'algorithme 9 présenté dans la section précédente.

Remarque 40 *L'exemple de produit de matrices tolérant aux fautes décrit ici impose que le corps de base du calcul soit un corps fini car un code linéaire est défini sur un tel corps et il est nécessaire que les produits de matrices soient cohérents (matrices génératrices et matrices opérandes).*

9.4.1.2 Méthodes

Les méthodes utilisées pour la construction d'ABFT ont suivi plusieurs idées maîtresses, le but étant toujours de détecter, localiser ou corriger le plus d'erreurs possibles avec un sur-coût algorithmique le plus faible possible.
Dans [100] Huang et Abraham considèrent le produit de matrices et ont proposé un codage par *contrôle de parité* selon les lignes et colonnes des matrices (*parity checksum*, en anglais). Par exemple, il s'agit d'ajouter une ligne à la matrice opérande A dont chaque élément est la somme des éléments de la colonne de A associée. Ceci correspond à un cas particulier de l'exemple présenté plus haut : la matrice génératrice G est telle que :

$$ {}^TG = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \\ 1 & \cdots & 1 \end{bmatrix} $$

La détection de l'erreur se fait par la vérification de la ligne de contrôle de parité sur la matrice résultat. Pour localiser et corriger une erreur il faut ajouter une colonne de contrôle de parité, créant ainsi un codage sur les deux dimensions de la matrice (i.e. *codage rectangle*).
La technique du contrôle de parité bi-dimensionnel est très efficace mais possède un taux de détection/correction très limité dans sa forme originale. Jou et Abraham [106] suivi de Anfinson et Luk [6], proposent d'utiliser des *contrôles de parité pondérés* (*weighted checksum*, en anglais) pour corriger 2 erreurs, étendus par Roy-Chowdhury et Banerjee [191, 192] à un nombre arbitraire d'erreurs. Les contrôles de parité pondérés correspondent en fait à l'utilisation de codes linéaires dans un anneau quelconque. Ils sont basés sur les travaux de Nair et Abaham [160], qui montrent qu'à tout code linéaire défini sur un corps fini, correspond un code linéaire dans les réels de même capacité de détection et correction. Pourtant il est à noter que, plongés dans les réels, les codes linéaires perdent leur *algorithme de décodage* puisque celui-ci repose bien souvent sur les propriétés particulières des corps finis. Notamment la construction pour la correction d'erreurs faite dans [192] utilise les codes de *Reed-Solomon* (voir section 3.2.3) mais ne peut décoder que lorsque les erreurs ont été préalablement localisées (i.e., cela revient à corriger des *effacements*).

La plus grande majorité des constructions d'ABFT repose sur les principes que nous venons d'énoncer pour la correction des erreurs. Les recherches se sont portés sur les problèmes d'erreur d'arrondis (*round-off*, en anglais) causés par :

- la manipulation de sommes de parité très grandes ; une solution est d'utiliser des résidus modulaires [106].
- la manipulation de valeurs flottantes. Les codes de contrôles de parité pondérés peuvent être adaptés en fonction des données qu'ils ont à coder pour minimiser la propagation de l'erreur [157, 160]. Pour les très grandes dimensions, il a été proposé le *contrôle de parité pondéré partitionné* [179], la donnée est partitionnée et, pour chaque sous-partie, un code adapté est utilisé.

Nous ne nous arrêterons pas dans ce mémoire sur le sujet du décalage (mentionné plus haut) entre le *modèle logique* et le *modèle physique* du système distribué. C'est l'un des points centraux de la recherche autour de l'ABFT et c'est dans ce but que Banerjee et Abraham [16] en 1986 proposent une méthode pour évaluer un ABFT sous un modèle de théorie des graphes. Ce modèle a été repris pour le diagnostic d'ABFT dans [189, 221, 161] et pour la construction d'ABFTs dans [158, 159, 90, 191, 192]. L'utilisation d'un modèle permet de chercher la meilleure architecture du système pour exécuter un ABFT donné ou de trouver le meilleur ABFT pour une architecture système donnée. Beaucoup de travaux se sont penchés sur l'automatisation de cette recherche d'optimum (i.e., développement de compilateurs générant des ABFT, par exemple [17]).

Pour finir cet état de l'art il faut mentionner le travail de Plank, Kim et Dongarra [169, 124], les premiers à proposer l'ABFT sur des réseaux de machines (*NOW*). Les systèmes distribués étudiés dans ce document sont tous des NOW, il est donc intéressant de remarquer que des constructions d'ABFT ont déjà été proposées pour ce genre de plate-formes. Nous noterons tout de même que les constructions proposées dans [169] correspondent plus à des techniques de reprise par sauvegarde que d'ABFT à proprement parler. En effet, les algorithmes (opérations matricielles) ne sont pas modifiés, il s'agit de sauvegarder les données intermédiaires du calcul, de façon distribuée, sur la mémoire locale des processeurs, la technique est donc nommée *sauvegarde sur mémoire volatile* [170] ("diskless checkpointing" en anglais).

9.4.2 Application au produit de matrices

Algorithme général Soient deux matrices A et B carrées à éléments dans un corps fini $\mathbb{F}_q$ de dimensions $(k \times k)$ et soit C leur produit : $C = A \times B$. Soient C_A et L_B deux codes linéaires, de formes respectives $[n, k, d_A]_q$ et $[m, k, d_B]_q$. Nous noterons $C_F = {}^T G_{C_A} \times A \times B \times G_{L_B}$, la *matrice encodée* de C par rapport aux codes (C_A, L_B) (les colonnes de C sont encodées dans C_A, les lignes de C sont encodées dans L_B). L'algorithme 11 décrit un ABFT général pour le produit de matrices.

Étude de l'ABFT L'algorithme 11 reprend de façon assez générale les ABFT proposés dans la littérature pour le produit de matrice. L'utilisation d'un code bi-dimensionnel (C est encodée suivant les lignes et les colonnes) permettant ainsi, avec un coût modéré, de détecter et corriger

Algorithme 11: ABFT pour produit de matrices

Entrée : Matrices A, B, matrices génératrices G_{C_A}, G_{L_B} des codes C_A et L_B

Sortie : C_F

Initialisation (encoder les colonnes de A et les lignes de B) sur R :

$$A_C \leftarrow {}^T G_{C_A} \times A$$
$$B_L \leftarrow B \times G_{L_B}$$

Traitement (produit de matrices distribué sur U) :

$$C_F \leftarrow A_C \times B_L$$

retourner C_F

plus d'erreurs : il est aisé de voir que la distance minimale du code qui à C lui associe C_F par rapport aux codes (C_A, L_B) est plus grande ou égale au produit des distances minimales de C_A et L_B.

Cet algorithme est très lié à la structure du produit de matrices, il ne sera utile qu'à l'algorithme 9 et restreindra son application à l'utilisation d'un code linéaire compatible avec le produit de matrice considéré. Les codes de Reed-Solomon se transposent bien pour travailler dans les entiers ou dans les rationnels, leur algorithme de décodage y garde toute son efficacité (nous les appelions alors *codes par interpolation de polynômes univariés*).

L'utilisation d'une construction ABFT telle que l'on vient de voir dans l'algorithme 9, permet de passer d'une complexité, sur les ressources U, de $W = O((k+\tau)k^\omega)\omega_q$ (cf. tableau 9.3) à $W = O((k+\tau)k^2)\omega_q$.

L'utilisation de codes de Reed-Solomon (proposée dans la section précédente) implique une taille minimale au corps fini $\mathbb{F}_q$: il faut que $q > k + \tau$. Les codes LDPC quand à eux peuvent être utilisés sans contrainte sur la taille du corps fini.

Remarque 41 *L'algorithme 6 itératif pour le produit de matrices présenté au début de ce chapitre se marie parfaitement à l'algorithme ABFT présenté ici. En effet, lorsque les colonnes de A sont des mots du code C_A et les lignes de B sont des mots du code L_B, il est facile de voir dans l'algorithme 6 que chaque matrice intermédiaire C_i est une* matrice encodée *par rapport au couple de codes (C_A, L_B). La figure 9.3 détaille les étapes de l'ABFT décrit dans l'algorithme 11 utilisant le produit de matrices par produit externe. A_C et B_L sont encodées par des codes systématiques et divisées respectivement en trois blocs de colonnes et lignes.*

9.5 Produit de matrices et codes par résidus

Nous avons vu que l'utilisation de codes correcteurs efficaces, d'un algorithme itératif adéquate et de techniques de certification de résultat permet d'adapter la construction générique ABFT pour

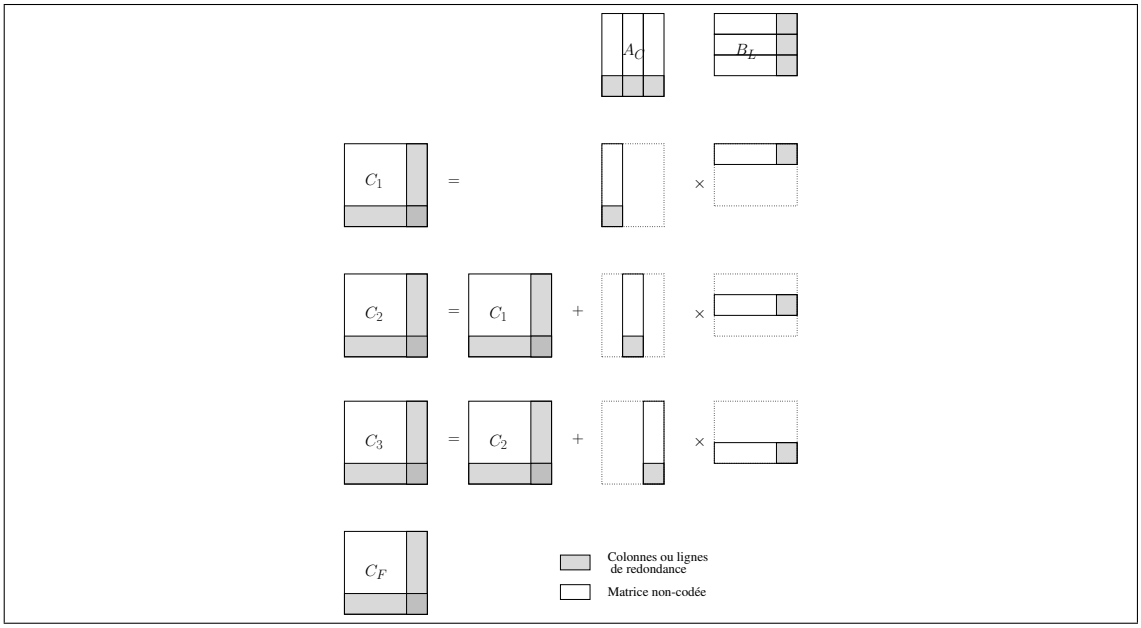

FIG. 9.3 – ABFT itératif pour le produit de matrices

le produit de matrices à des systèmes dont les modèles de fautes sont très contraignants.

Comme nous l'avons vu, la construction $ABFT$ proposée a un désavantage majeur : elle ne peut pas être utilisée de façon générique pour d'autres applications que le produit de matrice. Nous considérons ici une construction hybride entre ABFT (section 9.4) et réplication (section 9.3) qui pourra s'étendre facilement à un ensemble plus large d'applications. L'étude que nous faisons maintenant se base sur les principes des *systèmes de nombre résiduel* et *systèmes de nombre résiduel polynomiaux* (notés respectivement *RNS* et *PRNS* dans la suite) et les applications en théorie des codes qui en découlent : les *systèmes de nombre résiduel redondants* (notés *RRNS*) et les *systèmes de nombre résiduel polynomiaux redondants* (que nous noterons *RPRNS*). Après un rappel rapide du schéma général de tolérance aux fautes par nombre et polynômes résiduels nous verrons comment ils s'intègrent naturellement à l'algorithme 10 auto-tolérant aux fautes pour en améliorer les performances sur les ressources non-fiables U.

9.5.1 Schéma général et état de l'art

9.5.1.1 RNS et PRNS

Un *système de nombre résiduel* RNS est défini par un ensemble d'entiers naturels premiers entre eux deux à deux $(m_1, \cdots, m_n)$ que l'on nomme base RNS, notons $M = \prod_{i=1}^{n} m_i$. Soit un entier positif X plus petit que M, il peut être représenté dans la base RNS par l'ensemble $(x_1, \cdots, x_n)_{RNS}$ tel que, pour tout i, $x_i = X \mod m_i$.
D'après le *théorème des restes chinois*, il est possible de reconstruire X de façon unique à partir

de sa représentation dans la base RNS.

$$X = \sum_{i=1}^{n} M_i M_i^{-1} x_i \mod M$$

où $M_i = \frac{M}{m_i}$ et M_i^{-1} est l'inverse de M_i modulo m_i : $M_i M_i^{-1} = 1 \mod m_i$.

Cette représentation des entiers a été beaucoup étudiée et utilisée pour l'exécution parallèle de calculs arithmétiques sur des processeurs ayant une taille de registre limitée. En effet, si l'on considère deux entiers A et B représentés dans la base RNS et tels que $0 \leq A \star B < M$, où $\star$ représente l'addition, la soustraction ou la multiplication, alors l'opération peut être effectuée, de façon indépendante, sur chaque résidu :

$$\begin{aligned} C &= A \star B \\ &= (a_1 \star b_1 \mod m_1, \cdots, a_n \star b_n \mod m_n)_{RNS} \end{aligned}$$

Ainsi, chaque opération modulaire indépendante travaille sur des entiers de petite taille ($log(m_i)$ bits) alors que l'opération usuelle manipule des données sur $log(C)$ bits.

De la même façon que dans les entiers, les PRNS, travaillent sur des polynômes définis dans les corps finis. Soit $(m_1(X), \cdots, m_n(X))$ un ensemble de polynômes définis sur $\mathbb{F}_q$ premiers entre eux et $M(X) = \prod_{i=1}^{n} m_i(X)$. Soit $P(X) \in \mathbb{F}_q[X]$, tel que $deg(P) < deg(M)$, considérons sa représentation dans la base PRNS $(m_1(X), \cdots, m_n(X))$: $(p_1(X), \cdots, p_n(X))_{PRNS}$ où, pour tout i, $p_i(X) = P(X) \mod m_i(X)$. D'après le théorème des restes chinois dans $\mathbb{F}_q[X]$, il existe un unique polynôme de $\mathbb{F}_q[X]$ de degré inférieur à $deg(M)$ représenté par $(p_1(X), \cdots, p_n(X))$ et

$$P(X) = \sum_{i=1}^{n} M_i(X) M_i^{-1}(X) p_i(X) \mod M(X)$$

où $M_i(X) = \frac{M(X)}{m_i(X)}$ et $M_i^{-1}(X)$ est l'inverse de $M_i(X)$ modulo $m_i(X)$: $M_i(X) M_i^{-1}(X) = 1 \mod m_i(X)$.

9.5.1.2 RRNS et RPRNS

Un des intérêts essentiels des RNS et PRNS est de construire une représentation redondante des nombres et des polynômes. C'est à dire être capable de détecter et potentiellement de corriger lorsque certaines composantes de la représentation RNS (resp. PRNS) d'un entier (resp. d'un polynôme) sont erronées. Si l'on considère une base RNS $(m_1, \cdots, m_n)$ et un entier X tel que $X < \prod_{i=1}^{k} m_i$, où $k < n$, alors la représentation de X dans la base RNS $(x_1, \cdots, x_n)_{RNS}$ est redondante. En effet, si la remontée de X par le théorème des restes chinois donne un entier $\widetilde{X}$ tel que $\prod_{i=1}^{k} m_i < \widetilde{X} < \prod_{i=1}^{n} m_i$, alors la représentation de X possède des erreurs.

Cette idée a motivé la recherche autour de la construction de codes correcteurs résiduels, donnant naissance aux codes *RRNS* et *RPRNS*. Entre autres, on peut citer les travaux de Stone [208] et de Shiozaki [203] pour les codes par résidus polynomiaux et les travaux de Mandelbaum [145],

Krishna et al. [131, 210, 132] et de Yang et Hanzo [227, 140] sur les codes par entiers résiduels. Ces travaux de recherche se sont concrétisés par une théorie complète des codes par résidus, en particulier il a été montré que ces codes admettent une distance minimale maximale et sont étroitement liés aux codes de Reed-Solomon. En fait les codes RPRNS sont une généralisation des codes de Reed-Solomon et des codes par interpolation de polynômes univariés pour lesquels nous connaissons des algorithmes de codage et décodage efficaces (cf. sections 3.2.3 et 3.2.4).

9.5.1.3 Algorithmes efficaces de codage et décodage des codes RRNS et RPRNS

Pour un choix d'une base RPRNS telle que les $m_i(X)$ sont de degré 1, on se retrouve dans le cas des codes par interpolation polynomiale, pour lesquels nous avons vu un algorithme de décodage très efficace à base d'une généralisation de la transformée de Fourier rapide (voir section 3.2.4). Ces codes ont été présentés comme une généralisation des codes de Reed-Solomon, à tout ensemble de polynômes à coefficients dans un corps commutatif.

Nous allons donner maintenant une idée du lien entre les codes RRNS et les codes de Reed-Solomon et en déduire un algorithme de codage et décodage efficace.

L'évaluation et l'interpolation de polynômes peuvent être vu comme un cas particulier du théorème des restes chinois pour les polynômes. Un simple changement d'espace permet de passer des codes de Reed-Solomon aux codes de nombres résiduels redondant : lorsque RS correspond à l'anneau des polynômes à coefficients dans un corps fini, RRNS correspond à l'anneau des entiers relatifs. L'idée est de transposer les algorithmes de codage/décodage efficaces pour les codes RS dans les codes RRNS.

Considérons l'algorithme de codage et décodage présenté pour les codes de Reed-Solomon (section 3.2.3), le codage correspond exactement à la décomposition dans une base modulaire, le décodage est, quand à lui, la composition de deux algorithmes : un remontée par le théorème des restes chinois et une division euclidienne étendue.

La seule contrainte pour appliquer ces algorithmes est de travailler dans un anneau euclidien (ce qui est le cas de l'anneau des entiers relatifs). Il reste à prouver que l'algorithme de décodage se termine et donne le résultat escompté : i.e. énoncer et prouver le théorème 3 (de la section 3.2.3) dans les entiers relatifs.

Soit $(m_1, \cdots, m_n)$ une base RNS rangée en ordre croissant ($m_1 < \cdots < m_n$) et un entier X, tels que $0 \leq X < \prod_{i=1}^{k} m_i$, où $k < n$ et notons $M = \prod_{i=1}^{n} m_i$ et $(x_1, \cdots, x_n)$ la représentation de X dans la base RNS. Soit $(y_1, \cdots, y_n)$ la représentation bruitée de X dans la base RNS (on suppose que $n = k + 2t$ et que le nombre de composantes y_i erronées est inférieur ou égal à t). On peut reconstruire Y à partir de $(y_1, \cdots, y_n)$ par remontée du théorème des restes chinois, nous avons alors $0 \leq Y < M$. Notons $M_V = \prod_{y_i = x_i} m_i$ et $M_F = \prod_{y_i \neq x_i} m_i$.
Le théorème 3 s'énonce ainsi :

Théorème 11 *$M_F \cdot X$ apparaît dans la suite des restes de l'algorithme d'Euclide appliqué à M et Y. De plus, il s'agit du premier reste inférieur strictement à $\prod_{i=1}^{k+t} m_i$.*

Nous ne démontrerons pas ce théorème ici, la preuve correspond à la démonstration de Mandelbaum dans [145].

Maintenant que nous avons explicité un algorithme de décodage pour les codes RRNS, nous admettrons qu'il existe des algorithmes efficaces pour l'exécution de la remontée de restes chinois et l'algorithme d'Euclide étendu dans les entiers relatifs. En fait, il est possible d'exhiber des algorithmes de complexité comparables aux codes par interpolation polynomiale. La remontée des restes chinois peut se faire en $O(n \log^2 n)$ opérations (avec $O(n \log^2(n))$ précalculs) et une version du PGCD rapide existe dans les entiers en complexité $W = O(W_{\mathbb{Z}}(\log M) \log\log(M))$[1], où $W_{\mathbb{Z}}(\log M)$ correspond au coût d'un produit d'entiers sur $\log M$ bits. L'implémentation et même la description du PGCD rapide dans les entiers sont souvent considérées comme délicates dans la littérature.

Si l'on considère que $W_{\mathbb{Z}}(x) = O(x \log x \log\log x)$ et $M < (m_n)^n$, on obtient, pour l'algorithme de décodage, $\underline{W = \widetilde{O}(n \log m_n \log^2(n \log m_n))}$.

On rappelle que pour les codes par interpolation polynomiale il est possible de trouver les coefficients de Bezout recherchés directement par résolution d'un système linéaire, avec un algorithme tel que $W = O(n^2 \log n)\omega_q$ et $D = O(\log^3 n)\delta_q$ (cf. section 9.2.3.2). Cet algorithme peut être bien plus intéressant que l'algorithme d'Euclide lorsque le calcul se fait sur une machine parallèle (puisque la profondeur D est faible). A notre connaissance, une telle solution n'existe pas dans les entiers, nous devrons donc utiliser l'algorithme d'Euclide : $W = \widetilde{O}(n \log m_n \log^2(n \log m_n))$ et $\underline{D = \widetilde{O}(n \log m_n \log(n \log m_n))}$.

9.5.2 Codes par résidus pour le produit de matrices sur des systèmes de calcul globaux

L'utilisation de codes par résidus a permis la construction d'applications tolérantes aux fautes principalement dans les domaines de traitement du signal. Notons par exemple les travaux de Beckmann et Musicus [19] en 1993 sur la convolution linéaire (pour le produit de polynômes) tolérante aux fautes à l'aide de RPRNS. Il est intéressant de remarquer que l'algorithme tolérant au fautes le plus efficace décrit dans [19] pour leur problème utilise une base bien particulière de façon à optimiser, entre autres, la phase de remontée par le théorème des restes chinois, qui revient alors à exécuter une FFT. Il s'agit en fait d'une interpolation polynomiale : la FFT accélère l'évaluation et l'interpolation sur des puissances successives d'une racine n-ième de l'unité.

L'idée, comme pour une construction ABFT, est d'exprimer les données d'entrée du calcul sous forme encodée (dans une base de résidus) et de modifier les opérations du calcul pour qu'elles conservent cette base. Comme les opérations sont indépendantes sur chaque composante de la représentation par résidus, nous obtenons un algorithme facilement décomposable en tâches indépendantes (très proche de l'algorithme 10).

[1] voir, par exemple, le cours de Bruno Salvy sur le sujet : `http://algo.inria.fr/salvy/mpri/Cours13.pdf`

Les opérations sur des entiers (resp. polynômes à coefficients dans un corps fini) peuvent être utilisées directement dans une base RNS (resp. dans une base PRNS). Pour les opérations dans un corps fini il est possible d'utiliser un algorithme de produit dans les corps finis et une base de résidus adaptés (par exemple, voir les travaux de J.C. Bajard [15] qui se basent l'algorithme de multiplication de Montgomery).
L'algorithme 12 décrit un algorithme auto-tolérant aux fautes à l'aide de codes par résidus pour le produit de matrices définies dans un espace fini E de taille q. Il reprend le schéma de l'algorithme 10 auto-tolérant aux fautes sur les composantes des matrices. I il s'agit donc de représenter les composantes des matrices opérandes A et B dans une base de résidus adaptée à E ; puis d'exécuter le produit de matrice de façon à ce que les opérations d'addition et de multiplication dans E conservent la représentation résiduelle (E sera donc un sous-ensemble des entiers, de l'ensemble des polynômes à coefficient dans un corps fini ou un corps fini). Nous reprendrons les notations de l'algorithme 10 : la base de résidus est de taille $n = v + \tau$ et les moduli m_i sont tels que $q \leq \prod_{i=1}^{v} m_i$. Nous noterons $u = \max_i m_i$. Les sous-tâches f_i exécutées sur les ressources U sont au nombre de $v + \tau$.

Algorithme 12: Algorithme auto tolérant aux fautes à l'aide de codes par résidus.

Entrée : Deux matrices A et B, une base de résidus $(m_1, \cdots, m_n)$ redondante pour les éléments de E.
Sortie : $C = A \times B$ dont chaque composante est encodée dans dans la base de résidus.

Sur les ressources R :

Rendre accessible en lecture aux ressources U les matrices A et B stockées sur les ressources R;

Sur les ressources U :

Exécuter, de façon indépendante pour $0 < i \leq n$, f_i :
- $A_i = A \mod m_i$ et $B_i = B \mod m_i$;
- $C_i = A_i \times B_i$ à l'aide d'opérations conservant la base de résidus;
- Écrire C_i dans la zone de confiance;

Sur les ressources R :

Récupération, de $(C_1, \cdots, C_{v+\tau})$ (C est récupérée par décodage de chaque composante dans la base de résidus);

Algorithme générique L'algorithme 12 est équivalent à l'algorithme 10 sur les machines fiables R en utilisant un code par résidus et avec complexité équivalente aux codes par interpolation de polynômes univariés. L'utilisation d'un code par résidus adapté permet d'encoder directement les entrées et de travailler dans la base de résidus, le travail total sur les machines non-sûres en est amélioré, il passe de $W = O((v + \tau)k^\omega)\omega_q$ à $W = O((v + \tau)k^\omega)\omega_u$.

Un avantage certain d'utiliser des codes par résidus au lieu d'une construction ABFT à l'aide de code linéaires, est qu'elle s'adapte plus facilement à un ensemble plus large d'applications. Il

faut tout de même insister sur le fait qu'elle impose quelques contraintes (en comparaison à l'application directe de l'algorithme 10) : trouver une base de résidus qui ait à la fois des algorithmes de codage et décodage efficace (si l'on veux pouvoir tolérer beaucoup d'erreurs : τ grand) et être capable de modifier les opérations arithmétiques de façon à les faire dans base de résidus.

D'après les résultats que nous venons de voir pour les codes RRNS et RPRNS, nous pouvons en conclure qu'ils seront particulièrement intéressants pour les applications qui manipulent des valeurs intermédiaires très grandes et qui ne possèdent pas d'algorithme efficace d'impact 1. Parmi de telles applications, nous pouvons citer le déterminant de matrices. Un tel système auto-tolérant aux fautes a été testé pour le calcul du déterminant de matrices de grandes dimensions sur la plateforme expérimentale de grille Grid5000[2] par Thomas Stalinski et Majid Khonji dans le cadre de leur stage de Master[3].

9.6 Synthèse des résultats

9.6.1 Récapitulation des résultats

Le tableau 9.4 reprend les coûts des algorithmes auto-tolérants efficaces proposés dans les dernières sections. Nous en déduisons, pour le produit de matrices, des algorithmes de complexité équivalente ou plus faible (pour l'algorithme 9 par ABFT et l'algorithme 10 par RRNS/RPRNS) que la réplication pure sur les ressources U. Nous avons donc bien réussi a construire des algorithmes tolérants aux fautes à l'aide de codes correcteurs efficaces qui, pour un coût algorithmique équivalent ou plus faible à la réplication pure de tâches sur l'ensemble du système de calcul global $\{U, R\}$, offre une capacité de tolérance aux fautes bien plus intéressante et une gestion plus fine du taux de fautes à tolérer.

9.6.2 Application de l'algorithme ABFT pour le produit de matrices sur réseau pair-à-pair

Dans la section 9.4 a été présentée une construction ABFT visant à la tolérance aux fautes pour un calcul de produit de matrices dans les systèmes distribués de grande taille, cible de nos travaux de recherche. Nous allons présenter maintenant une application de cette construction dans le cas des réseaux pair-à-pair. A notre connaissance, il s'agit de la première étude de tolérance aux fautes pour les réseaux pair-à-pair basée sur les ABFT.

Nous considérons un réseau pair-à-pair, bien que nous ne fixions pas une hiérarchie particulière entre les différents pairs du système, il n'est pas interdit qu'il en existe une. Ainsi le protocole que nous présentons, illustré par la figure 9.4, peut être appliqué sur tout type de système pair-à-pair, depuis les *réseaux pair-à-pair purs*, complètement homogènes, comme *Freenet*[4], aux architec-

[2] `https://www.grid5000.org`
[3] Master SCCI 2009, Ensimag, Grenoble, France
[4] `http://freenetproject.org/`

		Zone de Confiance (R) Décodage	Zone non sûre (U) Produit de matrices
Stockage $k^2 \log q$	Référence	$W = O(k^\omega)\omega_q$ $D = O(\log k)\delta_q$	0 0
$\mathcal{C}$=$(dk^2, k^2, d)_E$ Stock. $dk^2 \log q$	Réplication	$W = k^2 \cdot O(d)\omega_q$ $D = \delta_q$	$W = d \cdot O(k^\omega)\omega_q$ $D = O(\log k)\delta_q$
Alg. 8 $\mathcal{C}$=$(n, k^2, d)_E$ $n = k^2 + \tau$ Stock. $(k^2 + \tau) \log q$	Reed-S. (systématique)	$W = O((k^2 + \tau) \log^2(k^2 + \tau))\omega_q$ $D = O(\log^3(k^2 + \tau))\delta_q$	$\underline{W = O(k^3 + \tau k^\omega)\omega_q}$ $D = O(\log k)\delta_q$
	LDGM (creux : $s < \frac{n}{2}$)	$W = O((k^2 + \tau))\omega_q$ $D = O(??)\delta_q$	$\underline{W = (k^2 + \tau) \cdot O(sk)\omega_q}$ $D = O(\log k)\delta_q$
Alg. 9 $\mathcal{C}$=$(n, k, d)_E$ $n = k + \tau$ Stock. $k(k + \tau) \log q$	Reed-S. (ABFT)	$W = k \cdot O((k + \tau) \log^2(k + \tau))\omega_q$ $D = O(\log^3(k + \tau))\delta_q$	$\underline{W = (k + \tau) \cdot O(k^2)\omega_q}$ $D = O(\log k)\delta_q$
	LDPC (ABFT)	$W = k \cdot O((k + \tau))\omega_q$ $D = O(??)\delta_q$	
Alg. 10 $(m_1, \cdots, m_n)_I$ $n = v + \tau$ Stock. $k^2(v + \tau) \log u$	RPRNS	$\underline{W = k^2 \cdot O((v + \tau) \log^2(v + \tau))\omega_u}$ $\underline{D = O(\log^3(v + \tau))\delta_u}$	$\underline{W = (v + \tau) \cdot O(k^\omega)\omega_u}$ $\underline{D = O(\log k)\delta_u}$
	RRNS	$\underline{W = \widetilde{O}(n \log m_n \log^2(n \log m_n))}$ $\underline{D = \widetilde{O}(n \log m_n \log(n \log m_n))}$	

TAB. 9.4 – Complexité des algorithmes auto-tolérants aux fautes optimisés

tures les plus structurées comme *Chord*[5] [207] ou *folding@home*[6] par exemple. Un des attributs essentiels d'un réseau pair-à-pair est son efficacité à transmettre les données et les répartir de façon distribuée (et redondante) sur l'ensemble des ressources du système. Nous ferons l'hypothèse que le réseau est doté de protocoles efficaces pour l'envoi, la recherche et la répartition de données.

Produit de matrices tolérant aux fautes Soit un utilisateur U (cf. figure 9.4) soumettant un produit de matrices (nous considérerons les matrices déjà encodées). La version par produit externe du produit de matrices nous permet de découper le calcul en itérations, et donc de corriger d'éventuelles erreurs et de récupérer des blocs manquants en cours de calcul. En cela le protocole est proche d'une technique de *reprise par sauvegarde*, sauf que chaque itération tolère un nombre d'erreurs fixé par le choix du code correcteur. Ainsi l'utilisateur U commence par envoyer sur le réseau pair-à-pair les matrices A et B pré-encodées accompagnées des différentes tâches de calcul à réaliser (elles-mêmes reliées à des blocs de données). Les données se répartissent à travers le système et les ressources exécutent les tâches sur les données qu'elles ont récupérées localement.

[5] `http://pdos.csail.mit.edu/chord/`
[6] `http://folding.stanford.edu/`

La découpe en tâches peut se faire statiquement par l'utilisateur U ou dynamiquement lors du calcul. En effet, comme nous l'avons vu, l'algorithme du produit de matrices peut être re-découpé de façon dynamique : une tâche ayant pour objet de calculer le produit externe de nb colonnes de A par nb lignes de B peut être, à volonté, re-divisé en sous-tâches qui calculeront, par exemple, le produit d'une colonne par une ligne, elles-mêmes peuvent être divisées en sous-tâches qui traiteront un morceau de colonne et de ligne. Plusieurs techniques de répartition de charges dynamiques sont connues et très efficaces pour les réseaux pair-à-pair, par exemple, l'utilisation de *tables de hachage distribuées* [172] pour les réseaux structurés ou le vol de travail [40, 48].
La *zone de confiance* est limitée aux ressources de l'utilisateur U, qui récupérera les différents blocs de matrice calculés par les ressources du système et, lorsque suffisamment auront été reçus, il tentera de corriger la matrice intermédiaire et de certifier la matrice corrigée. Dans le cas où la certification est passée, le calcul continue son cours sans action de U, il conserve l'"état stable" qu'il vient de constituer. Si la certification n'est pas passée, le calcul est interrompu et sera réinitialisé à partir du dernier "état stable" possédé par U.

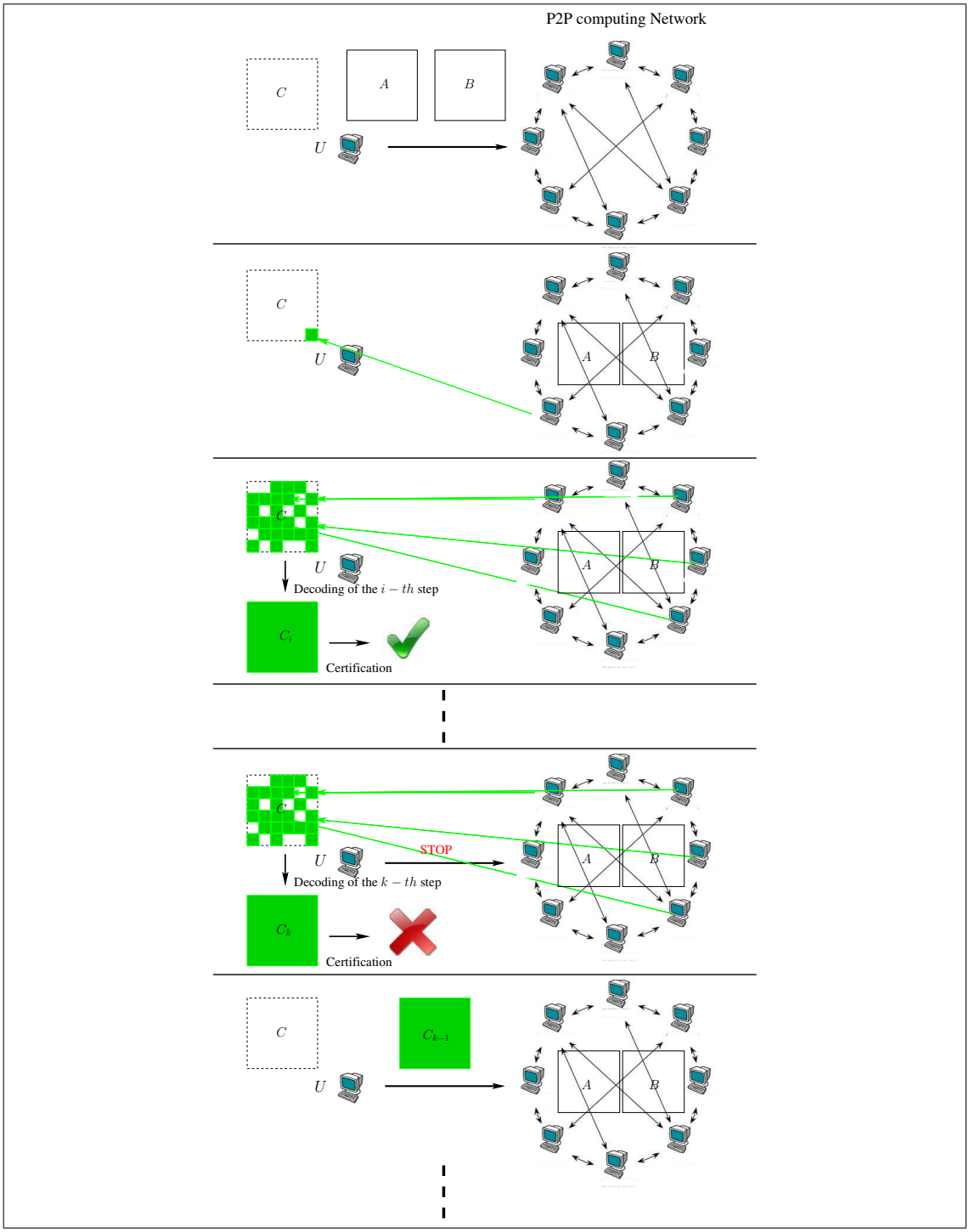

FIG. 9.4 – Protocole de tolérance aux fautes pour le produit de matrices sur réseau pair-à-pair

Conclusion 10

Ce chapitre clôt le mémoire de thèse en rappelant les objectifs, les points abordés et les ouvertures qui nous semblent pertinentes tant au niveau industriel qu'au niveau de la recherche académique.

Les travaux présentés dans ce mémoire touchent à des domaines variés de la cryptographie à clé secrète et de la sûreté de fonctionnement des systèmes. Les objectifs étaient de deux ordres :

- L'étude approfondie des chiffres par blocs, au niveau de leur sécurité mais aussi de leur performance dans le but de leur intégration dans un système distribué hétérogène.
- L'étude de la tolérance aux fautes des systèmes distribués des grande taille, en nous concentrant sur les plate-formes de calcul de type pair-à-pair ou grille de grappes.

Synthèse des sujets abordés : conclusions et perspectives

Sur l'ensemble de ce mémoire, les sujets suivants ont été abordés :

- **La parallélisation de fonctions de chiffrement symétriques** et de hachage via leur modes opératoires. L'utilisation d'un ordonnancement de tâches adaptatif par vol de travail nous a permis d'obtenir des performances proches de l'optimum, aussi bien en théorie qu'en pratique, dans un environnement hétérogène d'unités de calcul et de masquer le coût de la fonction cryptographique dans le traitement d'un flux de données.
  Perspectives : le développement d'une solution réseau haut débit sécurisée faisant usage d'un ensemble d'unités de calcul dédiées pour le chiffrement des données. C-S est porteur d'un projet *Minalogic* nommé SHIVA, auquel participe l'INRIA, qui a pour vocation la construction d'une telle solution ;
- L'étude de la sécurité des chiffres par bloc : tout d'abord la **cryptanalyse par différentielles impossibles**. L'utilisation d'un modèle original englobant les chiffres de Markov nous a permis d'évaluer, sous certaines conditions, la résistance d'un chiffre de type PSN ou SPN par rapport à ces attaques. Sous ces conditions, il a pu être montré que les chiffres CS-Cipher et AES sont immunisés contre de telles attaques.
  Perspectives : Les résultats sont conditionnés par une hypothèse qu'il reste à vérifier : les chiffres de Markov étudiés ne s'éloignent-ils pas trop du modèle théorique de chiffre de Markov en Support ? Des techniques numériques existent, dans le domaine de l'agrégation de chaînes de Markov, permettant d'évaluer cet éloignement. Il serait intéressant de voir ce qu'elles peuvent apporter ;
- L'étude de la sécurité des chiffres par bloc : les **attaques par canaux cachés**. Ces attaques sont particulièrement efficaces mais sont encore, à l'heure actuelle, restées à un stade heu-

ristique et expérimental. Le développement d'attaques de ce type aide a mieux comprendre comment et quand ces attaques fonctionnent, dans l'espoir de construire des contre-mesures robustes et efficaces qui font défaut aujourd'hui. L'attaque que nous présentons permet de tisser un nouveau lien entre les attaques par canaux cachés et la cryptanalyse conventionnelle (ici la cryptanalyse multi-linéaire) et d'observer d'un autre point de vue l'utilité des codes correcteurs dans le domaine des attaques par canaux cachés.
Perspectives : Cette attaque ouvre un champ large de recherche sur les attaques par canaux cachés entre approximations linéaires et mise en équations d'un chiffre symétrique (attaques algébriques). Plusieurs perspectives ont été proposées dans le chapitre, la plus intéressante étant sûrement de s'attaquer à la recherche et l'utilisation d'approximations de plus hauts degrés : un attaque intermédiaire donc entre ce que nous proposons et les attaques algébriques par canaux cachés ;

– L'étude du chiffre CS-Cipher, sa sécurité, son intégration et son dimensionnement n'ont pas été présentés ici, ils constituent une partie confidentielle dans nos recherches. Seule, une présentation rapide du chiffre original de Stern et Vaudenay a été faite.
– Le développement de solutions algorithmiques pour rendre le calcul de produit de matrices tolérant aux pannes dans un systèmes distribué ouvert, non seulement contraint à des pannes franches mais aussi des pannes par valeurs et de type byzantines, a donné des résultats probants : l'utilisation de codes correcteurs efficaces tels que les codes de Reed-Solomon ou les codes "Low Density Parity Check" a permis de développer des **algorithmes tolérants un grand nombre de pannes** sans surcoût significatif par rapport à la méthode classique de réplication de tâches. Nous avons vu que l'optimisation des performances se faisait au détriment de la généricité de la méthode : l'ABFT construit est entièrement dépendant du calcul choisi. D'un autre côté, l'utilisation de systèmes de nombre résiduel, permet de garder une certaine généricité de la construction tout en apportant une capacité de tolérance aux fautes importante dans le cas d'attaques malicieuses et une optimisation significative du calcul.
Perspectives : La mise en place ou la simulation de telles techniques de tolérance aux fautes pour des modèles de fautes très contraignants est la prochaine étape de ce travail. Dans le cadre d'un projet de Master, des étudiants ont mis en place une telle architecture de tolérance aux fautes à l'aide d'une base de nombre résiduels pour le calcul du déterminant de matrices de grandes dimensions. Ce travail peut être continué selon différentes directions : le développement efficace d'un algorithme de décodage dans une base RNS sur machines parallèles, l'intégration d'une telle technique de tolérance aux fautes générique directement dans un intergiciel tel que Kaapi et la comparaison et le couplage à des techniques plus classiques sur de telles plate-formes (e.g., *reprise par sauvegarde*, réduite à de pannes franches).

Bibliographie

[1] Nist pub 800-38a 2001 ed, recommendation for block cipher modes of operation – methods and techniques. Rapport technique, December 2001.

[2] Security evaluation of nessie first phase. Rapport technique, NESSIE, September 23, 2001.

[3] Dorit AHARONOV et Michael BEN-OR : Fault-tolerant quantum computation with constant error rate. *SIAM Journal on Computing*, 38(4):1207–1282, 2009.

[4] Mehdi-Laurent AKKAR et Christophe GIRAUD : An implementation of des and aes, secure against some attacks. *In* Çetin KAYA KOÇ, David NACCACHE et Christof PAAR, éditeurs : *CHES*, volume 2162 de *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

[5] Ross J. ANDERSON, Eli BIHAM et Lars R. KNUDSEN : The case for serpent. *In AES Candidate Conference*, pages 349–354, 2000.

[6] C.J. ANFINSON et F.T. LUK : A linear algebraic model of algorithm-based fault tolerance. *Computers, IEEE Transactions on*, 37(12):1599–1604, Dec 1988.

[7] Kazumaro AOKI, Tetsuya ICHIKAWA, Masayuki KANDA, Mitsuru MATSUI, Shiho MORIAI, Junko NAKAJIMA et Toshio TOKITA : Camellia : A 128-bit block cipher suitable for multiple platforms - design and analysis. *In* Douglas R. STINSON et Stafford E. TAVARES, éditeurs : *Selected Areas in Cryptography*, volume 2012 de *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000.

[8] Kazumaro AOKI et Kazuo OHTA : Strict evaluation of the maximum average of differential probability and the maximum average of linear probability (special section on cryptography and information security). *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(1):2–8, 19970125.

[9] Jean ARLAT, Yves CROUZET, Yves DESWARTE, Jean-Charles FABRE, Jean-Claude LAPRIE et David POWELL : *Tolérance Aux Fautes*, pages 241–270. Les Editions Vuibert, J.Akoka, I.Comyn-Wattiau (Eds), 2006. ISBN 2-7117-4846-4.

[10] Alexei E. ASHIKHMIN et Simon LITSYN : Fast decoding algorithms for first order reed-muller and related codes. *Des. Codes Cryptography*, 7(3):187–214, 1996.

[11] A. AVIZIENIS : Fault-tolerance : The survival attribute of digital systems. *Proceedings of the IEEE*, 66(10):1109–1125, Oct. 1978.

[12] Algirdas AVIZIENIS, Jean-Claude LAPRIE et Brian RANDELL : Dependability and its threats - a taxonomy. *In* René JACQUART, éditeur : *IFIP Congress Topical Sessions*, pages 91–120. Kluwer, 2004.

[13] Algirdas AVIZIENIS, Jean-Claude LAPRIE, Brian RANDELL et Carl E. LANDWEHR : Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[14] Mustafa BADAROGLU, Kris TIRI, Stéphane DONNAY, Piet WAMBACQ, Hugo De MAN, Ingrid VERBAUWHEDE et Georges G. E. GIELEN : Clock tree optimization in synchronous cmos digital circuits for substrate noise reduction using folding of supply current transients. *In DAC*, pages 399–404. ACM, 2002.

[15] Jean Claude BAJARD : A Residue Approach of the Finite Fields Arithmetics. *In The Asilomar Conference on Signals, Systems, and Computers*, page 5, Asilomar Conference Center California US, 2007. IEEE conferences publishing.

[16] Rrithviraj BANERJEE et Jacob A. ABRAHAM : Bounds on algorithm-based fault tolerance in multiple processor systems. *IEEE Trans. Comput.*, 35(4):296–306, 1986.

[17] P. BANJEREE, Prithviraj BANERJEE, Vijay BALASUBRAMANIAN et Amber ROYCHOWDHURY : Compiler assisted synthesis of algorithm-based checking in multiprocessors. *In In Foundations of Dependable Computing, Vol III : System Implementation*, pages 159–211. Kluwer Academic Publishers, 1994.

[18] Gregory V. BARD, Nicolas T. COURTOIS et Chris JEFFERSON : Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf(2) via sat-solvers. Cryptology ePrint Archive, Report 2007/024, 2007. `http://eprint.iacr.org/`.

[19] P.E. BECKMANN et B.R. MUSICUS : Fast fault-tolerant digital convolution using a polynomial residue number system. *Signal Processing, IEEE Transactions on*, 41(7):2300–2313, Jul 1993.

[20] Mihir BELLARE, Anand DESAI, Eron JOKIPII et Phillip ROGAWAY : A concrete security treatment of symmetric encryption. *In FOCS '97 : Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, page 394, Washington, DC, USA, 1997. IEEE Computer Society.

[21] Michael A. BENDER et Michael O. RABIN : Online scheduling of parallel programs on heterogeneous systems with applications to cilk. *Theory of Computing Systems Special Issue on SPAA*, 35:2002, 2002.

[22] Julien BERNARD : *Observation et analyse d'applications embarquées multiprocesseurs en vue d'une meilleure répartition de charge*. Thèse de doctorat, Laboratoire Informatique et Distribution de Grenoble, France, décembre 2008.

[23] Xavier BESSERON : *Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle*. Thèse de doctorat, Laboratoire Informatique et Distribution de Grenoble, France, 2010.

[24] A. BIENNER et B. DEQUIDT : Implémentations parallèles de md6, une fonction de hachage cryptographique moderne. Projet Intensif Ensimag 2ème Année, 2009. `http://moais.imag.fr/membres/jean-louis.roch/perso_html/transfert/2009-06-19-IntensiveProjects-M1-SCCI-Reports/BiennerDequidt_fr.pdf`.

[25] Eli BIHAM : New types of cryptoanalytic attacks using related keys (extended abstract). *In EUROCRYPT*, pages 398–409, 1993.

[26] Eli BIHAM : A fast new des implementation in software. pages 260–272. Springer-Verlag, 1997.

[27] Eli BIHAM, Alex BIRYUKOV et Adi SHAMIR : Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. *Lecture Notes in Computer Science*, 1592:12–23, 1999.

[28] Eli BIHAM, Orr DUNKELMAN et Nathan KELLER : The rectangle attack - rectangling the serpent. *In* Birgit PFITZMANN, éditeur : *EUROCRYPT*, volume 2045 de *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[29] Eli BIHAM et Adi SHAMIR : Differential cryptanalysis of des-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[30] Eli BIHAM et Adi SHAMIR : Differential cryptanalysis of the full 16-round des. *In* Ernest F. BRICKELL, éditeur : *CRYPTO*, volume 740 de *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[31] Eli BIHAM et Adi SHAMIR : Differential fault analysis of secret key cryptosystems. *In CRYPTO '97 : Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 513–525, London, UK, 1997. Springer-Verlag.

[32] Dario BINI et Victor Y. PAN : *Polynomial and matrix computations (vol. 1) : fundamental algorithms*, pages 228–311. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.

[33] Alex BIRYUKOV, Christophe De CANNIÈRE et Michaël QUISQUATER : On multiple linear approximations. *In* Matthew K. FRANKLIN, éditeur : *CRYPTO*, volume 3152 de *Lecture Notes in Computer Science*, pages 1–22. Springer, 2004.

[34] Alex BIRYUKOV et David WAGNER : Slide attacks. *In* Lars R. KNUDSEN, éditeur : *FSE*, volume 1636 de *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[35] Alex BIRYUKOV et David WAGNER : Advanced slide attacks. *In EUROCRYPT*, pages 589–606, 2000.

[36] Richard E. BLAHUT : *Theory and Practice of Error Control Codes*. Addison-Wesley.

[37] Manuel BLUM et Sampath KANNAN : Designing programs that check their work. *In STOC*, pages 86–97. ACM, 1989.

[38] Manuel BLUM, Michael LUBY et Ronitt RUBINFELD : Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[39] Robert D. BLUMOFE, Christopher F. JOERG, Bradley C. KUSZMAUL, Charles E. LEISERSON, Keith H. RANDALL et Yuli ZHOU : Cilk : An efficient multithreaded runtime system. *In Journal of Parallel and Distributed Computing*, pages 207–216, 1995.

[40] Robert D. BLUMOFE et Charles E. LEISERSON : Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, 1999.

[41] G. BOLCH, S. GREINER, H. de MEER et K. S. TRIVEDI : *Queueing Networkds and Markov Chains. Modeling and Performance Evaluation with Computer Science Evaluations*. Wiley-Interscience, 1998.

[42] Dan BONEH, Richard A. DEMILLO et Richard J. LIPTON : On the importance of checking cryptographic protocols for faults (extended abstract). *In EUROCRYPT*, pages 37–51, 1997.

[43] George BOSILCA, Remi DELMAS, Jack DONGARRA et Julien LANGOU : Algorithmic based fault tolerance applied to high performance computing. *CoRR*, abs/0806.3121, 2008.

[44] Robert S. BOYER et J. Strother MOORE : *The Correctness Problem in Computer Science*. Academic Press, Inc., Orlando, FL, USA, 1982.

[45] Eric BRIER, Christophe CLAVIER et Francis OLIVIER : Correlation power analysis with a leakage model. *In* Marc JOYE et Jean-Jacques QUISQUATER, éditeurs : *CHES*, volume 3156 de *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[46] A. BROKALAKIS, A.P. KAKAROUNTAS et C.E. GOUTIS : A high-throughput area efficient fpga implementation of aes-128 encryption. *In Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*, pages 116–121, Nov. 2005.

[47] Marco BUCCI, Luca GIANCANE, Raimondo LUZZI et Alessandro TRIFILETTI : Three-phase dual-rail pre-charge logic. *In* Louis GOUBIN et Mitsuru MATSUI, éditeurs : *CHES*, volume 4249 de *Lecture Notes in Computer Science*, pages 232–241. Springer, 2006.

[48] Javier BUSTOS-JIMÉNEZ, Denis CAROMEL et José M. PIQUER : Load balancing : Toward the infinite network and beyond. *In Job Scheduling Strategies for Parallel Processing*, pages 176–191. Springer-Verlag, 2007.

[49] Vincent CARLIER, Hervé CHABANNE, Emmanuelle DOTTAX et Hervé PELLETIER : Generalizing square attack using side-channels of an aes implementation on an fpga. *In* Tero RISSA, Steven J. E. WILTON et Philip Heng Wai LEONG, éditeurs : *FPL*, pages 433–437. IEEE, 2005.

[50] K. Mani CHANDY et Leslie LAMPORT : Distributed snapshots : Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

[51] Suresh CHARI, Charanjit S. JUTLA, Josyula R. RAO et Pankaj ROHATGI : Towards sound approaches to counteract power-analysis attacks. *In* Michael J. WIENER, éditeur : *CRYPTO*, volume 1666 de *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[52] Suresh CHARI, Josyula R. RAO et Pankaj ROHATGI : Template attacks. *In* Burton S. Kaliski JR., Çetin KAYA KOÇ et Christof PAAR, éditeurs : *CHES*, volume 2523 de *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[53] Ning CHEN et Zhiyuan YAN : Complexity analysis of reed-solomon decoding over gf(2m) without using syndromes. *EURASIP J. Wirel. Commun. Netw.*, 2008:1–11.

[54] Zhimin CHEN et Yujie ZHOU : Dual-rail random switching logic : A countermeasure to reduce side channel leakage. *In* Louis GOUBIN et Mitsuru MATSUI, éditeurs : *CHES*, volume 4249 de *Lecture Notes in Computer Science*, pages 242–254. Springer, 2006.

[55] Benoît CHEVALLIER-MAMES, Mathieu CIET et Marc JOYE : Low-cost solutions for preventing simple side-channel analysis : Side-channel atomicity. *IEEE Trans. Computers*, 53(6):760–768, 2004.

[56] Yoon-Hwa CHOI et Miroslaw MALEK : A fault-tolerant fft processor. *IEEE Trans. Comput.*, 37(5):617–621, 1988.

[57] Yoon-Hwa CHOI et Miroslaw MALEK : A fault-tolerant systolic sorter. *IEEE Trans. Comput.*, 37(5):621–624, 1988.

[58] D. COPPERSMITH : The data encryption standard (des) and its strength against attacks. *IBM J. Res. Dev.*, 38(3):243–250, 1994.

[59] Jean-Sébastien CORON, Emmanuel PROUFF et Matthieu RIVAIN : Side channel cryptanalysis of a higher order masking scheme. *In* Pascal PAILLIER et Ingrid VERBAUWHEDE, éditeurs : *CHES*, volume 4727 de *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.

[60] Nicolas COURTOIS et Josef PIEPRZYK : Cryptanalysis of block ciphers with overdefined systems of equations. *In* Yuliang ZHENG, éditeur : *ASIACRYPT*, volume 2501 de *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[61] Joan DAEMEN : *Cipher and Hash Function Design Strategies based on Linear and Differentialerential Cryptanalysis*. 1995.

[62] Joan DAEMEN, Lars KNUDSEN et Vincent RIJMEN : The block cipher SQUARE. *Lecture Notes in Computer Science*, 1267:149+, 1997.

[63] Joan DAEMEN et Vincent RIJMEN : Probability distributions of correlation and differentials in block ciphers, 2006.

[64] Joan DAEMEN et Vincent RIJMEN : Aes submission document on rijndael, June 1998.

[65] I. DAMAJ, M. ITANI et H. DIAB : Serpent cryptography on static and dynamic reconfigurable hardware. *In Computer Systems and Applications, 2006. IEEE International Conference on.*, pages 680–684, 8, 2006.

[66] Ivan DAMGÅRD : A design principle for hash functions. *In CRYPTO '89 : Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 416–427, London, UK, 1990. Springer-Verlag.

[67] Vincent DANJEAN, Roland GILLARD, Serge GUELTON, Jean-Louis ROCH et Thomas ROCHE : Adaptive loops with kaapi on multicore and grid : Applications in symmetric cryptography. *In* ACM PUBLISHING, éditeur : *Parallel Symbolic Computation'07 (PASCO'07)*, London, Ontario, Canada, July 2007.

[68] M.C. DAVEY et D.J.C. MACKAY : Low density parity check codes over gf(q). *In Information Theory Workshop, 1998*, pages 70–71, Jun 1998.

[69] Donald W. DAVIES et Sean MURPHY : Pairs and triplets of des s-boxes. *J. Cryptology*, 8(1):1–25, 1995.

[70] D. DIVSALAR, H. JIN et R. J. MCELIECE : Coding theorems for "turbo-like" codes. *In Proceedings Thirty-Sixth Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, USA, 1998.

[71] J. DORNSTETTER : On the equivalence between berlekamp's and euclid's algorithms (corresp.). *Information Theory, IEEE Transactions on*, 33(3):428–431, May 1987.

[72] Ilya DUMER et Rafail E. KRICHEVSKIY : Soft-decision majority decoding of reed-muller codes. *IEEE Transactions on Information Theory*, 46(1):258–264, 2000.

[73] E. N. (Mootaz) ELNOZAHY, Lorenzo ALVISI, Yi-Min WANG et David B. JOHNSON : A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[74] E.N. ELNOZAHY, D.B. JOHNSON et W. ZWAENEPOEL : The performance of consistent checkpointing. *In Reliable Distributed Systems, 1992. Proceedings., 11th Symposium on*, pages 39–47, Oct 1992.

[75] Funda ERGÜN, Sampath KANNAN, Ravi KUMAR, Ronitt RUBINFELD et Mahesh VISWANATHAN : Spot-checkers. *J. Comput. Syst. Sci.*, 60(3):717–751, 2000.

[76] Michael J. FISCHER, Nancy A. LYNCH et Mike PATERSON : Impossibility of distributed consensus with one faulty process. *In PODS*, pages 1–7. ACM, 1983.

[77] R. FOURQUET, P. LOIDREAU et C. TAVERNIER : Finding good linear approximations of block ciphers and its application to cryptanalysis of reduced round des. *In Workshop on Coding Theory and Cryptography, Ullensvang, Norvège*, 2009.

[78] Jung fu CHENG et Robert J. MCELIECE : Some high-rate near capacity codecs for the gaussian channel. *In 34th Allerton Conference on Communications, Control and Computing*, 1996.

[79] R. G. GALLAGER : *Low Density Parity Check Codes*. Thèse de doctorat, MIT, Cambridge, Mass., September 1960.

[80] Shuhong GAO : A new algorithm for decoding reed-solomon codes. 2002.

[81] Joachim Von Zur GATHEN et Jurgen GERHARD : *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[82] Gunnar GAUBATZ et Berk SUNAR : Leveraging the multiprocessing capabilities of modern network processors for cryptographic acceleration. *In NCA '05 : Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pages 235–238, Washington, DC, USA, 2005. IEEE Computer Society.

[83] Thierry GAUTIER, Xavier BESSERON et Laurent PIGEON : Kaapi : A thread scheduling runtime system for data flow computations on cluster of multi-processors. *In PASCO '07 : Proceedings of the 2007 international workshop on Parallel symbolic computation*, pages 15–23, New York, NY, USA, 2007. ACM.

[84] Thierry GAUTIER, Jean-Louis ROCH et Frédéric WAGNER : Fine grain distributed implementation of a dataflow language with provable performances. *In ICCS '07 : Proceedings of the 7th international conference on Computational Science, Part II*, pages 593–600, Berlin, Heidelberg, 2007. Springer-Verlag.

[85] B. GERARD et J.-P. TILLICH : On linear cryptanalysis with many linear approximations. Rapport technique, 2007.

[86] Benedikt GIERLICHS, Lejla BATINA, Pim TUYLS et Bart PRENEEL : Mutual information analysis. *In* Elisabeth OSWALD et Pankaj ROHATGI, éditeurs : *CHES*, volume 5154 de *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[87] Oded GOLDREICH et Leonid A. LEVIN : A hard-core predicate for all one-way functions. *In STOC*, pages 25–32. ACM, 1989.

[88] Oded GOLDREICH, Ronitt RUBINFELD et Madhu SUDAN : Learning polynomials with queries : The highly noisy case. *In FOCS*, pages 294–303, 1995.

[89] Louis GOUBIN et Jacques PATARIN : Des and differential power analysis (the "duplication" method). *In* Çetin KAYA KOÇ et Christof PAAR, éditeurs : *CHES*, volume 1717 de *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[90] D. GU, D.J. ROSENKRANTZ et S.S. RAVIL : Design and analysis of test schemes for algorithm-based fault tolerance. *In Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pages 106–113, Jun 1990.

[91] Sylvain GUILLEY, Philippe HOOGVORST et Renaud PACALET : A fast pipelined multimode des architecture operating in ip representation. *Integration*, 40(4):479–489, 2007.

[92] Helena HANDSCHUH et Bart PRENEEL : Blind differential cryptanalysis for enhanced power attacks. *In* Eli BIHAM et Amr M. YOUSSEF, éditeurs : *Selected Areas in Cryptography*, pages 163–173, 2006.

[93] Ctr-Mode Encryption HELGER, Helger LIPMAA, David WAGNER et Phillip ROGAWAY : Comments to nist concerning aes modes of operations :, 2000.

[94] Luca HENZEN, Flavio CARBOGNANI, Jean-Philippe AUMASSON, Sean O'NEIL et Wolfgang FICHTNER : VLSI implementations of the cryptographic hash functions MD6 and ÔrRUPT. *In IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE Press, 2009. To appear.

[95] Howard HEYS : A tutorial on linear and differential cryptanalysis. Rapport technique, 2001.

[96] Howard M. HEYS et Stafford E. TAVARES : Substitution-permutation networks resistant to differential and linear cryptanalysis. *J. Cryptology*, 9(1):1–19, 1996.

[97] Seokhie HONG, Sangjin LEE, Jongin LIM, Jaechul SUNG, Dong Hyeon CHEON et Inho CHO : Provable security against differential and linear cryptanalysis for the spn structure. *In* Bruce SCHNEIER, éditeur : *FSE*, volume 1978 de *Lecture Notes in Computer Science*, pages 273–283. Springer, 2000.

[98] G. HORNAUER, R. WERNSDORF et W. STEPHAN : Markov ciphers and alternating groups. *In EUROCRYPT '93 : Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 453–460, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[99] Xiao-Yu HU, Marc P. C. FOSSORIER et Evangelos ELEFTHERIOU : On the computation of the minimum distance of low-density parity-check codes. *IEEE International Conference on Communications*, 7, 2004.

[100] Kuang-Hua HUANG et J.A. ABRAHAM : Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, 33(6):518–528, 1984.

[101] W.C. HUFFMAN et V. PLESS : *Fundamentals of Error-Correcting Codes*. Cambridge University Press, Cambridge, UK, 2003.

[102] G. Kabatiansky I. DUMER et C. TAVERNIER : List decoding of the first order binary reed muller codes. *Problems of Information Transmission*, 43(3):225–232, 2007.

[103] Yuval ISHAI, Amit SAHAI et David WAGNER : Private circuits : Securing hardware against probing attacks. *In* Dan BONEH, éditeur : *CRYPTO*, volume 2729 de *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[104] Samir JAFAR : *Programmation des systèmes parallèles distribuées : tolérance aux pannes, résilience et adaptabilité*. Thèse de doctorat, Laboratoire Informatique et Distribution de Grenoble, France, 2006.

[105] J.-Y. JOU et J.A. ABRAHAM : Fault-tolerant fft networks. *IEEE Transactions on Computers*, 37(5):548–561, 1988.

[106] Jing-Yang JOU et J.A. ABRAHAM : Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures. *Proceedings of the IEEE*, 74(5):732–741, May 1986.

[107] Burton S. Kaliski JR. et Matthew J. B. ROBSHAW : Linear cryptanalysis using multiple approximations. *In* Yvo DESMEDT, éditeur : *CRYPTO*, volume 839 de *Lecture Notes in Computer Science*, pages 26–39. Springer, 1994.

[108] Pascal JUNOD et Serge VAUDENAY : Fox : A new family of block ciphers. *In* Helena HANDSCHUH et M. Anwar HASAN, éditeurs : *Selected Areas in Cryptography*, volume 3357 de *Lecture Notes in Computer Science*, pages 114–129. Springer, 2004.

[109] G. KABATIANSKY et C. TAVERNIER : List decoding with reed muller codes of order one. *In nine International Workshop On Algebraic and Combinatorial Coding Theory, Proceedings ACCT-9*, pages 230–236, 2004.

[110] G. KABATIANSKY et C. TAVERNIER : List decoding of first order reed-muller codes ii. *In tenth International Workshop On Algebraic and Combinatorial Coding Theory, Proceedings ACCT-10*, 2006.

[111] Masayuki KANDA, Youichi TAKASHIMA, Tsutomu MATSUMOTO, Kazumaro AOKI et Kazuo OHTA : A strategy for constructing fast round functions with practical security against differential and linear cryptanalysis. *In SAC '98 : Proceedings of the Selected Areas in Cryptography*, pages 264–279, London, UK, 1999. Springer-Verlag.

[112] Ju-Sung KANG, Seokhie HONG, Sangjin LEE, Okyeon YI, Choonsik PARK et Jongin LIM : Practical and provable security against differential and linear cryptanalysis for substitution-permutation networks. *ETRI Journal*, 23:158–167, 2001.

[113] Bruce KAPRON, David KEMPE, Valerie KING, Jared SAIA et Vishal SANWALANI : Fast asynchronous byzantine agreement and leader election with full information. *In SODA '08 : Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1038–1047, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[114] Liam KELIHER : *Linear cryptanalysis of substitution-permutation networks*. Thèse de doctorat, Kingston, Ont., Canada, Canada, 2004. Adviser-Meijer, Henk and Adviser-Tavares, Stafford.

[115] Liam KELIHER : Refined analysis of bounds related to linear and differential cryptanalysis for the aes. *In* Hans DOBBERTIN, Vincent RIJMEN et Aleksandra SOWA, éditeurs : *AES Conference*, volume 3373 de *Lecture Notes in Computer Science*, pages 42–57. Springer, 2004.

[116] Liam KELIHER, Henk MEIJER et Stafford E. TAVARES : Improving the upper bound on the maximum average linear hull probability for rijndael. *In* Serge VAUDENAY et Amr M. YOUSSEF, éditeurs : *Selected Areas in Cryptography*, volume 2259 de *Lecture Notes in Computer Science*, pages 112–128. Springer, 2001.

[117] Liam KELIHER, Henk MEIJER et Stafford E. TAVARES : New method for upper bounding the maximum average linear hull probability for spns. *In* Birgit PFITZMANN, éditeur : *EUROCRYPT*, volume 2045 de *Lecture Notes in Computer Science*, pages 420–436. Springer, 2001.

[118] Liam KELIHER et Jiayuan SUI : Exact maximum expected differential and linear probability for 2-round advanced encryption standard (aes). Cryptology ePrint Archive, Report 2005/321, 2005. `http://eprint.iacr.org/`.

[119] John KELSEY, Tadayoshi KOHNO et Bruce SCHNEIER : Amplified boomerang attacks against reduced-round mars and serpent. *In* Bruce SCHNEIER, éditeur : *FSE*, volume 1978 de *Lecture Notes in Computer Science*, pages 75–93. Springer, 2000.

[120] J.G. KEMENY et J.L. SNELL : *Finite Markov Chains*. Springer-Verlag, 1983.

[121] Carl KESSELMAN et Ian FOSTER : *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[122] Javed I. KHAN, W. LIN et D. Y. Y. YUN : Adaptive algorithm-based fault tolerance for parallel computing in linear systems. *In ICPP '94 : Proceedings of the 1994 International Conference on Parallel Processing*, pages 176–183, Washington, DC, USA, 1994. IEEE Computer Society.

[123] Jon-Lark KIM, Keith E. MELLINGER et Leo STORME : Small weight codewords in ldpc codes defined by (dual) classical generalized quadrangles. *Des. Codes Cryptography*, 42(1): 73–92, 2007.

[124] Youngbae KIM : *Fault Tolerant Matrix Operations for Parallel and Distributed Systems*. Thèse de doctorat, The University of Tennessee, Knoxville, 1996.

[125] Valerie KING, Jared SAIA, Vishal SANWALANI et Erik VEE : Towards secure and scalable computation in peer-to-peer networks. *In FOCS '06 : Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 87–98, Washington, DC, USA, 2006. IEEE Computer Society.

[126] Lars R. KNUDSEN : Truncated and higher order differentials. *In Fast Software Encryption*, pages 196–211, 1994.

[127] P. KOCHER, J. JAFFE et B. JUN. : Introduction to differential power analysis and related attacks. Rapport technique, Cryptography Research Inc., 1998.

[128] Paul C. KOCHER : Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. *In* Neal KOBLITZ, éditeur : *CRYPTO*, volume 1109 de *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[129] Paul C. KOCHER, Joshua JAFFE et Benjamin JUN : Differential power analysis. *In* Michael J. WIENER, éditeur : *CRYPTO*, volume 1666 de *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[130] Richard KOO et Sam TOUEG : Checkpointing and rollback-recovery for distributed systems. *IEEE Trans. Software Eng.*, 13(1):23–31, 1987.

[131] H. KRISHNA, K.-Y. LIN et J.-D. SUN : A coding theory approach to error control in redundant residue number systems. i. theory and single error correction. *Circuits and Systems II : Analog and Digital Signal Processing, IEEE Transactions on*, 39(1):8–17, Jan 1992.

[132] Hari KRISHNA et Jenn-Dong SUN : On theory and fast algorithms for error correction in residue number system product codes. *IEEE Trans. Computers*, 42(7):840–853, 1993.

[133] Xuejia LAI et James L. MASSEY : A proposal for a new block encryption standard. *In EUROCRYPT*, pages 389–404, 1990.

[134] Xuejia LAI, James L. MASSEY et Sean MURPHY : Markov ciphers and differential cryptanalysis. *Lecture Notes in Computer Science*, 547:17– ? ?, 1991.

[135] Leslie LAMPORT : Email, 28-05-1987. `http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt`.

[136] Leslie LAMPORT, Robert SHOSTAK et Marshall PEASE : The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[137] Susan K. LANGFORD et Martin E. HELLMAN : Differential-linear cryptanalysis. *In CRYPTO '94 : Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 17–25, London, UK, 1994. Springer-Verlag.

[138] Thanh-Ha LE, Jessy CLÉDIÈRE, Cécile CANOVAS, Bruno ROBISSON, Christine SERVIÈRE et Jean-Louis LACOUME : A proposition for correlation power analysis enhancement. *In* Louis GOUBIN et Mitsuru MATSUI, éditeurs : *CHES*, volume 4249 de *Lecture Notes in Computer Science*, pages 174–186. Springer, 2006.

[139] Hervé LEDIG, Frédéric MULLER et Frédéric VALETTE : Enhancing collision attacks. *In* Marc JOYE et Jean-Jacques QUISQUATER, éditeurs : *CHES*, volume 3156 de *Lecture Notes in Computer Science*, pages 176–190. Springer, 2004.

[140] T.H. LIEW, Lie-Liang YANG et L. HANZO : Systematic redundant residue number system codes : analytical upper bound and iterative decoding performance over awgn and rayleigh channels. *Communications, IEEE Transactions on*, 54(6):1006–1016, June 2006.

[141] R. LIPTON : New directions in testing. *In Distributed Computing and Cryptography, DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, volume 2, pages 191–202. American Mathematical Society, 1991.

[142] Jiqiang LU, Orr DUNKELMAN, Nathan KELLER et Jongsung KIM : New impossible differential attacks on aes. *In* Dipanwita Roy CHOWDHURY, Vincent RIJMEN et Abhijit DAS, éditeurs : *INDOCRYPT*, volume 5365 de *Lecture Notes in Computer Science*, pages 279–293. Springer, 2008.

[143] F. T. LUK et H. PARK : Fault-tolerant matrix triangularizations on systolic arrays. *IEEE Trans. Comput.*, 37(11):1434–1438, 1988.

[144] Jiqiang LV et Yongfei HAN : Enhanced des implementation secure against high-order differential power analysis in smartcards. *In* Colin BOYD et Juan Manuel González NIETO, éditeurs : *ACISP*, volume 3574 de *Lecture Notes in Computer Science*, pages 195–206. Springer, 2005.

[145] D. MANDELBAUM : On a class of arithmetic codes and a decoding algorithm (corresp.). *Information Theory, IEEE Transactions on*, 22(1):85–88, Jan 1976.

[146] James L. MASSEY : Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.

[147] James L. MASSEY : Safer k-64 : A byte-oriented block-ciphering algorithm. *In* Ross J. ANDERSON, éditeur : *FSE*, volume 809 de *Lecture Notes in Computer Science*, pages 1–17. Springer, 1993.

[148] Mitsuru MATSUI : Linear cryptoanalysis method for des cipher. *In EUROCRYPT*, pages 386–397, 1993.

[149] Mitsuru MATSUI : The first experimental cryptanalysis of the data encryption standard. *In* Yvo DESMEDT, éditeur : *CRYPTO*, volume 839 de *Lecture Notes in Computer Science*, pages 1–11. Springer, 1994.

[150] Mitsuru MATSUI et Junko NAKAJIMA : On the power of bitslice implementation on intel core2 processor. *In CHES '07 : Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, pages 121–134, Berlin, Heidelberg, 2007. Springer-Verlag.

[151] Alfred J. MENEZES, Paul C. van OORSCHOT et Scott A. VANSTONE : *Handbook of Applied Cryptography*. CRC Press, 2001.

[152] Ralph C. MERKLE : A digital signature based on a conventional encryption function. *In CRYPTO '87 : A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, pages 369–378, London, UK, 1988. Springer-Verlag.

[153] Thomas S. MESSERGES, Ezzy A. DABBISH et Robert H. SLOAN : Examining smart-card security under the threat of power analysis attacks. *IEEE Trans. Computers*, 51(5):541–552, 2002.

[154] A. MISHRA et P. BANERJEE : An algorithm-based error detection scheme for the multigrid method. *Computers, IEEE Transactions on*, 52(9):1089–1099, Sept. 2003.

[155] D. E. MULLER : Application of boolean algebra to switching circuit design and to error detection. *IEEE Transactions on Computers*, (3):6–12, 1954.

[156] V. S. Sukumaran NAIR : *Analysis and design of algorithm-based fault-tolerant systems*. Thèse de doctorat, Champaign, IL, USA, 1990.

[157] V.S.S. NAIR et J.A. ABRAHAM : General linear codes for fault-tolerant matrix operations on processor arrays. *In Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on*, pages 180–185, Jun 1988.

[158] V.S.S. NAIR et J.A. ABRAHAM : A model for the analysis, design and comparison of fault-tolerant wsi architectures. *In Workshop on Wafer Scale Integration*, Como, Italy, June 1989.

[159] V.S.S. NAIR et J.A. ABRAHAM : Hierarchical design and analysis of fault-tolerant multiprocessor systems using concurrent error detection. *In Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pages 130–137, Jun 1990.

[160] V.S.S. NAIR et J.A. ABRAHAM : Real-number codes for fault-tolerant matrix operations on processor arrays. *IEEE Transactions on Computers*, 39(4):426–435, 1990.

[161] V.S.S. NAIR, Y.V. HOSKOTE et J.A. ABRAHAM : Probabilistic evaluation of online checks in fault-tolerant multiprocessor systems. *Computers, IEEE Transactions on*, 41(5):532–541, May 1992.

[162] Kaisa NYBERG : On the construction of highly nonlinear permutations. *In EUROCRYPT*, pages 92–98, 1992.

[163] Kaisa NYBERG et Lars R. KNUDSEN : Provable security against a differential attack. *J. Cryptology*, 8(1):27–37, 1995.

[164] D. OF : Fips pub, 1980.

[165] Sangwoo PARK, Soo Hak SUNG, Seongtaek CHEE, E-Joong YOON et Jongin LIM : On the security of rijndael-like structures against differential and linear cryptanalysis. *In* Yuliang ZHENG, éditeur : *ASIACRYPT*, volume 2501 de *Lecture Notes in Computer Science*, pages 176–191. Springer, 2002.

[166] Sangwoo PARK, Soo Hak SUNG, Sangjin LEE et Jongin LIM : Improving the upper bound on the maximum differential and the maximum linear hull probability for spn structures and aes. *In* Thomas JOHANSSON, éditeur : *FSE*, volume 2887 de *Lecture Notes in Computer Science*, pages 247–260. Springer, 2003.

[167] H. D. PFISTER, I. SASON et R. URBANKE : Capacity-Achieving Ensembles for the Binary Erasure Channel with Bounded Complexity. *In Fourth ETH–Technion Meeting on Information Theory and Communications*, 2004.

[168] J. PIEPRZYK et L. TOMBAK : Soviet encryption algorithm. preprint 94-10, Department of Computer Science, The University of Wollongong, 1994.

[169] J. S. PLANK, Y. KIM et J. DONGARRA : Algorithm-based diskless checkpointing for fault tolerant matrix operations. *In 25th International Symposium on Fault-Tolerant Computing*, pages 351–360, Pasadena, CA, June 1995.

[170] James S. PLANK, Kai LI et Michael A. PUENING : Diskless checkpointing. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):972–986, 1998.

[171] Jean-Jacques QUISQUATER et David SAMYDE : Electromagnetic analysis (ema) : Measures and counter-measures for smart cards. *In* Isabelle ATTALI et Thomas P. JENSEN, éditeurs : *E-smart*, volume 2140 de *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[172] Sylvia RATNASAMY, Paul FRANCIS, Mark HANDLEY, Richard KARP et Scott SCHENKER : A scalable content-addressable network. *In SIGCOMM '01 : Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[173] A. L. N. REDDY et P. BANERJEE : Algorithm-based fault detection for signal processing applications. *IEEE Trans. Comput.*, 39(10):1304–1308, 1990.

[174] I. S. REED : A class of multiple-error-correcting codes and the decoding scheme. *IEEE Transactions on Information Theory*, (4):38–49, 1954.

[175] I. S. REED et G. SOLOMON : Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[176] James REINDERS : *Intel threading building blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2007.

[177] Mathieu RENAULD, François-Xavier STANDAERT et Nicolas VEYRAT-CHARVILLON : Algebraic side-channel attacks on the aes : Why time also matters in dpa. *In* Christophe CLAVIER et Kris GAJ, éditeurs : *CHES*, volume 5747 de *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009.

[178] Mathieu RENAULD et Francois-Xavier STANDAERT : Algebraic side-channel attacks. Cryptology ePrint Archive, Report 2009/279, 2009. `http://eprint.iacr.org/`.

[179] J. REXFORD et N.K. JHA : Partitioned encoding schemes for algorithm-based fault tolerance in massively parallel systems. *Parallel and Distributed Systems, IEEE Transactions on*, 5(6):649–653, Jun 1994.

[180] T. RICHARDSON et R. URBANKE : The capacity of low-density parity-check codes under message-passing decoding. *Information Theory, IEEE Transactions on*, 47(2):599–618, 2001.

[181] Thomas J. RICHARDSON, Mohammad Amin SHOKROLLAHI et Rüdiger L. URBANKE : Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, 2001.

[182] Thomas RISTENPART, Eran TROMER, Hovav SHACHAM et Stefan SAVAGE : Hey, you, get off of my cloud ! exploring information leakage in third-party compute clouds. *In ACM Conference on Computer and Communications Security*, 2009, to appear.

[183] Matthieu RIVAIN, Emmanuelle DOTTAX et Emmanuel PROUFF : Block ciphers implementations provably secure against second order side channel analysis. *In* Kaisa NYBERG, éditeur : *FSE*, volume 5086 de *Lecture Notes in Computer Science*, pages 127–143. Springer, 2008.

[184] R. L. RIVEST, B. AGRE, D. V. BAILEY, C. CRUTCHFIELD, Y. DODIS, A. Khan K. E. FLEMING, J. KRISHNAMURTHY, Y. LIN, L. REYZIN, E. SHEN, J. SUKHA, D. SUTHERLAND, E. TROMER et Y. L. YIN : The md6 hash function : A proposal to nist for sha-3. Rapport technique, 2008.

[185] J.-L. ROCH, T. GAUTIER et R. REVIRE : Athapascan : Api for asynchronous parallel programming. Rapport technique RT-0276, INRIA Rhône-Alpes, projet APACHE, February 2003.

[186] Jean-Louis ROCH et Sebastien VARRETTE : Probabilistic certification of divide & conquer algorithms on global computing platforms. Application to fault-tolerant exact matrix-vector product. *In* ACM PUBLISHING, éditeur : *Parallel Symbolic Computation'07 (PASCO'07)*, London, Ontario, Canada, July 2007.

[187] Thomas ROCHE, Roland GILLARD et Jean-Louis ROCH : Provable security against impossible differential cryptanalysis application to cs-cipher. *In* Le Thi Hoai AN, Pascal BOUVRY et Pham Dinh TAO, éditeurs : *MCO*, volume 14 de *Communications in Computer and Information Science*, pages 597–606. Springer, 2008.

[188] R. ROLLAND : Décodage des codes de reed-muller d'ordre 1. *Dossiers Acrypta*, 2008.

[189] D. J. ROSENKRANTZ et S. S. RAVI : Improved bounds for algorithm-based fault tolerance. *IEEE Trans. Comput.*, 42(5):630–635, 1993.

[190] Amber ROY-CHOWDHURY et Prithviraj BANERJEE : A fault-tolerant parallel algorithm for iterative solution of the laplace equation. *In ICPP '93 : Proceedings of the 1993 International Conference on Parallel Processing*, pages 133–140, Washington, DC, USA, 1993. IEEE Computer Society.

[191] Amber ROY-CHOWDHURY et Prithviraj BANERJEE : Algorithm-based fault location and recovery for matrix computations. *In FTCS*, pages 38–47, 1994.

[192] Amber ROY-CHOWDHURY et Prithviraj BANERJEE : Algorithm-based fault location and recovery for matrix computations on multiprocessor systems. *IEEE Trans. Computers*, 45(11):1239–1247, 1996.

[193] Amber ROY-CHOWDHURY, Nikolas BELLAS et Prithviraj BANERJEE : Algorithm-based error-detection schemes for iterative solution of partial differential equations. *IEEE Transactions on Computers*, 45(4):394–407, 1996.

[194] Luis F. G. SARMENTA : Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Comp. Syst.*, 18(4):561–572, 2002.

[195] Bruce SCHNEIER : Description of a new variable-length key, 64-bit block cipher (blowfish). *In* Ross J. ANDERSON, éditeur : *FSE*, volume 809 de *Lecture Notes in Computer Science*, pages 191–204. Springer, 1993.

[196] Bruce SCHNEIER : *Applied cryptography : protocols, algorithms, and source code in C*. Wiley, New York, 2nd édition, 1996.

[197] Claus P. SCHNORR et Serge VAUDENAY : Black box cryptanalysis of hash networks based on multipermutations. *Lecture Notes in Computer Science*, 950:47–57, 1995.

[198] Claus P. SCHNORR et Serge VAUDENAY : *La Sécurité des Primitives Cryptographiques*. 1995.

[199] Kai SCHRAMM, Gregor LEANDER, Patrick FELKE et Christof PAAR : A collision-attack on aes : Combining side channel- and differential-attack. *In* Marc JOYE et Jean-Jacques QUISQUATER, éditeurs : *CHES*, volume 3156 de *Lecture Notes in Computer Science*, pages 163–175. Springer, 2004.

[200] Kai SCHRAMM, Thomas J. WOLLINGER et Christof PAAR : A new class of collision attacks and its application to des. *In* Thomas JOHANSSON, éditeur : *FSE*, volume 2887 de *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003.

[201] Claude E. SHANNON : A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[202] Claude E. SHANNON : Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28, 1949.

[203] A. SHIOZAKI : Decoding of redundant residue polynomial codes using euclid's algorithm. *Information Theory, IEEE Transactions on*, 34(5):1351–1354, Sep 1988.

[204] Damien STEHLÉ : Rapport de stage : Les algorithmes hgcd sur les entiers et les polynômes, 2001. `http://perso.ens-lyon.fr/damien.stehle/downloads/stage1.ps.gz`.

[205] Jacques STERN et Serge VAUDENAY : Cs-cipher. *In FSE '98 : Proceedings of the 5th International Workshop on Fast Software Encryption*, pages 189–205, London, UK, 1998. Springer-Verlag.

[206] W.J. STEWART : *An Introduction to Numerical Solution of Markov Chains*. Princeton University Press, New Jersey, 1994.

[207] Ion STOICA, Robert MORRIS, David KARGER, M. Frans KAASHOEK et Hari BALAKRISHNAN : Chord : A scalable peer-to-peer lookup service for internet applications. *In Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[208] Jeremy J. STONE : Multiple-burst error correction with the chinese remainder theorem. *Journal of the Society for Industrial and Applied Mathematics*, 11(1):74–81, Mar. 1963.

[209] Makoto SUGITA, Kazukuni KOBARA, Kazuhiro UEHARA, Shuji KUBOTA et Hideki IMAI : Relationships among differential, truncated differential, impossible differential cryptanalyses against word-oriented block ciphers like RIJNDAEL, e2. *In AES Candidate Conference*, pages 242–254, 2000.

[210] J.-D. SUN et H. KRISHNA : A coding theory approach to error control in redundant residue number systems. ii. multiple error detection and correction. *Circuits and Systems II : Analog and Digital Signal Processing, IEEE Transactions on*, 39(1):18–34, Jan 1992.

[211] Zhangxi TAN, Chuang LIN, Hao YIN et Bo LI : Optimization and benchmark of cryptographic algorithms on network processors. *IEEE Micro*, 24(5):55–69, 2004.

[212] Andrew S. TANENBAUM et Maarten van STEEN : *Distributed Systems : Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[213] D.L. TAO et C.R.P. HARTMANN : A novel concurrent error detection scheme for fft networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):198–221, 1993.

[214] Kris TIRI et Ingrid VERBAUWHEDE : A logic level design methodology for a secure dpa resistant asic or fpga implementation. *In DATE*, pages 246–251. IEEE Computer Society, 2004.

[215] Michael TUNSTALL et Olivier BENOÎT : Efficient use of random delays in embedded software. *In* Damien SAUVERON, Constantinos MARKANTONAKIS, Angelos BILAS et Jean-Jacques QUISQUATER, éditeurs : *WISTP*, volume 4462 de *Lecture Notes in Computer Science*, pages 27–38. Springer, 2007.

[216] Sébastien VARRETTE : *Sécurité des Architectures de Calcul Distribué : Authentification et Certification de Résultats*. Thèse de doctorat, Université du Luxembourg et Laboratoire Informatique et Distribution de Grenoble, France, 2007.

[217] Serge VAUDENAY : On the need for multipermutations : Cryptanalysis of md4 and SAFER. *In Fast Software Encryption*, pages 286–297, 1994.

[218] Serge VAUDENAY : An experiment on des statistical cryptanalysis. *In ACM Conference on Computer and Communications Security*, pages 139–147, 1996.

[219] Serge VAUDENAY : On the security of cs-cipher. *In FSE '99 : Proceedings of the 6th International Workshop on Fast Software Encryption*, pages 260–274, London, UK, 1999. Springer-Verlag.

[220] N. VEYRAT-CHARVILLON et F-X. STANDAERT : Mutual information analysis : How, when and why ? Can be found here : http ://www.dice.ucl.ac.be/˜fstandae/PUBLIS/67.pdf.

[221] B. VINNAKOTA et N.K. JHA : Diagnosability and diagnosis of algorithm-based fault tolerant systems. *In Circuits and Systems, 1989., Proceedings of the 32nd Midwest Symposium on*, pages 28–31 vol.1, Aug 1989.

[222] David WAGNER : The boomerang attack. *In* Lars R. KNUDSEN, éditeur : *FSE*, volume 1636 de *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[223] Shuang WANG, S. CHENG et Qiang WU : A parallel decoding algorithm of ldpc codes using cuda. *In Signals, Systems and Computers, 2008 42nd Asilomar Conference on*, pages 171–175, Oct. 2008.

[224] Sying-Jyan WANG et N. K. JHA : Algorithm-based fault tolerance for fft networks. *IEEE Trans. Comput.*, 43(7):849–854, 1994.

[225] Stephen A. WARD, Luis F. G. SARMENTA et Luis F. G. SARMENTA : Volunteer computing. Rapport technique, 2001.

[226] M. WIESMANN, F. PEDONE et A. SCHIPER : A Systematic Classification of Replicated Database Protocols based on Atomic Broadcast. *In Proceedings of the 3rd Europeean Research Seminar on Advances in Distributed Systems (ERSADS'99)*, 1999.

[227] Lie-Liang YANG et L. HANZO : Redundant residue number system based error correction codes. *In Vehicular Technology Conference, 2001. VTC 2001 Fall. IEEE VTS 54th*, volume 3, pages 1472–1476 vol.3, 2001.

[228] Yao-Ming YEH et Tse-yun FENG : Algorithm-based fault tolerance for matrix inversion with maximum pivoting. *J. Parallel Distrib. Comput.*, 14(4):373–389, 1992.

[229] J. F. ZIEGLER, H. W. CURTIS, H. P. MUHLFELD, C. J. MONTROSE et B. CHIN : Ibm experiments in soft fails in computer electronics (1978–1994). *IBM J. Res. Dev.*, 40(1):3–18, 1996.

IV

Annexes

Preuves des théorèmes 7 et 8 A

Dans cette annexe, nous démontrons les théorèmes 7 et 8 utilisés respectivement pour évaluer une borne supérieure et inférieure des probabilités différentielles d'un chiffre symétrique de structure PSN. Toutes les notations et lemmes utilisés ici sont définis dans le chapitre 5.

Théorème 12 (théorème 7 : Borne Supérieure pour Structure PSN) *Soit E un algorithme de chiffrement de Markov et $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov associée. Pour tout $i \in \{2, \cdots, R\}$,*

$$\begin{aligned}\Pr(X_i = x_i | X_0 = x_0) \quad \leq \quad & [DP_{max} \times (2^l - 1)]^{wt(\gamma_{\mathcal{L}(x_0)})} \\ & \times \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}, \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \\ & \times \Pr(\chi(X_i) = \gamma_{x_i} | \chi(X_1) = \gamma_{\mathcal{L}(x_0)})\end{aligned}$$

Preuve Commençons par introduire X_1 dans la formule via la loi des probabilités totales :

$$\Pr(X_i = x_i | X_0 = x_0) = \sum_{x_1} \Pr(X_i = x_i | X_1 = x_1, X_0 = x_0) \Pr(X_1 = x_1 | X_0 = x_0)$$

Le lemme 5 nous donne :

$$\Pr(X_i = x_i | X_0 = x_0) \leq (\mathrm{DP}_{max})^{wt(\gamma_{x_1})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}(x_0)} = \gamma_{x_1}}} \Pr(X_i = x_i | X_1 = x_1, X_0 = x_0)$$

Si $\gamma_{\mathcal{L}(x_0)} = \gamma_{x_1}$ pour tout les éléments de la somme, nous avons :

$$\Pr(X_i = x_i | X_0 = x_0) \leq (\mathrm{DP}_{max})^{wt(\gamma_{\mathcal{L}(x_0)})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}(x_0)} = \gamma_{x_1}}} \Pr(X_i = x_i | X_1 = x_1, X_0 = x_0)$$

Et comme $(X_0, X_1, \cdots, X_R)$ est une chaîne de Markov,

$$\Pr(X_i = x_i | X_0 = x_0) \leq (\mathrm{DP}_{max})^{wt(\gamma_{\mathcal{L}(x_0)})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}(x_0)} = \gamma_{x_1}}} \Pr(X_i = x_i | X_1 = x_1)$$

Qui peut se ré-écrire sous la forme :

$$\Pr(X_i = x_i | X_0 = x_0) \leq (\mathrm{DP}_{max})^{wt(\gamma_{\mathcal{L}(x_0)})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}(x_0)} = \gamma_{x_1}}} \frac{\Pr(X_i = x_i, X_1 = x_1)}{\Pr(X_1 = x_1)}$$

Or $\Pr(X_1 = x_1)$ est une constante pour tout x_1 et vaut $1/2^n$:

$$\Pr(X_i = x_i | X_0 = x_0) \leq 2^n (\text{DP}_{max})^{wt(\gamma_{\mathcal{L}(x_0)})} \sum_{\substack{x_1 t.q. \\ \gamma_{\mathcal{L}(x_0)} = \gamma_{x_1}}} \Pr(X_i = x_i, X_1 = x_1)$$

En sommant, nous obtenons :

$$\Pr(X_i = x_i | X_0 = x_0) \leq 2^n (\text{DP}_{max})^{wt(\gamma_{\mathcal{L}(x_0)})} \Pr(X_i = x_i, \chi(X_1) = \gamma_{\mathcal{L}(x_0)})$$

Ainsi, en développant le dernier terme et en remarquant que
$\Pr(\chi(X_1) = \gamma_{\mathcal{L}(x_0)}) = \frac{(2^l - 1)^{wt(\gamma_{\mathcal{L}(x_0)})}}{2^n}$,

$$\Pr(X_i = x_i | X_0 = x_0) \leq [\text{DP}_{max} \times (2^l - 1)]^{wt(\gamma_{\mathcal{L}(x_0)})} \Pr(X_i = x_i | \chi(X_1) = \gamma_{\mathcal{L}(x_0)})$$

Enfin, en introduisant le terme $\chi(X_i)$ via la loi des probabilités totale,

$$\begin{aligned}\Pr(X_i = x_i | X_0 = x_0) \leq [\text{DP}_{max} \times (2^l - 1)]^{wt(\gamma_{\mathcal{L}(x_0)})} \times \\ \sum_{\gamma} \Pr(X_i = x_i | \chi(X_i) = \gamma, \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \Pr(\chi(X_i) = \gamma | \chi(X_1) = \gamma_{\mathcal{L}(x_0)})\end{aligned}$$

Il suffit de remarquer que $\Pr(X_i = x_i | \chi(X_i) = \gamma, \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \neq 0$ uniquement si $\gamma = \gamma_{x_i}$ pour obtenir le résultat cherché. □

Théorème 13 (théorème 8 : Borne Inférieure pour Structure PSN) *Soit E un algorithme de chiffrement de Markov et $(X_0, X_1, \cdots, X_R)$ la chaîne de Markov associée. Pour tout $i \in \{2, \cdots, R\}$,*

$$\begin{aligned}\Pr(X_i = x_i | X_0 = x_0) \quad \geq \quad & [(DP_{min})^2 \times (2^l - 1)]^{wt(\gamma_{\mathcal{L}(x_0)})} \\ & \times \Pr(X_i = x_i | \chi(X_i) = \gamma_{x_i}, \chi(X_1) = \gamma_{\mathcal{L}(x_0)}) \\ & \times \Pr(\chi(X_i) = \gamma_{x_i} | \chi(X_1) = \gamma_{\mathcal{L}(x_0)})\end{aligned}$$

Preuve Commençons par introduire X_1 dans la formule via la loi des probabilités totales :

$$\Pr(X_i = x_i | X_0 = x_0) = \sum_{x_1} \Pr(X_i = x_i | X_1 = x_1, X_0 = x_0) \Pr(X_1 = x_1 | X_0 = x_0)$$

Le lemme 6 nous donne :

$$\Pr(X_i = x_i | X_0 = x_0) \geq (\text{DP}_{min})^{wt(\gamma_{x_1})} \sum_{\substack{x_1 t.q. \\ \Pr(X_1 = x_1 | X_0 = x_0) \neq 0}} \Pr(X_i = x_i | X_1 = x_1)$$

D'après le lemme 4, si $\Pr(X_1 = x_1 | X_0 = x_0) \neq 0$ alors $\gamma_{\mathcal{L}(x_0)} = \gamma_{x_1}$, nous avons donc :

$$\Pr(X_i = x_i | X_0 = x_0) \geq (\text{DP}_{min})^{wt(\gamma_{\mathcal{L}(x_0)})} \sum_{\substack{x_1 t.q. \\ \Pr(X_1 = x_1 | X_0 = x_0) \neq 0}} \Pr(X_i = x_i | X_1 = x_1)$$

Notons $\left\{ \begin{array}{lcl} Poss_{x_0} & = & \{x, \Pr(X_1 = x | X_0 = x_0) \neq 0\} \\ Supp_{x_0} & = & \{x, \gamma_x = \gamma_{\mathcal{L}(x_0)}\} \end{array} \right.$

Nous allons évaluer $\sum_{x_1 \in Poss_{x_0}} \Pr(X_i = x_i | X_1 = x_1)$ en remarquant tout d'abord que $Poss_{x_0} \subset Supp_{x_0}$ (d'après le lemme 4).

De plus le cardinal de $Supp_{x_0}$ est $(2^l - 1)^{wt(\gamma_{\mathcal{L}(x_0)})}$ et le cardinal de $Poss_{x_0}$ vaut, au moins, $(\text{DP}_{min}(2^l - 1))^{wt(\gamma_{\mathcal{L}(x_0)})}$, nous avons donc :

$$Card(\{Poss_{x_0}\}) \geq (\text{DP}_{min})^{wt(\gamma_{\mathcal{L}(x_0)})} Card(\{Supp_{x_0}\})$$

Enfin, considérant $(X_0, X_1, \cdots, X_R)$ comme une chaîne de Markov :
$\forall (x, y), \Pr(X_i = y | X_1 = x)$ est indépendant du fait que $x \in Poss_{x_0}$ ou $x \in Supp_{x_0}$, nous obtenons donc :

$$\Pr(X_i = x_i | X_0 = x_0) \geq (\text{DP}^2_{min})^{wt(\gamma_{\mathcal{L}(x_0)})} \sum_{x_1 \in Supp_{x_0}} \Pr(X_i = x_i | X_1 = x_1)$$

La fin de la preuve est identique à la preuve du théorème 12 □

Attaque MLAC sur traces de consommation du DPA-context B

Cette annexe décrit les détails de l'attaque MLAC dont les résultats sont présentés dans la section 6.2.4 du chapitre 6. Nous allons d'abord voir la mise en oeuvre de l'attaque, puis montrer, à l'aide de résultats de simulations, que l'utilisation d'approximations est particulièrement adaptée aux attaques par canaux cachés.

B.1 Mise en oeuvre de l'attaque

Les traces de consommation utilisées pour tester l'attaque sont des traces réelles, publiques, distribuées lors d'un concours international lancé par le groupe de recherche VLSI du département COMELEC de l'école d'ingénieur française Telecom ParisTech : dpa-contest[1]. Plusieurs jeux de traces de consommation sont proposés, nous avons utilisé le jeu de traces `secmatv1_2006_04_0809` : 81089 traces correspondant à la mesure de la consommation de courant durant autant d'exécutions d'une implémentation du chiffre DES. Les détails de cette implémentation peuvent être trouvés dans [91], la figure B.1 en donne une vue d'ensemble, suffisante pour nos besoins.

Sur la figure B.1, seule la partie gauche nous intéresse, la partie droite illustre la génération de clés de rondes. Les blocs FP et IP correspondent respectivement à la permutation finale et initiale du DES, le registre LR est un registre de 64 bits contenant la sortie de la dernière ronde exécutée (i.e. l'entrée de la prochaine ronde). Enfin, *Round logic* est un bloc de logique combinatoire définissant une ronde de chiffrement : il prend en entrée la valeur stockée dans le registre LR et la clé de ronde produite par le bloc *Key schedule*.

Nous noterons $H(x)$ la fonction de poids de Hamming, $IP(x)$ et $FP(x)$ seront respectivement la permutation initiale et finale de l'algorithme de chiffrement et $DES_n(x,k)$ sera la fonction de chiffrement DES sur ses n premiers tours (x étant un vecteur de 64 bits et k une clé secrète sur 56 bits). Enfin nous noterons $DES_0(x,k) = IP(x)$ et l'exécution complète de chiffre : $DES(x,k) = FP(DES_{16}(x,k))$. D'après la figure B.1, après chaque ronde, la valeur de sortie est stockée dans le registre LR, ainsi, dans le modèle HD (distance de Hamming), les valeurs intermédiaires $\{V_i\}_{1\leq i\leq 16}$ qui nous intéressent sont telles que :

$$V_i(x,k) = DES_{i-1} \oplus DES_i(x,k)$$

[1] `http://www.dpacontest.org/`

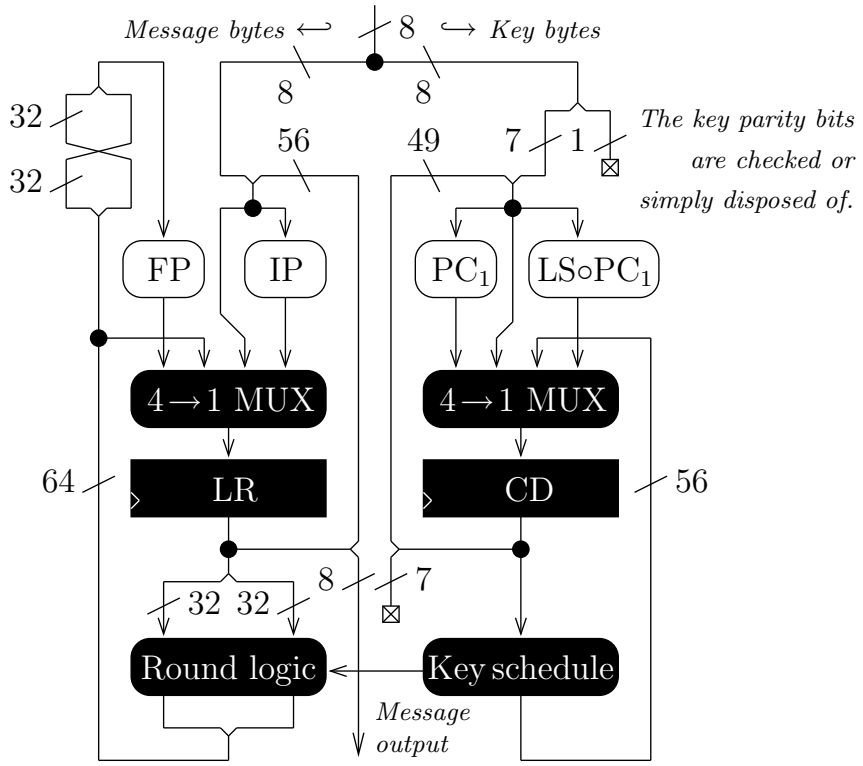

FIG. B.1 – Schéma d'implémentation du chiffre DES (tiré de [91])

Celles que nous avons utilisé pour l'attaque sont celles pour lesquelles nous avons trouvé de bonnes et nombreuses approximations linéaires :

- V_1 qui correspond à la fin de la première ronde.
- V_2 qui correspond à la fin de la deuxième ronde.
- V_{15} qui correspond à la fin de la 15ème ronde.
- V_{16} qui correspond à la fin de la 16ème ronde (avant FP).

Chaque trace de consommation correspond à l'exécution de l'algorithme de chiffrement en entier pour une valeur d'entrée particulière (la clé est la même pour toutes les traces) : maintenant que les valeurs intermédiaires sont choisies, il faut trouver dans les traces de consommation les index qui correspondent à chacune d'elles. Pour les mesures du dpa-contest, toutes les traces sont synchronisées les unes avec les autres, il suffit donc de trouver, pour l'ensemble des traces, le moment (l'index) de fin de chaque ronde. L'attaque MLAC suppose que cette donnée est connue de l'attaquant, à l'aide par exemple d'un premier traitement des traces de consommation. Dans notre cas, nous avons choisi les index qui donnent les meilleurs résultats :

- V_1 qui correspond à la fin de la première ronde : index 5749 des traces.
- V_2 qui correspond à la fin de la deuxième ronde : index 6374 des traces.

- V_{15} qui correspond à la fin de la 15ème ronde : index 14491 des traces.
- V_{16} qui correspond à la fin de la 16ème ronde : index 15749 des traces.

La figure B.2 (resp. figure B.3) représente les valeurs de chaque trace de consommation à l'index 6374 (resp. 14491) en fonction de la valeur exacte de $H(V_2)$ (resp. $H(V_{15})$). Ces figures montrent bien la dépendance entre les mesures de consommation et les valeurs intermédiaires, justifiant ainsi le modèle en distance de Hamming pour cette implémentation.

Remarque 42 *Le nuage de points de la figure B.3 est notablement plus fin que celui de la figure B.2, signifiant une meilleure corrélation entre mesure et valeur intermédiaire pour le point de fuite V_{15}. Cela rejoint une remarque faite dans le chapitre 6 : les mesures semblent moins bruitées en fin de chiffrement (V_{15}) qu'en début de chiffrement (V_2).*

Des approximations linéaires ont été trouvées pour chacune des valeurs intermédiaires :
- $< H(V_1(x,k)), \Gamma_H >$
- $< H(V_2(x,k)), \Gamma_H >$
- $< H(V_{15}(x,k)), \Gamma_H >$
- $< H(V_{16}(x,k)), \Gamma_H >$

avec le masque de sortie Γ_H prenant les valeurs 32 et 16. La valeur de Γ_H sélectionne soit le bit de poids fort de la distance de Hamming ($\Gamma_H = 32$), soit le deuxième bit de poids fort ($\Gamma_H = 16$). Le tableau B.1 donne un exemple d'approximations linéaires utilisées pour monter l'attaque : il s'agit de 11 approximations de la valeur $< H(V_2(x,k)), \Gamma_H >$, elles font intervenir un total de 6 bits de clé ($K[j]$ est le $j$ème bit de la clé secrète et $P[j]$ est le $j$ème bit du message d'entrée).

Algorithme d'attaque MLAC L'attaque se déroule exactement comme décrit dans la section 6.2.3.1 du chapitre 6, la seule différence est que nous avions supposé que la valeur de $H(V_i)$ était connue exactement par l'observation des traces de consommation. Ici, le choix de masques de sortie simples (16 ou 32) nous permet d'évaluer avec une précision assez grande la valeur de $< H(V_i(x,k)), \Gamma_H >$. En effet, considérons N traces de consommation en un point de fuite donné (V_i), nous supposerons que si la mesure de consommation d'une trace est supérieure à la consommation moyenne sur l'ensemble des N traces, alors $< H(V_i(x,k)), 32 >= 1$ et $< H(V_i(x,k)), 16 >= 0$. Cette supposition correspond à dire deux choses :
- Si la distance de Hamming est supérieure à 32 alors il y a de grandes chances pour que la mesure de consommation soit au dessus de la moyenne.
- Il y a une chance négligeable pour que la distance de Hamming soit strictement supérieure à 58 et inférieure à 16.

Nous allons voir dans la section suivante que ces suppositions sont tout à fait légitimes.

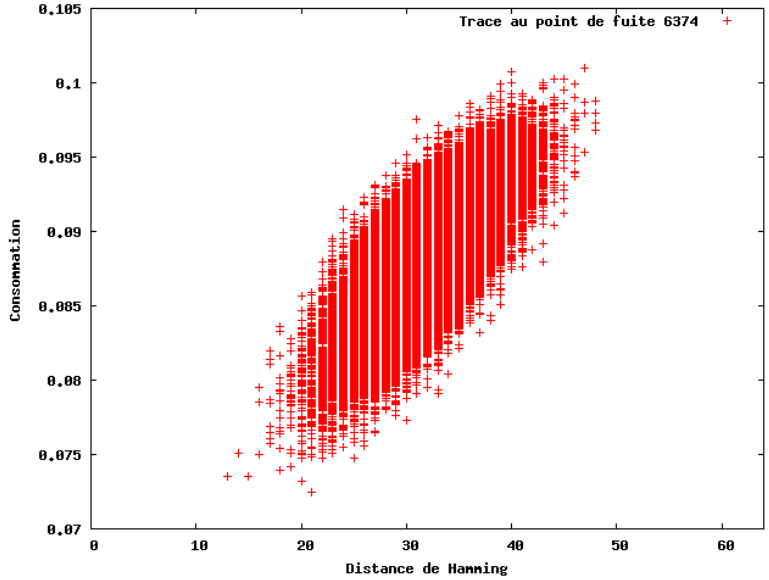

FIG. B.2 – Répartition des mesures de courant au point 6374 en fonction de $H(V_2)$

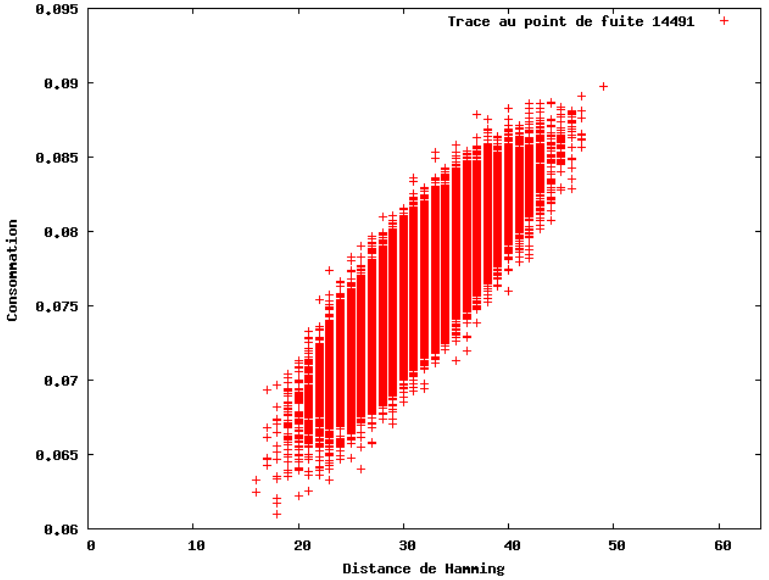

FIG. B.3 – Répartition des mesures de courant au point 14491 en fonction de $H(V_{15})$

Γ_H	Biais	Équations
0x10	0.0219	0 ⊕ P[5]⊕ P[26]⊕ P[27]⊕ P[31]⊕ P[45]⊕ P[53]⊕ P[61]⊕ K[6]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]
0x20	0.0215	1 ⊕ P[5]⊕ P[26]⊕ P[27]⊕ P[31]⊕ P[45]⊕ P[53]⊕ P[61]⊕ K[6]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]
0x20	0.0134	0 ⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[45]⊕ P[53]⊕ K[6]⊕ K[7]⊕ K[29]⊕ K[61]
0x20	0.0156	1 ⊕ P[5]⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[45]⊕ K[6]⊕ K[29]⊕ K[38]⊕ K[61]
0x10	0.0142	0 ⊕ P[5]⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[45]⊕ K[6]⊕ K[29]⊕ K[38]⊕ K[61]
0x20	0.0189	1 ⊕ P[5]⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[53]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[61]
0x10	0.0189	0 ⊕ P[5]⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[53]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[61]
0x10	0.0126	1 ⊕ P[26]⊕ P[27]⊕ P[37]⊕ P[45]⊕ P[53]⊕ P[61]⊕ K[6]⊕ K[7]⊕ K[52]⊕ K[61]
0x20	0.0163	0 ⊕ P[5]⊕ P[8]⊕ P[9]⊕ P[37]⊕ P[45]⊕ P[53]⊕ P[61]⊕ K[6]⊕ K[7]⊕ K[38]⊕ K[52]⊕ K[61]
0x10	0.0167	1 ⊕ P[5]⊕ P[8]⊕ P[9]⊕ P[37]⊕ P[45]⊕ P[53]⊕ P[61]⊕ K[6]⊕ K[7]⊕ K[38]⊕ K[52]⊕ K[61]
0x10	0.0215	1 ⊕ P[5]⊕ P[14]⊕ P[15]⊕ P[31]⊕ P[37]⊕ P[45]⊕ P[61]⊕ K[6]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x10	0.0146	0 ⊕ P[5]⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[45]⊕ P[61]⊕ K[6]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x10	0.0148	1 ⊕ P[5]⊕ P[8]⊕ P[9]⊕ P[31]⊕ P[37]⊕ P[45]⊕ P[61]⊕ K[6]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x20	0.0223	0 ⊕ P[5]⊕ P[14]⊕ P[15]⊕ P[31]⊕ P[37]⊕ P[45]⊕ P[61]⊕ K[6]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x20	0.0182	0 ⊕ P[5]⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[53]⊕ P[61]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x10	0.0152	0 ⊕ P[5]⊕ P[26]⊕ P[27]⊕ P[31]⊕ P[37]⊕ P[53]⊕ P[61]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x10	0.0187	1 ⊕ P[5]⊕ P[28]⊕ P[29]⊕ P[31]⊕ P[37]⊕ P[53]⊕ P[61]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x20	0.0157	1 ⊕ P[5]⊕ P[26]⊕ P[27]⊕ P[31]⊕ P[37]⊕ P[53]⊕ P[61]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x20	0.0191	0 ⊕ P[5]⊕ P[26]⊕ P[27]⊕ P[31]⊕ P[37]⊕ P[45]⊕ P[53]⊕ P[61]⊕ K[6]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]
0x10	0.0183	1 ⊕ P[5]⊕ P[26]⊕ P[27]⊕ P[31]⊕ P[37]⊕ P[45]⊕ P[53]⊕ P[61]⊕ K[6]⊕ K[7]⊕ K[29]⊕ K[38]⊕ K[52]⊕ K[61]

TAB. B.1 – Approximations linéaires de $< H(V_2(x,k)), \Gamma_H >$

B.2 Simulations

Pour simuler des attaques nous nous contentons de supposer que la mesure de consommation nous donne la valeur exacte du poids de Hamming de la valeur intermédiaire choisie. Sous cette hypothèse, nous avons lancé l'attaque sur les chiffres DES et AES. L'attaque sur le DES est la même que celle présentée plus haut, les mêmes approximations linéaires sont utilisées. Pour la fonction de chiffrement de l'AES, seulement le dernier tour a été approximé (particulièrement simple puisque ce dernier tour ne contient pas la fonction de diffusion `Mix Column`). Les résultats sont donnés sur la figure B.4.

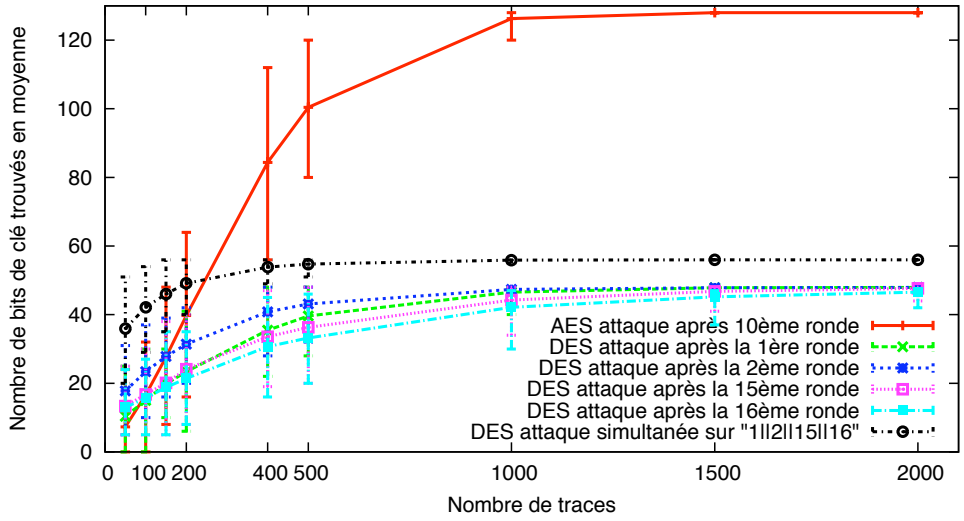

FIG. B.4 – Complexité de l'attaque MLAC sur AES et DES en simulation

Attaque du DES La figure B.5 reprend les résultats, présentés dans le chapitre 6, de l'attaque sur les traces de consommation du dpa-contest, la figure B.6 correspond aux résultat de la même attaque, cette fois-ci en simulation.

La seule grande différence entre ces deux figures est la courbe correspondant à l'attaque sur le deuxième tour. Cela confirme le fait que les mesures pour cette valeur intermédiaire (V_2) sont particulièrement bruitées sur les traces du dpa-contest.

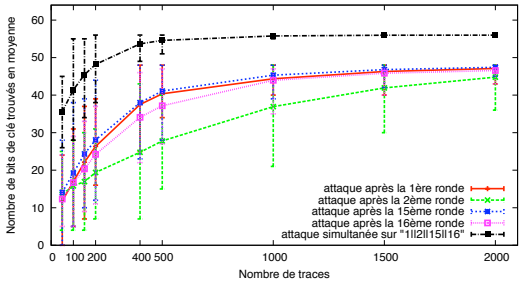

FIG. B.5 – Complexité de l'attaque sur les traces du dpa-contest

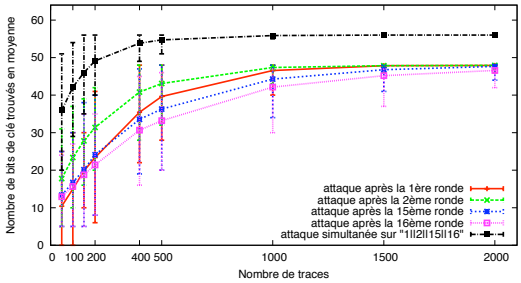

FIG. B.6 – Complexité de l'attaque en simulation

On peut voir sur la figure B.7 la comparaison entre l'attaque sur les traces réelles et l'attaque en simulation, la différence est très faible. Rappelons que les barres verticales représentent les valeurs maximales et minimales atteintes, les intervalles de confiance sont toujours inférieurs à 1%.
Ainsi, l'erreur commise lors du choix de la valeur de $< H(V_i(x,k)), \Gamma_H >$ à partir des mesures de consommation (cf. section précédente) a une influence négligeable devant la réussite de l'attaque (pour les traces du dpa-contest). En réalité cette erreur est masquée par le biais des approximations linéaires, ce n'est qu'à partir d'une certaine limite qu'elle est visible dans les résultats (c'est le cas pour V_2).

Cette remarque rend légitime l'approche MLAC par rapport à une attaque algébrique : l'utilisation d'approximations au lieu de systèmes d'équations exacts rend la méthode plus souple pour un type d'attaque faisant appel à des mesures physiques, et donc imparfaites.

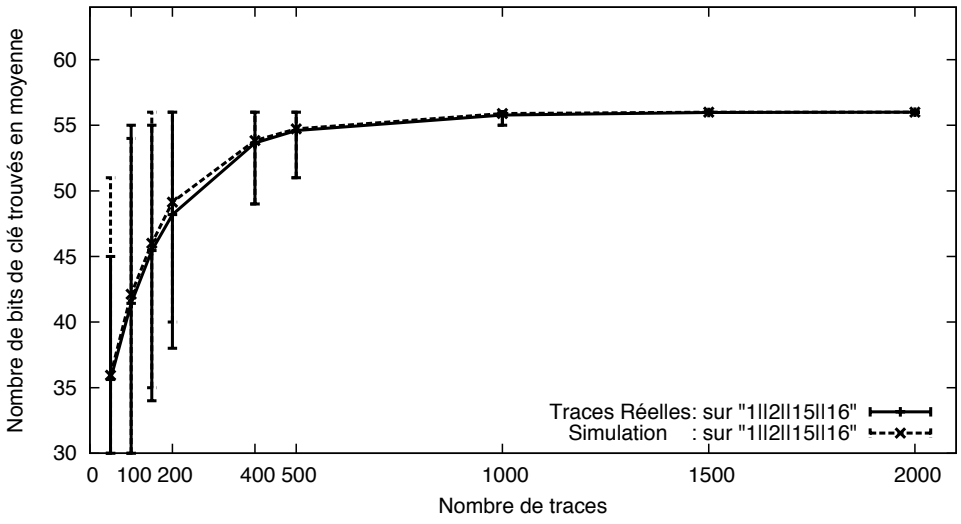

FIG. B.7 – Complexité de l'attaque MLAC sur le DES, comparaison entre traces réelles et simulations

Coût des opérations arithmétiques C

Dans le chapitre 9, l'ensemble de définition E des matrices n'est pas explicité. Or la complexité des opérations arithmétiques sur E, nécessaires dans le produit de matrices ou dans les algorithmes de codage (addition, multiplication, division euclidienne et pgcd), sont dépendantes de la taille et des caractéristiques de l'ensemble. Dans le tableau C.1, sont répertoriés les coûts de ces opérations pour quelques exemples de E, rappelons que nous nous intéressons au coût sur plate-formes parallèles (travail ω_q et profondeur δ_q de l'opération) et pour de grandes tailles d'éléments. Dans le tableau, pour alléger les formules, on pose $n = \log q$.

Les algorithmes pour la multiplication sur les entiers, polynômes et corps finis utilisent la Transformée de Fourrier Rapide, intéressante surtout sur de grands éléments. D'autres algorithmes existent, il ne s'agit pas ici d'être exhaustif. Les complexités énoncées dans le tableau sont tirées de [32] pour les corps de Galois et les polynômes. Pour les corps de Galois, exhiber une complexité des opérations est d'autant plus difficile que les types de corps finis sont très différents. C'est bien souvent au cas par cas que les algorithmes les plus efficaces en pratiques sont développés. Ici, nous considérons le cas particulier $GF(p^m)$ où p est assez grand pour qu'il existe une racine $m$ème de l'unité dans $\mathbb{Z}/p\mathbb{Z}$ et ainsi utiliser la FFT dans les opérations polynomiales. Les complexités sont alors équivalentes à celles des opérations arithmétiques sur les polynômes, plus précisément, pour le coût séquentiel de la multiplication, nous trouvons ([32]) :

$$O(m \log m \log\log m \log p \log\log p \log\log\log p) = \widetilde{O}(m \log p) = \widetilde{O}(\log q)$$

Le cas des Rationnels n'est pas donné, les complexités sont les mêmes que dans le cas des entiers à part si le numérateur et de dénominateur sont réduits en facteurs premiers à chaque opération, dans ce cas toutes les opérations sont surchargées par un calcul de pgcd et lui empruntent sa complexité.

	Opération	Coût séquentiel	Coût parallèle	
			Travail ω_q	Profondeur δ_q
Entiers (opérandes $\leq q$)	addition	$O(n)$	$O(n)$	$O(\log n)$
	multiplication	$O(n \log n \log\log n)$	$O(n \log n \log\log n)$	$O(\log n)$
	division euclid., reste modulaire	$O(n \log n \log\log n)$	$O(n \log n \log\log n)$	$O(\log^2 n)$
	pgcd, inv. modulaire	$O(n \log^2 n \log\log n)$	$O(n \log^2 n \log\log n)$	$O(n)$
Polynômes à coeff. dans un corps $\mathbb{K}$ (degrés $\leq n$)	addition	$O(n)$	$O(n)$	$O(\log n)$
	multiplication	$O(n \log n)$	$O(n \log n)$	$O(\log n)$
	division euclid., reste modulaire	$O(n \log n)$	$O(n \log n)$	$O(\log^2 n)$
	pgcd, inv. modulaire	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(n \log n)$
	évaluation/ interpolation en n points	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(\log^3 n)$
Corps fini $GF(q)$ $q = p^m$	addition	$O(\log q)$	$O(\log q)$	$O(\log\log q)$
	multiplication	$\widetilde{O}(\log q)$	$\widetilde{O}(\log q)$	$\widetilde{O}(\log\log q)$
	division	$\widetilde{O}(\log q)$	$\widetilde{O}(\log q)$	$\widetilde{O}(\log^2 \log q)$

TAB. C.1 – Tableau récapitulatif de complexité des opérations arithmétiques $n = \log q$

Résumé

Dimensionnement et intégration d'un chiffre symétrique dans le contexte d'un système d'information distribué de grande taille

L'évolution des architectures de systèmes d'information n'a de cesse depuis l'avènement des télécommunications. Pour s'adapter aux nouvelles dimensions en nombre d'utilisateurs et taille des données, les systèmes d'information modernes sont répartis et hétérogènes. Ce constat ouvre de nouveaux enjeux de sûreté de fonctionnement pour les systèmes d'information, il s'agit de garantir disponibilité, fiabilité, innocuité, confidentialité, intégrité et maintenabilité tout au long du cycle de vie du système.
C'est dans ce contexte que cette thèse a été conduite, les fruits des travaux de recherche gravitent autour de plusieurs notions de la sûreté de fonctionnement :

- La *confidentialité* et l'*intégrité* avec le design, l'implémentation et l'étude de robustesse des chiffres symétriques par bloc. Les chiffres DES, AES et CS-Cipher ont été étudiés dans un spectre large rassemblant les implémentations logicielles et matérielles et leurs performances, la malléabilité des structures de diffusion, la cryptographie conventionnelle et les attaques par canaux auxiliaires.
- La *fiabilité* avec le développement d'algorithmes auto-tolérants aux fautes et la certification de résultats. Cette étude est dédiée aux systèmes distribués de calcul tels que les clusters, les grilles ou encore les réseaux pair-à-pairs utilisés pour des calculs d'algèbre linéaires.

Mots-clés : sûreté de fonctionnement, chiffre symétrique, cryptanalyse, attaques par canaux auxiliaires, algorithmes auto-tolérants aux fautes, systèmes distribués.

Abstract

Capacity planning and integration of a symmetric cipher within the framework of a large distributed information system.

Information systems' architecture is in constant evolution since the advent of telecommunications. In order to keep up with the growth of users and data sizes, modern information systems are distributed and heterogeneous. This evolution affects every attribute of the system's dependability : availability, reliability, safety, confidentiality, integrity and maintainability.
Our work is in the line with information system dependability, the main contributions focus on the following aspects :

- *Confidentiality* and *integrity*, with the design, the implementation and the robustness study of symmetric ciphers. Block ciphers DES, AES and CS-Cipher have been examined within a broad spectrum : software and hardware implementations and their performances, diffusion pattern malleability, conventional cryptanalysis and side-channel attacks.
- *Reliability* with the constructions of fault tolerant algorithms and result certification. This study is dedicated to the computation of involved problems in scientific fields using large distributed systems for their massive computational power such as grids or peer-to-peer computing platforms.

Keywords : dependability, symmetric ciphers, cryptanalysis, side-channel attacks, algorithm-based fault tolerance, distributed systems.