Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Liste des acronymes

AES Advanced Encryption Standard. 6

ASIC Application-Specific Integrated Circuit. 20

CAS Chinese Academy of Science. 85

CIFRE Convention Industrielle de Formation par la Recherche. 5

CMOS Complementary Metal-Oxide Semiconductor. 23

CPA Correlation Power Analysis. 27, 31

DES Data Encryption Standard. 13

DPA Differential Power Analysis. 27, 30

FIPS Federal Information Processing Standards. 15

FPGA Field-Programmable Gate Array. 20

GE Gates Equivalent. 22

NIST National Institute of Standards and Technology. 14, 112

NSA National Security Agency. 15

OSCCA Office of State Commercial Cryptography Administration. 85, 111, I, II

RTL Register-Transfert Level. 21

SCA Side-Channel Analysis. 24

SM4 Shāngyè Mìmǎ 4. 13

SNR Signal-to-Noise Ratio. 33

SPA Simple Power Analysis. 27

SPN Schéma Substitution-Permutation. 12, 14, 15

Chapitre 1

Introduction générale

Sommaire

On considère souvent les mathématiques comme une science abstraite et détachée de la réalité. Les mathématiques appliquées existent bel et bien. L'analyse et la modélisation d'un phénomène physique viennent le plus souvent des mathématiques et l'envie de pouvoir l'utiliser, le prévoir ou l'améliorer est alors toujours très forte. L'application du savoir mathématique est alors présente dans de nombreux domaines. On cite par exemple la construction d'un avion qui vole, empêcher que celui-ci ne s'écrase s'il prend la foudre, concevoir une voiture consommant peu d'essence, envoyer un satellite sur orbite avec le moins de carburant possible, estimer l'impact du changement climatique, prévoir la météo, améliorer les connaissances sur l'évolution des maladies de l'homme et du vivant en général,... Nous nous intéressons dans ce manuscrit à deux branches précises des mathématiques : l'arithmétique modulaire et la théorie des nombres, qui donnent naissance à la *cryptologie*. La cryptologie est la science du secret. Le terme vient des mots grecs anciens *kruptos* qu'on traduit par "secret" et *logos* par "discours".

De l'Antiquité à notre époque moderne, ce chapitre présente dans les Sections 1.1,1.2 et 1.3 l'objectif de la cryptologie, ses origines et son évolution à travers le temps. La Section 1.4 décrit le contexte dans lequel cette thèse s'inscrit, le sujet sur lequel nous nous sommes penchés et les différents objectifs mis en place afin de contribuer à la recherche académique et industrielle en cryptologie. La structure de ce manuscrit en cinq chapitres ainsi que les résultats obtenus sont détaillés en Section 1.5. Enfin, la Section 1.6 présente les motivations auxquelles ce manuscrit aspire à atteindre.

1.1 Jargon de la cryptologie

La cryptologie permet à deux personnes, appelées traditionnellement Alice et Bob, de communiquer à travers un canal, de telle sorte qu'un attaquant, Ève, ne puisse pas comprendre le message échangé. Alice va ainsi transformer un *message clair* en un *message chiffré* par un *algorithme de chiffrement*, en utilisant une *clé secrète*. Le message chiffré est ensuite envoyé le long du canal et Bob, qui connaît la clé secrète, va récupérer le message original par un *algorithme de déchiffrement*. La Figure 1 illustre la situation.

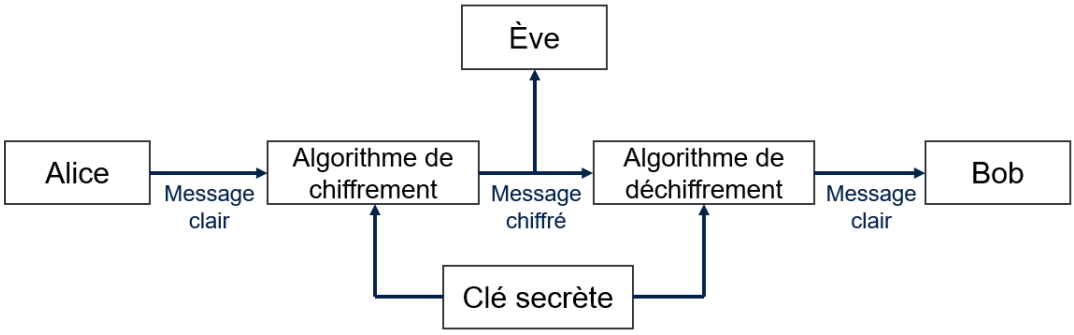


FIGURE 1 – Principe de communication en cryptologie entre Alice et Bob

L'étude de la cryptologie s'articule autour de quatre principes : la *confidentialité* assure que le message secret soit bien lu par des personnes choisies au préalable, l'*authenticité* permet que l'expéditeur soit bien l'auteur du message envoyé, la *non-répudiation* affirme que le message secret ait bien été envoyé et/ou reçu et l'*intégrité* vérifie que le message n'a pas été modifié sans autorisation ou par erreur. Afin de satisfaire ces principes, la science du secret se divise en deux branches : la *cryptographie*, soit l'écriture du message secret et la *cryptanalyse*, soit l'analyse du message secret. La Figure 2 illustre par exemple l'analyse d'un message chiffré par un célèbre personnage de bande dessinée.

FIGURE 2 – La cryptographie (à gauche) et la cryptanalyse (à droite) en image

1.2 Histoire de la cryptologie

Les origines de la cryptographie remontent à plus de 4000 ans, avec la découverte d'une tombe égyptienne à Saqqarah où reposait un haut responsable de la V^e dynastie des pharaons. Les écrits retrouvés sur les parois du sarcophage contenaient des hiéroglyphes modifiés [1]. Ces nouveaux

1. Des égyptologues ont montré la fonction de ces inscriptions. Elles avaient pour but d'assister le défunt dans chacune des étapes à franchir dans le dangereux périple qui l'attendait dans le royaume des morts. Ces textes funéraires sont en lien avec le *Livre des morts*, baptisé ainsi par le savant allemand Lepsius.

symboles représentent le premier élément essentiel de la cryptographie : une modification volontaire de l'écriture. Les découvertes archéologiques égyptiennes montrent ainsi que la cryptographie, et plus précisément la transmission sûre d'informations, sont aussi anciennes que l'invention de l'écriture elle-même.

Le premier exemple de cryptographie a été conçu durant l'Antiquité, entre le X^e et le VII^e siècle av. J.-C., avec une technique de chiffrement par transposition utilisée par les Spartiates et les Athéniens pendant la guerre du Péloponnèse. Ce chiffrement consiste à réorganiser les caractères d'un message. Au moyen de deux bâtons identiques (appelés *scytales*, voir Figure 3) partagés par l'expéditeur et le destinataire, un message écrit sur une bandelette enroulée sur ces bâtons n'aurait plus aucune signification une fois la bandelette déroulée. Par la suite les systèmes de chiffrements deviennent de plus en plus compliqués jusqu'à l'invention des machines qui encodent. Ces dernières sont destinées à transmettre des messages d'un point à un autre sur de grandes distances, à l'aide de codes secrets pour une transmission rapide et fiable. On peut par exemple citer la création par Edison du télégraphe utilisant le code Morse au XIX^e siècle ou encore les machines Enigma (voir Figure 4) de Sherbius dont les nazis se servaient pendant la Seconde Guerre mondiale.

FIGURE 3 – Exemple d'un scytale

De la cryptographie naît la cryptanalyse, le besoin de "casser" les messages chiffrés, soit l'analyse des messages secrets. Ce sont les Arabes qui, les premiers, ont compris les mécanismes de la cryptographie et ont commencé à développer la cryptanalyse. En particulier, en 1412, Al-Kalka-Shandi décrit comment procéder au calcul de fréquence des lettres d'un message chiffré afin de le décoder. Un autre exemple frappant en cryptanalyse est la compréhension du langage des hiéroglyphes par Champollion en 1822 à partir de la Pierre de Rosette (voir Figure 5), trouvée en Égypte en 1799. La pierre contenait trois texte d'un décret de Ptolémée V Epiphane, inscrits en hiéroglyphes, en démotique et en grec ancien. Après avoir compris que les parties en démotique et en grec ancien correspondaient au même texte, Champollion a trouvé la clé pour traduire les hiéroglyphes [50]. Enfin, nous citons le cas le plus connu du XX^e siècle, le décodage des machines Enigma par les services de renseignements Polonais et Britanniques. Parmi les héros, le mathématicien Turing est l'homme que l'on considère comme le père de l'informatique moderne. En effet, il résulte de l'analyse du code Enigma le premier ordinateur de l'histoire, soit la découverte la plus spectaculaire de l'histoire de la cryptanalyse militaire.

FIGURE 4 – Machine Enigma

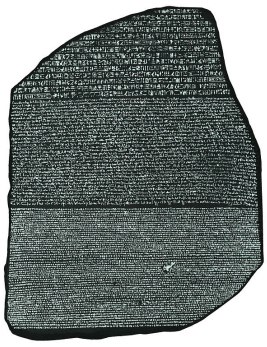


FIGURE 5 – Pierre de Rosette

On trouve peu de détails sur les innovations en cryptographie et cryptanalyse durant le début de la guerre froide, probablement parce que ces informations sont encore "top secret". C'est l'arrivée des ordinateurs et l'utilisation d'Internet dans les années 1970 qui bousculent l'époque moderne avec de nouvelles applications cryptographiques. La cryptographie se divise alors en deux grandes familles d'algorithmes de chiffrement que nous présentons ci-dessous.

1.3 Les types de chiffrement en cryptographie

De l'Antiquité au milieu du XX^{e} siècle, les méthodes de chiffrement nécessitent au préalable la mise en commun entre Alice et Bob d'une même clé secrète, indispensable au chiffrement du message clair et au déchiffrement du message chiffré. Ce type de cryptographie est appelée *cryptographie à clé secrète* ou encore *cryptographie symétrique*. Selon le principe de Kerckhoffs [44] (1883), la sécurité de la cryptographie symétrique repose sur la protection de la clé secrète. Autrement dit, l'algorithme de chiffrement peut être connu mais pas la clé. Dans les années 1970, la quantité d'algorithmes symétriques a explosé proportionnellement à l'expansion du traitement informatique. Afin d'assurer leur transmission, les informations sont codées puis chiffrées en langage binaire (composé de deux caractères, 0 ou 1, appelés des *bits*), traité par les ordinateurs. Si les algorithmes symétriques présentent l'avantage d'utiliser des clés de petites tailles (moins de 256 bits) et donc d'être rapides à exécuter, leur nature les exposent au risque qu'un récepteur indésirable (Ève) parvienne à obtenir la clé secrète. Pour chiffrer un message, il faut donc au préalable avoir échangé une clé avec le destinataire du message, et cela par une voie absolument sûre, sinon chiffrer devient inutile. Cette faiblesse est connue comme le *problème de la distribution de clé* et occasionne de réels soucis logistiques : on peut par exemple citer le système utilisé pour le Téléphone rouge entre le Kremlin et la Maison-Blanche en 1963.

C'est en 1976 que Diffie et Hellman [28] ont proposé une solution au problème de la distribution de clé, en introduisant le concept de *cryptographie à clé publique* ou *cryptographie asymétrique*. Utilisant deux clés, l'une pour chiffrer et l'autre pour déchiffrer, la cryptographie à clé publique a réellement vu le jour à partir de 1978 avec l'invention du `RSA` par Rivest, Shamir et Adlmeman [76]. Basés sur des problèmes mathématiques difficiles à résoudre, comme la factorisation d'un nombre élevé en un produit de nombres premiers pour le `RSA`, les algorithmes à clé publique transforment le problème de la distribution de clé en un processus de communications en plusieurs étapes. Alice chiffre un message avec une clé publique, c'est à dire connue de tous, et l'envoie à Bob. Dès lors, le message est incompréhensible, même en connaissant cette clé publique. Bob pourra utiliser une clé secrète, connue de lui seul, afin de déchiffrer le message secret. En règle générale, les algorithmes à clé publique sont plus lents à exécuter que les algorithmes symétriques car ils manipulent des clés de taille très importante. Par exemple, l'*Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI)* recommande en 2010 d'utiliser des clés de taille supérieure ou égale à 2048 bits pour l'algorithme `RSA`.

De nos jours, la cryptographie (symétrique et asymétrique) peut être assurée au moyen de *cartes à puce*, des petits objets électroniques de quelques millimètres capables de stocker des éléments confidentiels dans une mémoire interne et d'effectuer des calculs cryptographiques grâce à une source d'énergie. On parle également de *systèmes ou composants embarqués*. En toute ignorance, tout un chacun utilise quotidiennement des applications de la cryptographie. Par exemple, il suffit de payer un achat avec une carte de crédit : la sécurité de cette transaction bancaire repose sur l'exécution d'algorithmes cryptographiques sur la puce de la carte de crédit. Nous présentons dans la suite dans quel contexte s'inscrivent les recherches effectuées dans cette thèse.

1.4 Contexte, sujet et objectifs de cette thèse, entre académie et industrie

Depuis les années 2000, le marché des cartes à puce a explosé dans de nombreux domaines d'applications, tels que les clés de voiture, les badges d'accès, les cartes de transport, les banques, les passeports électroniques ou encore très récemment l'identification biométrique. Afin d'assurer la confiance dans leurs produits et de pouvoir les vendre, les développeurs de cartes à puce demandent un certificat auprès d'un organisme de confiance. En France, c'est l'ANSSI qui s'assure que le système embarqué ne présente pas de faille de conception face aux nombreuses at-

taques de cryptanalyse. En effet, la carte à puce étant un objet concret, il est facile pour un individu malveillant de se la procurer et de l'exploiter afin de retrouver les données confidentielles qu'elle protège. Dans ce manuscrit nous nous intéressons à une famille d'attaques en cryptanalyse moderne [2], introduite par Kocher [46] en 1996 et appelée *attaques par observations*. Ces dernières analysent des grandeurs physiques observables (e.g. consommation de courant électrique) lors de l'exécution d'un algorithme cryptographique sur la carte à puce afin d'en extraire de l'information sur les données secrètes. Elles exploitent donc les vulnérabilités du composant et la façon dont l'algorithme est implémenté dessus.

L'évolution des techniques en cryptanalyse moderne a eu un impact très important sur le monde industriel. Le besoin de protéger les cartes à puce contre les attaques est devenu nécessaire. Garantir la sécurité d'une carte à puce tout en étant efficace en termes de temps d'exécution et de stockage est un sujet de recherche et de développement très prisé par les mondes académiques et industriels. De nombreuses *contre-mesures* ont alors été proposées dans l'état de l'art (comme par exemple les travaux [18, 3, 43, 37, 11, 35]) afin de parer aux attaques par observations. Cependant, en 2005, Mangard *et. al.* [56] exploitent des phénomènes, appelés des *glitches*, présents au sein d'un composant électronique. Les auteurs démontrent que ces glitches peuvent supprimer les protections des développeurs. C'est en 2006 que les techniques de défense contre les glitches évoluent, avec la contre-mesure de Nikova *et. al.* [65], appelée *implémentation* [3] *à seuil*. C'est dans ce contexte que s'inscrit cette thèse dont le sujet est le suivant :

Étude de la résistance des algorithmes cryptographiques symétriques face à la cryptanalyse moderne.

L'objectif principal de cette thèse est de contribuer à l'état de l'art cryptographique en proposant de nouvelles contre-mesures pour la cryptographie symétrique contre les attaques par observations. Nous avons choisi de baser nos recherches sur le travail de Nikova *et. al.* sur les implémentations à seuil.

J'ai eu l'opportunité de mener ces recherches dans le cadre d'un dispositif par Convention Industrielle de Formation par la Recherche (CIFRE) dans l'entreprise STMicroelectronics en partenariat avec le LIP6 (Laboratoire d'Informatique de Paris 6) à l'EDITE (École Doctorale Informatique, Télécommunications et Électronique) de Paris. STMicroelectronics est une multinationale franco-italienne qui fabrique et commercialise notamment des puces électroniques sécurisées. Durant ces trois années, j'ai étudié au sein de l'équipe de cryptographie et de sécurité (laboratoire CSLab) de STMicroelectronics, ayant pour but de valider la sécurité des cartes à puce produites par l'entreprise. L'équipe est composée de cinq ingénieurs et deux étudiants en thèse. À la différence d'un contrat de thèse dans un laboratoire public, j'ai ainsi pu me familiariser avec le monde industriel technologique. Cette double casquette industrie-académie m'a par exemple permis de comprendre les contraintes électroniques liées aux composants embarqués avant de proposer de nouvelles contre-mesures, tout aussi sécurisées qu'efficaces en terme de performances.

Dans une dynamique de s'intégrer au marché industriel mondial, l'objectif secondaire donné par STMicroelectronics dans cette thèse est de proposer une implémentation sécurisée contre les attaques par observations d'un algorithme symétrique très utilisé en Chine mais peu connu par la communauté cryptographique occidentale, appelé `SM4` [29]. Afin de participer à certains marchés chinois, il est en effet obligatoire d'intégrer une telle implémentation dans les produits de l'entreprise. Le travail réalisé pourrait être intégré dans les produits de l'entreprise.

La section suivante décrit le contenu des différents chapitres de ce manuscrit.

2. On parle de cryptanalyse moderne à partir des années 1970, en lien avec l'expansion du traitement informatique.
3. Une implémentation est l'action d'écrire, réaliser un programme informatique dans le but de le concrétiser.

1.5 Organisation du manuscrit de thèse

Comme illustré en Figure 6, ce manuscrit décrit en cinq chapitres le travail de recherche qui nous a amené à proposer de nouvelles idées en cryptographie symétrique, dans les mondes académique et industriel. Un sixième chapitre est consacré à la conclusion générale de ce manuscrit.

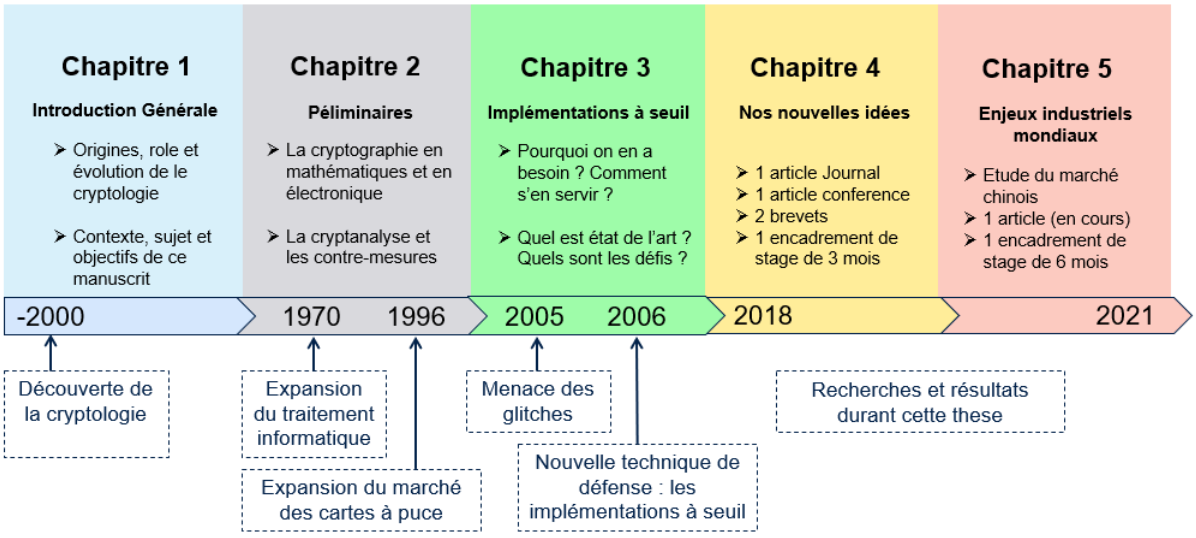


FIGURE 6 – Organisation du manuscrit de thèse en 5 chapitres

Nous détaillons ci-dessous chacun des chapitres de ce manuscrit ainsi que les travaux effectués.

- *Chapitre 1*. Ce chapitre introduit de manière générale la cryptologie. Nous y présentons l'histoire et l'évolution de la cryptologie de l'Antiquité à notre époque moderne. Nous décrivons également le contexte scientifique dans lequel le sujet de recherche de cette thèse s'inscrit. La fin du chapitre est dédiée à l'organisation du manuscrit et à la présentation brève des résultats obtenus durant ces trois années de thèse en entreprise, aussi bien pour le monde académique (publications) qu'industriel (brevets).

- *Chapitre 2*. Le deuxième chapitre introduit tous les préliminaires nécessaires à la compréhension de ce manuscrit. Nous présentons tout d'abord les notions mathématiques utilisées dans ce manuscrit ainsi que le principe des algorithmes symétriques à chiffrement par blocs. Nous décrivons notamment l'algorithme classique de la cryptographie symétrique couramment utilisé dans les applications de la carte à puce, l'Advanced Encryption Standard (AES). Puis, nous étudions le fonctionnement d'un circuit électronique et ses étapes de conception. Cette introduction nous permet de comprendre les mécanismes d'attaque en cryptanalyse et plus particulièrement ceux d'une attaque par observations sur un composant électronique. Nous illustrons notre explication par une application d'une telle attaque sur l'AES. Enfin, nous présentons les différentes contre-mesures proposées dans l'état de l'art afin de se protéger des attaques par observations.

- *Chapitre 3*. Le troisième chapitre présente les différents modèles de sécurité impliqués dans ce manuscrit ainsi que le principe des implémentations à seuil de Nikova *et. al.* [65]. Cette contre-mesure a été proposée dans l'état de l'art en 2006 dans le but de se protéger des attaques par observations malgré la présence de phénomènes inattendus au sein des composants électronique, appelés des *glitches* [56]. Afin de ne pas confondre avec la notion traditionnellement donnée en électronique, nous définissons et illustrons les glitches dans le

contexte des attaques par observations. Enfin, nous mettons en valeur les différentes propositions d'implémentations à seuil matérielles de l'AES de l'état de l'art ainsi que leur performances en terme de vitesse d'exécution et de surface. Le but est de comprendre les différents défis à relever pour apporter de nouvelles idées et contribuer ainsi à la recherche cryptographique.

- *Chapitre 4*. Nous présentons dans le quatrième chapitre les résultats de la première partie de nos travaux. Notre but est de proposer une nouvelle manière de sécuriser un algorithme cryptographique symétrique contre les attaques par observations en présence de glitches, avec de meilleures performances que celles des propositions existantes de l'état de l'art. Nos recherches ont débouché sur deux articles étroitement liés. Nous les décrivons ci-dessous :

  ▶ *Monomial Evaluation of Polynomial Function Protected by Threshold Implementations - Illustration on the AES* - [49]. Simon Landry, Yanis Linge et Emmanuel Prouff. Conférence WISTP 2019.

  Ce papier propose une nouvelle façon d'évaluer le calcul d'un monôme. Notre construction se base sur les implémentations à seuil introduites par Nikova *et. al.* [65] et permet une sécurité à l'ordre 1 en présence de glitches. De manière à pouvoir comparer les performances de notre méthode avec l'état de l'art [21, 22, 40, 88], nous appliquons notre technique sur la partie non-linéaire de l'algorithme symétrique AES. Plus encore, nous proposons une construction sécurisée pour toutes les étapes de calcul de cet algorithme. Enfin, nous étendons notre proposition en une évaluation sécurisée générique du calcul de fonctions polynomiales (sous certaines conditions), ce qui n'était jusque là pas possible dans l'état de l'art. Ce papier a été publié et présenté à la conférence *WISTP* à Paris en décembre 2019.

  ▶ *Monomial Evaluation of Polynomial Function Protected by Threshold Implementations - Illustration on the AES - Extended Version* [48]. Simon Landry, Yanis Linge et Emmanuel Prouff. Journal Cryptography and Communications 2020.

  Cet article est une extension de l'article [49] auquel nous apportons un support mathématique et matériel. Nous fournissons en effet des preuves mathématiques permettant de démontrer la sécurité contre les attaques par observation à l'ordre 1 en présence de glitches de notre évaluation monomiale, mais aussi de notre construction sécurisée de tous les calculs impliqués dans l'AES. Afin d'être complet, nous ajoutons la description de l'implémentation à seuil de la génération des clés de tour de l'algorithme. De plus, nous implémentons sur le FPGA *PYNQ-Z2* notre proposition de la partie non-linéaire de l'AES (la plus difficile à protéger). En simulant l'implémentation grâce au logiciel *Vivado Design Suite* nous montrons ainsi le caractère fonctionnel de notre technique et nous mesurons ses performances. Nous mettons ainsi en valeurs les avantages de notre techniques par rapport à celles de l'état de l'art. Ce papier a été publié dans le journal *Cryptography and Communication* en juin 2021.

Afin d'être en raccord avec les articles de l'état de l'art les plus récents, nous nous sommes également concentrés sur la recherche d'une construction résistante à des attaques par observations aux ordres supérieurs en présence de glitches. Dans le chapitre 4, nous proposons ainsi une nouvelle approche, différente de celle de Nikova *et. al.* , quant à la manière de construire une implémentation à seuil d'une fonction polynomiale. Notre technique est générique pour n'importe quel ordre de sécurité souhaité contre les attaques par observations en présence de glitches, et pourrait s'appliquer sur n'importe quelle fonction polynomiale

(*i.e.* autre que la partie non-linéaire de l'AES).

Par ailleurs, les recherches effectuées ont donné naissance à d'autres idées permettant de se protéger contre la cryptanalyse moderne. Nous avons choisi, avec mon tuteur d'entreprise Yanis Linge, de présenter deux brevets au sein de STMicroelectronics. Ces innovations ont été acceptées par l'entreprise. Le deuxième brevet (le plus récent) étant encore confidentiel, nous ne donnons que le principe du premier ci-dessous :

- *Brevet 1 - Fault detection.* Proposée en avril 2019 à STMicroelectronics, cette innovation est basée sur le principe des implémentations à seuil. L'information secrète est traitée de manière à détecter la présence d'un attaquant lors d'une *attaque par faute*. Contrairement aux attaques par observations, les attaques par fautes sont des attaques invasives. Elles consistent à perturber le comportement d'un composant pendant l'exécution d'un algorithme sur celui-ci. Nous ne les considérons pas dans ce manuscrit de thèse. Notre technique permet également de réduire considérablement le taux de réussite d'un attaquant même en présence de glitches.

- *Chapitre 5.* Nous présentons dans le cinquième chapitre les résultats de la deuxième partie de nos recherches. Afin de contribuer au marché industriel mondial des composants électroniques, nous nous sommes concentrés sur l'étude de l'algorithme symétrique SM4, utilisé par le gouvernement chinois notamment pour la sécurité des réseaux sans fils. En collaboration avec Gaëtan Sclafani, un étudiant en stage de six mois que j'ai eu le plaisir d'encadrer, nous avons pu écrire un article (en cours de soumission dans une conférence) décrit ci-dessous :

  - *Benchmark of SM4 Countermeasures Against* 1^{st}*-Order Side-Channel Analysis Attacks* - citation à venir. Simon Landry, Yanis Linge, Emmanuel Prouff et Gaëtan Sclafani.

    Le SM4 [29] étant très peu étudié par la communauté cryptographique occidentale, nous centralisons dans cet article les informations sur l'état de l'art du SM4. Nous décrivons notamment les mécanismes mathématiques de la partie non linéaire de l'algorithme SM4, ce qui n'était jusque là pas clair dans la littérature cryptographique. Nous présentons les différentes contre-mesures de l'état de l'art pour le SM4 contre les attaques par observations. Le modèle de sécurité visé ne prend pas en compte les glitches. Afin de pouvoir comparer les techniques étudiées, nous les implémentons en logiciel sur composant STM32. Les critères visés sont la vitesse d'exécution en nombre de cycles et la mémoire utilisée.

  De plus, nous contribuons à l'état de l'art par la première proposition d'une implémentation à seuil complète du SM4, prouvée mathématiquement sécurisée à l'ordre 1 contre les attaques par observations en présence de glitches.

- *Chapitre 6.* Ce chapitre conclue ce manuscrit. Nous résumons les différents travaux effectués durant cette thèse et les contributions apportées à l'état de l'art. Nous discutons également de futures pistes de recherches intéressantes à explorer pour de nouveaux projets. Nous terminons par une discussion autour des avantages issus des collaborations entre chercheurs du monde public et ingénieurs du domaine industriel.

1.6 Motivations

À travers ce manuscrit je souhaiterais mettre en valeur le lien étroit qu'il existe entre la recherche académique et la recherche industrielle. Bien que visant des objectifs différents, découvrir

de nouveaux savoirs et les publier pour l'une, mettre au point des produits innovants et fiables répondant à un marché dans un contexte de forte compétition internationale pour l'autre, elles sont loin d'être indépendantes. En tant qu'acteur de la sphère publique et privée, j'ai pu durant ces trois années de thèse CIFRE, participer à la valorisation de la collaboration entre chercheurs académiques et industriels. De la recherche des techniques de chiffrement au cours de l'Histoire, à l'émergence des cartes à puce et leurs applications industrielles, la cryptographie représente à ce titre une illustration parfaite.

Je ne vous expliquerai pas comment installer un anti-virus sur votre ordinateur, ni comment éviter qu'un pirate vous vole votre compte en banque. Comprendre les mécanismes d'attaque sur une carte à puce telle que votre carte bleue et comment s'en protéger sont des objectifs que ce manuscrit aspire à atteindre. La cryptographie, l'électronique et la cryptanalyse sont les pas d'une "danse de la sécurité" en constante évolution, et la musique qui accompagne et régit cette danse n'est autre que la mathématique. Je vous invite à découvrir cet univers dans le chapitre suivant.

Chapitre 2

Préliminaires : cryptographie, électronique et cryptanalyse

Sommaire

Dans ce chapitre, nous introduisons quelques bases de la cryptologie et de l'électronique. Nous détaillons dans la Section 2.1 les mécanismes de chiffrement et de déchiffrement en cryptographie symétrique. Nous décrivons notamment les opérations cryptographiques de l'algorithme symétrique AES [27]. Dans notre quotidien, les algorithmes cryptographiques sont embarqués dans des composants électroniques (e.g. des cartes à puce) [55], ce qui les rend vulnérables à certaines attaques en cryptanalyse. Afin de mieux comprendre le fonctionnement de ces attaques, nous donnons des prérequis importants en électronique dans la Section 2.2. Nous expliquons par exemple d'où viennent les informations physiques qu'un attaquant exploite dans le but de retrouver la clé secrète utilisée par l'algorithme cryptographique sur le composant électronique. Puis, la Section 2.3 est dédiée à la présentation des attaques par observations, nous en donnons un exemple sur l'AES. Enfin, nous présentons dans la Section 2.4 des moyens classiques de défense étudiés par la communauté cryptographique contre ces attaques.

2.1 Introduction à la cryptographie symétrique

2.1.1 Les mathématiques en cryptographie

Nous débutons ce chapitre par l'introduction de notions mathématiques importantes utilisées en cryptographie.

En algèbre linéaire, un *corps fini*, aussi connu sous le nom de *corps de Galois* (*Galois Field* en anglais), est un corps commutatif dont le nombre d'éléments est fini. Il est décrit par son cardinal p^n où p est premier et représente sa caractéristique, et n est un entier non nul. On le note alors $\mathrm{GF}(p^n)$. De plus, l'ensemble des éléments non-nuls de $\mathrm{GF}(p^n)$ est noté $\mathrm{GF}(p^n)^\star$.

Un corps fini est utilisé pour décrire des opérations sur des données informatiques, représentées par leur forme binaire. Les données cryptographiques sont représentées par des chaînes de n bits (e.g. les *octets* (8 bits)), que nous pouvons associer à des vecteurs dans l'espace vectoriel $\mathrm{GF}(2)^n$. Le symbole $\oplus$ est alors utilisé pour dénoter la somme dans $\mathrm{GF}(2)^n$, *i.e.* la somme modulo 2 bits à bits. On appelle cette opération un "Xor" dans la suite. On note également $\bigoplus$ la somme généralisée associée.

Afin de définir les mécanismes de chiffrement et de déchiffrement d'un algorithme cryptographique, les vecteurs de données sont manipulés par des *fonctions*. Nous représenterons par la suite l'entrée d'une fonction par une lettre minuscule et sa sortie par une lettre majuscule (e.g. x et X, respectivement). En cryptographie, il existe deux types de fonctions : les fonctions booléennes et les fonctions vectorielles. Nous les détaillons dans la suite.

Définition 1. ***Fonction booléenne.*** *Soit n un entier naturel positif. Une fonction f définie de* $\mathrm{GF}(2)^n$ *dans* $\mathrm{GF}(2)$ *est appelée une fonction booléenne.*

Une fonction opérant sur des vecteurs binaires peut s'exprimer à l'aide de fonctions booléennes. Toute fonction booléenne peut s'écrire de manière unique sous la forme d'un polynôme en n variables à coefficients dans GF(2). On appelle cette représentation la *forme algébrique normale* (*algebraic normal form* en anglais), notée ANF, et définie par :

$$f(x) = \bigoplus_{\mathrm{I}\subseteq\{1,\ldots,n\}} a_{\mathrm{I}} x^{\mathrm{I}},$$

où les éléments a_{I} appartiennent au corps GF(2) [15]. Cette dernière définition nous conduit à la nouvelle notion de *degré algébrique* d'une fonction booléenne, définie comme suit.

Définition 2. ***Degré algébrique d'une fonction booléenne [15].*** *Le degré d'une ANF, noté s, est appelé degré algébrique d'une fonction booléenne tel que $s = max\{|\mathrm{I}|; a_{\mathrm{I}} \neq 0\}$, où $|\mathrm{I}|$ représente la taille en bits de* I.

Dans un cas plus général, on introduit ci-dessous la notion d'une *fonction vectorielle*.

Définition 3. ***Fonction vectorielle.*** *Soit n et m deux entiers naturels positifs. Une fonction* F *définie de* $\mathrm{GF}(2)^n$ *dans* $\mathrm{GF}(2)^m$ *est appelée une fonction vectorielle. On parle alors de (n,m)-fonction* F.

Étant donnée une (n,m)-fonction F, les fonctions booléennes $(f_1,\ldots,f_m)$ définies pour tout $x \in \mathrm{GF}(2)^n$ par $\mathrm{F}(x) = (f_1(x),\ldots,f_m(x))$ sont appelées les *fonctions coordonnées booléennes*. Ainsi, le degré algébrique d'une (n,m)-fonction F est par définition le degré algébrique maximal des fonctions coordonnées booléennes de F. Nous le définissons plus formellement ci-dessous.

Définition 4. ***Degré algébrique d'une fonction vectorielle [15].*** *Le degré algébrique d'une fonction vectorielle, noté s, est défini tel que $s = max\{|\mathrm{I}|; a_{\mathrm{I}} \neq (0,\ldots,0); \mathrm{I} \subseteq \{1,\ldots,n\}\}$, où $|\mathrm{I}|$ représente la taille en bits de* I.

Les fonctions booléennes et vectorielles représentent ainsi la base des calculs mathématiques des algorithmes cryptographiques. Elles permettent par exemple d'assurer la sécurité en cryptographie symétrique [1], qui repose sur deux principes fondamentaux introduits par Shannon [81] en 1949 : la *diffusion* et la *confusion*. Ces principes ont pour but d'empêcher les analyses basées sur les statistiques des sources de messages clairs et chiffrés. On parle de *biais statistique* que nous définissons comme suit.

Définition 5. ***Biais statistique.*** *Un biais statistique est un moyen d'obtenir de l'information à partir d'un message clair ou chiffré afin de réaliser une attaque en cryptanalyse et retrouver le message secret.*

Par exemple, si un texte est écrit en français, on sait qu'il contient plus de lettres 'e' que de 'z', un attaquant pourrait alors retrouver la clé de chiffrement en analysant le biais statistique d'occurrence des lettres dans le texte chiffré.

Le principe de diffusion permet d'éviter qu'un biais statistique en entrée se retrouve en sortie. L'étude des statistiques en sortie doit en effet donner le moins possible d'informations sur l'entrée. La diffusion est souvent associée à la notion d'*effet d'avalanche* introduite par Feistel [32], qui indique qu'une modification d'un élément en entrée doit avoir des répercussions importantes au fur et à mesure que la donnée se propage dans la structure de l'algorithme. Plus particulièrement, la perturbation d'un bit en entrée doit changer chaque bit en sortie avec une probabilité de 0,5. La diffusion est assurée par des fonctions linéaires (e.g. permutations entre des éléments d'un corps fini).

Le principe de confusion permet de rendre la relation entre la clé secrète de chiffrement et le message chiffré la plus complexe possible. Il est important d'éviter les fonctions ayant une forme algébrique simple, telles que les fonctions linaires. La confusion est donc réalisée par des fonctions non-linéaires (e.g. boîtes-S).

Un algorithme cryptographique symétrique est construit comme une composition de fonctions linéaires et non-linéaires afin de mettre en oeuvre une phase de diffusion et de confusion, respectivement. Cette structure est notamment adoptée par les algorithmes de chiffrement par blocs que nous abordons dans la partie suivante.

2.1.2 Chiffrement par blocs

Le chiffrement par blocs est une des deux grandes familles de chiffrement symétrique [2]. Un algorithme de chiffrement par blocs est dit *itératif* car le message clair en entrée est découpé en blocs de même longueur et chacun de ces blocs est chiffré pendant un nombre de *tours* défini. Pour chaque bloc, on itère une *fonction de tour* paramétrée par une *clé de tour* de taille fixée. Cette clé de tour est créée à partir de la clé cryptographique secrète originale au moyen d'un algorithme dit de *cadencement de clé* (*key schedule* en anglais). De plus, la fonction de tour agit sur un bloc et alterne les opérations de diffusion et de confusion afin d'assurer la sécurité de l'algorithme. À la fin de tous les tours, la sortie du dernier bloc correspond au message chiffré. Le déchiffrement se fait de la même manière, en inversant l'ordre des clés de tours.

Deux schémas de chiffrement par blocs ont été proposés : le schéma de Feistel et le Schéma Substitution-Permutation (SPN). Nous présentons leur fonctionnement dans la suite.

1. Nous rappelons qu'un algorithme est dit symétrique quand il utilise la même clé cryptographique pour les phases de chiffrement et de déchiffrement.
2. L'autre étant le chiffrement à flot.

Schéma de Feistel

Ce schéma de construction a été introduit par Feistel [32] en 1973. Dans sa version *équilibrée*, il divise le bloc de message clair en deux blocs de taille identique. À chaque tour, dans un premier temps, un des deux blocs est combiné avec une version transformée de l'autre bloc. Plus précisément, un premier bloc est chiffré grâce à la fonction de tour, et le résultat est additionné au deuxième bloc. Puis, dans un second temps, à part pour le dernier tour, les blocs obtenus en sortie sont échangés et correspondent à l'entrée du tour suivant. Le schéma de Feistel est illustré en Figure 7.

Remarque 1. *On parle de schéma de Feistel équilibré car la fonction de tour manipule un des deux blocs du message clair. Dans certains algorithmes symétriques, le schéma de Feistel est utilisé dans une forme non-équilibrée, où le bloc de message clair peut être divisé en plusieurs blocs (supérieurs à 2) tel que la fonction de tours traite un nombre de blocs supérieur ou inférieur à deux. Par exemple, pour l'algorithme symétrique chinois Shāngyè Mìmǎ 4 (SM4) [29], le message clair est divisé en quatre blocs et la fonction de tour manipule trois des quatre blocs (voir Chapitre 5).*

Nous détaillons la transformation d'un bloc par le schéma de Feistel. On suppose que le bloc de message clair x est de taille n paire et on le découpe en deux blocs de longueur $\frac{n}{2}$, notés L_0 et R_0. On définit ainsi $x = \mathrm{L}_0 \| \mathrm{R}_0$, où le symbole $\|$ représente la concaténation entre L_0 et R_0. À chaque tour $1 \leq i \leq r$, le bloc d'entrée $(\mathrm{L}_{i-1}, \mathrm{R}_{i-1})$ de taille n est transformé en un bloc $(\mathrm{L}_i, \mathrm{R}_i)$ de taille n. Grâce à un algorithme de cadencement de clé, r clés de tours, notées rk_i, de taille $\frac{n}{2}$ sont générées à partir de la clé cryptographique secrète originale. On note F une fonction de tour définie pour des blocs de taille $\frac{n}{2}$ et prenant en entrée les blocs $(\mathrm{L}_{i-1}, \mathrm{R}_{i-1})$ et la clé de tour rk_i. La transformation du bloc $(\mathrm{L}_{i-1}, \mathrm{R}_{i-1})$ est illustrée en Figure 7 et se fait par les formules suivantes :

$$\mathrm{L}_i = \mathrm{R}_{i-1}, \qquad \mathrm{R}_i = \mathrm{L}_{i-1} \oplus \mathrm{F}(\mathrm{R}_{i-1}, rk_i). \tag{2.1}$$

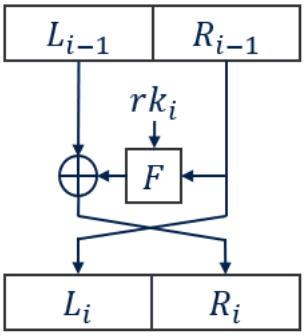


FIGURE 7 – Transformation d'un bloc $(\mathrm{L}_{i-1}, \mathrm{R}_{i-1})$ en un bloc $(\mathrm{L}_i, \mathrm{R}_i)$ par le schéma de Feistel

Comme dit précédemment, au $r^{ème}$ tour, les blocs ne sont pas échangés. Ainsi, au bout des r tours, le chiffré est représenté par $\mathrm{X} = \mathrm{R}_r \| \mathrm{L}_r$.

Remarque 2. *Si on connaît les clés de tours, le schéma de Feistel est inversible. Le déchiffrement de* $\mathrm{X} = \mathrm{R}_r \| \mathrm{L}_r$ *se fait donc exactement de la même manière que le chiffrement en appliquant les clés de tour dans l'ordre inverse. Au dernier tour, on retrouvera bien, sans échanger les blocs, le message clair* $x = \mathrm{L}_0 \| \mathrm{R}_0$.

L'algorithme symétrique le plus connu basé sur le schéma de Feistel est l'algorithme Data Encryption Standard (DES). Dans la suite nous présentons un autre schéma de chiffrement par bloc.

Schéma de substitution-permutation

Le schéma SPN est une autre construction itérative qui transforme un bloc en un nouveau par une série de transformations mathématiques. On suppose qu'un bloc de données de taille n est découpé en l sous-blocs de taille identique. On définit le nombre de tours par la valeur r et on déduit de la clé cryptographique secrète originale k les clés de tour rk_i avec $1 \leq i \leq r$ grâce à un algorithme de cadencement de clé. On pose ensuite $x_0 = x \oplus k$, puis, pour $1 \leq i \leq r$, on calcule $x_i = \mathrm{F}(x_{i-1}, rk_i)$ où F est la fonction de tour. Cette dernière est illustrée en Figure 8 avec ces notations et se compose par les étapes suivantes :

— Le bloc x_{i-1} de taille n est découpé en l sous-blocs de taille plus petite. La fonction de tour commence par l'application de l boîtes de substitution sur chacun des sous-blocs. Ces boîtes sont appelées des *boîtes-S* et sont notées SBox. Ce sont des opérations non-linéaires qui assurent la propriété de confusion. Le bloc de taille n de sortie obtenu est noté u_i.

— Une permutation linéaire (notée P) sur les bits du bloc u_i est ensuite appliquée. On note v_i le résultat.

— Enfin, la clé de tour rk_i est additionnée au bloc v_i. On obtient $x_i = v_i \oplus rk_i$.

Au bout des r tours, le bloc chiffré est représenté par $\mathrm{X} = x_r$.

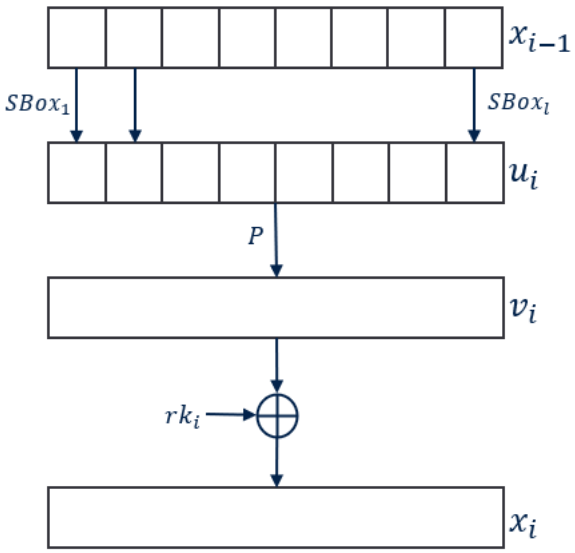


FIGURE 8 – Fonction de tour par le schéma SPN : transformation d'un bloc x_{i-1} en un bloc x_i

L'algorithme symétrique le plus connu basé sur le schéma SPN est l'algorithme AES. Durant cette thèse, nous avons appliqué nos recherches sur cet algorithme. Il est également très étudié par la communauté cryptographique dans l'état de l'art. Pour ces raisons, nous détaillons son fonctionnement dans la suite.

2.1.3 Description de l'algorithme Advanced Encryption Standard (AES)

Le 2 janvier 1997, l'Institut National des Normes et de la Technologie (plus connu sous son nom anglais : National Institute of Standards and Technology (NIST)) a lancé un concours afin de remplacer le DES et développer un nouveau standard de chiffrement, appelé *Advanced Encyption Standard* (AES). Il y était demandé que l'AES chiffre un bloc de message clair de taille 128 bits et supporte une clé cryptographique de taille 128, 192 et 256 bits. De plus, il était nécessaire que l'AES soit libre de droits dans le monde entier. Le processus de sélection de l'AES s'est donc déroulé de manière publique et internationale, et la communauté scientifique était invitée à attaquer et évaluer les performances des différents chiffrements proposés par les candidats. Après une finale

entre cinq participants, le 2 octobre 2000, la proposition de deux chercheurs belges, Daemen et Rijmen a été choisie comme gagnante du concours. Leur chiffrement, portant le nom *Rijndael* [27], est approuvé par le bureau des standards du gouvernement des États-Unis (plus connu sous son nom anglais : Federal Information Processing Standards (FIPS)) et l'Agence de Sécurité Nationale (plus connue sous son nom anglais : National Security Agency (NSA)). Rijndael est notamment utilisé par l'administration fédérale des États-Unis afin de stocker des documents aux informations classées top secrètes.

Comme mentionné précédemment, l'AES suit une structure SPN et chiffre un message clair de 128 bits grâce à une clé cryptographique de taille égale à 128, 192 ou 256 bits. L'AES est un algorithme symétrique itératif. Le nombre de tours est de 10, 12 ou 14 si la clé cryptographique à une taille de 128, 192 ou 256 bits, respectivement. De la même façon, l'algorithme de cadencement de clé diffère en fonction de la taille de la clé cryptographique. De plus, la notation AES-128 permet de se référer à un AES avec une clé cryptographique de 128 bits (AES-192 et AES-256 pour les deux autres tailles de clé). Dans la suite, la notation AES désigne par défaut l'AES-128. Nous expliquons les mécanismes de fonctionnement de l'AES-128 ci-dessous.

Structure algébrique de l'AES-128

L'AES-128 chiffre un bloc de message clair de 128 bits grâce à un bloc de clé cryptographique de 128 bits. Ces deux blocs sont représentés par deux tableaux de 16 octets, chacun composé de quatre lignes et quatre colonnes. Un tel tableau est appelé un *état de données* ou un *état d'AES* dans la suite. Une fonction de tour est appliquée sur un état de donnée à chaque tour. Pour l'AES-128, 10 tours sont nécessaires afin de créer un bloc chiffré. Pour chaque tour une clé de tour est utilisée et générée par un algorithme de cadencement de clé à partir de la clé secrète originale. La procédure de déchiffrement diffère de celle du chiffrement par l'utilisation des clés de tours dans l'ordre inverse et par l'application inverse des calculs effectués dans la fonction de tour.

Concernant les notations, la clé cryptographique secrète originale k est utilisée pour le chiffrement d'un état clair x. Une fonction de tour F est exécutée pendant 10 tours afin de produire un état chiffré X. Une clé de tour est notée rk_i où i correspond au numéro du tour, compris entre 1 et 10. De plus, un état de donnée quelconque est représenté par 16 octets notés par une lettre minuscule et deux indices correspondant à la ligne et la colonne de l'état. Avec ces notations, nous décrivons la fonction de tour de l'AES-128 dans la suite.

Fonction de tour de l'AES-128

La fonction de tour consiste en la composition de 4 opérations, appelées *AddRoundKey*, *SubBytes*, *ShiftRows* et *MixColumns*. La Figure 9 décrit l'ordre dans lequel ces étapes sont appliquées à un état clair d'AES-128. Le chiffrement débute par une opération *AddRoundKey* qui ajoute la clé cryptographique à l'état clair. Ensuite, neuf tours consécutifs sont exécutés. Chacun d'eux applique les quatre opérations citées précédemment et implique une clé de tour. Enfin, un dixième tour est effectué, où l'opération *MixColumns* n'est pas utilisée. Nous détaillons le rôle des quatre opérations ci-dessous.

AddRoundKey. L'ajout de la clé de tour se fait par un Xor entre chaque octet de l'état de données et celui correspondant à la clé de tour. La taille de la clé de tour est donc la même que celle de l'état de données manipulé, soit 128 bits. En remarque, la clé de tour de la première application de *AddRoundKey* correspond à la clé cryptographique originale.

SubBytes. Cette opération est la seule transformation non-linéaire, permettant d'assurer la propriété de confusion. Elle consiste à substituer un octet par un autre et s'applique à chacun des 16 octets d'un état interne. On appelle cette substitution une *boîte-S*. La transformation *SubBytes*

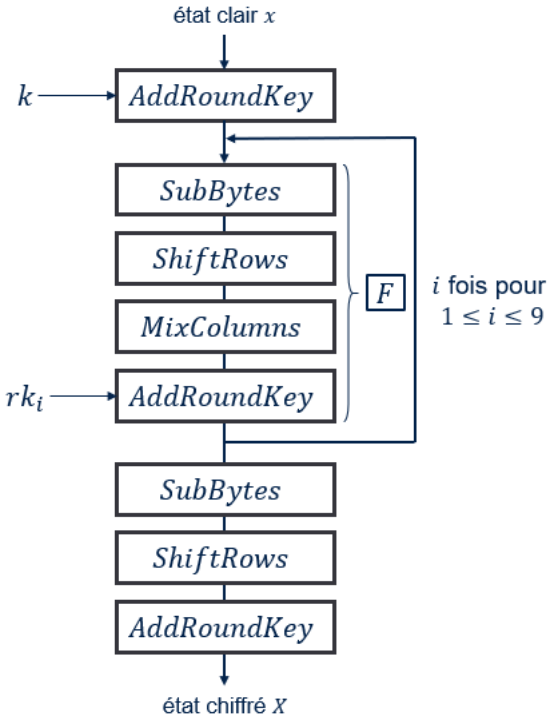


FIGURE 9 – Chiffrement de l'algorithme AES-128

agissant sur un octet est illustrée en Figure 10. Elle s'effectue au moyen d'une même boite-S définie sur GF(2^8).

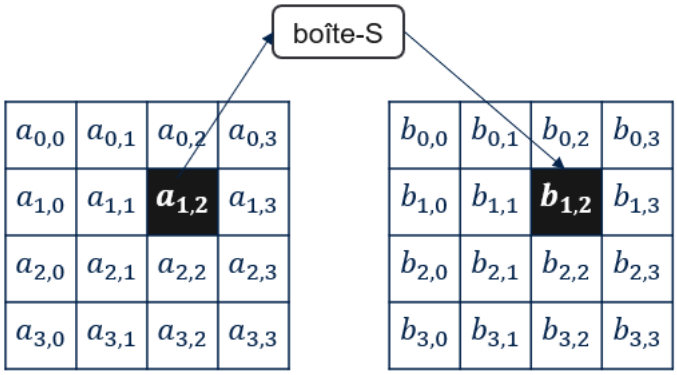


FIGURE 10 – Opération *SubBytes* sur un octet : substitution d'un octet par un autre via la boîte-S

Une boîte-S est souvent pré-calculée avant l'exécution de l'AES-128 et stockée dans une LUT. La Figure 11 représente toutes les substitutions possibles d'un octet dans sa notation hexadécimale sous forme d'un tableau à double entrées.

Il est également possible de représenter la boîte-S de l'AES par la composition d'une fonction inverse et d'une fonction affine. La substitution d'un octet a de l'état interne par la boîte-S de l'AES, notée SBox, et est ainsi définie par la décomposition algébrique suivante :

$$\mathrm{SB}ox(a) = \mathrm{AF}_{\mathrm{AES}}(\mathrm{INV}_{\mathrm{AES}}(a)), \tag{2.2}$$

telle que :

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

FIGURE 11 – Table de correspondance de la boîte-S de l'AES-128 pour un octet en notation hexadécimale, en commençant par la ligne, puis la colonne. Par exemple, l'octet $9a$ est substitué par l'octet $b8$.

— $\mathrm{AF}_{\mathrm{AES}}(a)$ est une transformation affine définie par $\mathrm{AF}_{\mathrm{AES}}(a) = \mathscr{M}_{\mathrm{AES}}(a) \oplus \mathrm{CST}_{\mathrm{AES}}$,

où la matrice $\mathscr{M}_{\mathrm{AES}}$ est définie sur $\mathrm{GF}(2)^8 \times \mathrm{GF}(2)^8$ par $\mathscr{M}_{\mathrm{AES}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$

et la constante $\mathrm{CST}_{\mathrm{AES}}$ est définie par le vecteur colonne de 8 bits $\mathrm{CST}_{\mathrm{AES}} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$.

— L'opération $\mathrm{INV}_{\mathrm{AES}}$ est définie par $\mathrm{INV}_{\mathrm{AES}} : a \in \mathrm{GF}(2^8) \mapsto a^{254} \in \mathrm{GF}(2^8)$, soit l'inverse multiplicatif de a dans le corps $\mathrm{GF}(2^8) \simeq \mathrm{GF}(2)[a]/(a^8 + a^4 + a^3 + a + 1)$.

ShiftRows. Cette transformation linéaire décale de manière cyclique vers la gauche les lignes d'un état de données, comme présenté en Figure 12. En particulier, la première ligne n'est pas décalée, la deuxième est décalée d'une position, la troisième de deux positions et la quatrième de trois

positions. Ces positions sont choisies de manière à assurer la propriété de diffusion dans l'algorithme.

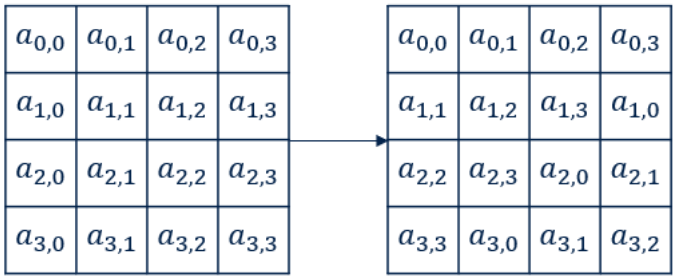


FIGURE 12 – Opération *ShiftRows* sur un état de données de l'AES-128.

MixColumns. Cette fonction linéaire contribue à la propriété de diffusion en mélangeant des colonnes d'un état de données. Elle prend en entrée une colonne de l'état et la multiplie à une matrice, comme illustré dans la Figure 13.

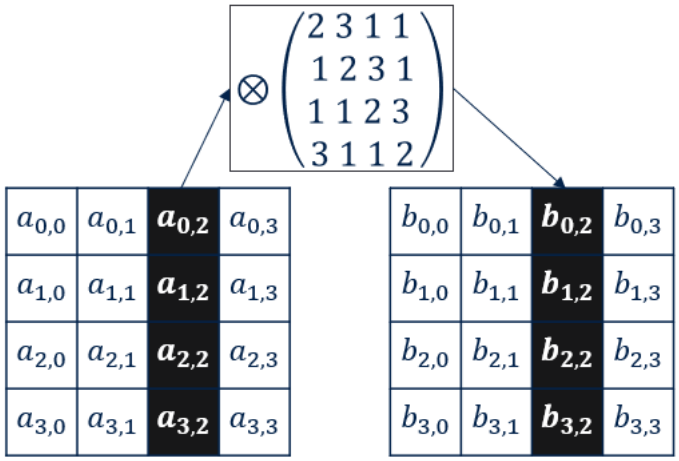


FIGURE 13 – Opération *MixColumns* sur une colonne d'un état de données de l'AES-128.

Il reste à discuter de la diversification de la clé secrète dans l'AES-128. On décrit dans la suite la procédure permettant de générer une clé de tour différente pour chaque tour de l'algorithme.

Algorithme de cadencement de clé de l'AES-128

Cet algorithme prend en entrée la clé cryptographique secrète originale k pour construire 11 clés de tours rk_i de longueur 16 octets, pour $0 \leq i \leq 10$. On rappelle que la première clé de tour rk_0 correspond à k. De plus, l'algorithme de cadencement de clé travaille sur des mots de 4 octets. Ainsi, chaque clé de tour est constituée de 4 mots. La concaténation des 11 clés de tours est appelée *clé étendue* et consiste en 44 mots, notés $w[0], \ldots, w[43]$, où chaque $w[j]$ est un mot $(0 \leq j \leq 43)$. Plus précisément, une clé de tour est définie par :

$$rk_i = w[4 \times i] \| w[4 \times i + 1] \| w[4 \times i + 2] \| w[4 \times i + 3].$$

L'algorithme de cadencement de clé présenté en Algorithme 1 retourne la clé étendue en sortie. Il utilise un tableau Rcon de dix constantes hexadécimales définies en ligne 1 de l'Algorithme 1 ainsi que deux nouvelles opérations appelées *SubWord* et *RotWord* que nous décrivons ci-dessous :

SubWord. Cette fonction applique la boîte-S de l'AES-128 à chacun des quatre octets du mot $w[j]$.

RotWord. Cette fonction exécute une rotation d'une position vers la gauche des quatre octets du mot $w[j]$.

Algorithme 1 Algorithme de cadencement de clé de l'AES-128

Entrée(s) : clé cryptographique k vu comme un état de 16 octets : $k = k[0], ..., k[15]$
Sortie(s) : clé étendue : $w[0], ..., w[43]$

$Rcon = [01000000, 02000000, 04000000, 08000000, 10000000,$
$\quad 20000000, 40000000, 80000000, 1b000000, 36000000]$
pour $i \in \{0, ..., 3\}$ **faire**
$\quad w[i] \leftarrow k[4 \times i] \| k[4 \times i + 1] \| k[4 \times i + 2] \| k[4 \times i + 3]$ ▷ initialisation de $rk_0 = k$.
pour $j \in \{4, ..., 43\}$ **faire**
$\quad tmp \leftarrow w[j-1]$
$\quad$ **si** $j = 0 \ (mod\ 4)$ **alors**
$\quad\quad tmp \leftarrow SubWord(RotWord(tmp)) \oplus Rcon[j/4]$
$\quad w[j] \leftarrow w[j-4] \oplus tmp$
retourner $(w[0], ..., w[43])$

Nous venons de décrire toutes les opérations nécessaires pour calculer un chiffrement d'un message clair d'un AES-128. Le processus est similaire pour un AES-192 ou un AES-256 (voir détails dans [27]). Pour déchiffrer, les opérations et les clés de tours doivent être exécutées dans l'ordre inverse. De plus, les fonctions *SubBytes*, *ShiftRows* et *MixColumns* doivent être remplacées par leur fonctions inverses.

Dans la vie de tous les jours, les algorithmes cryptographiques symétriques sont implémentés sur des circuits électroniques, comme par exemple des cartes à puce. Ces composants connectés sont sujets à des attaques mathématiques et physiques. Nous nous intéressons dans la suite aux attaques physiques. Afin de comprendre leur fonctionnement et les moyens de défense mis en place pour les contrer, il est nécessaire de prendre connaissances de certaines bases en électronique. La section suivante introduit des notions importantes sur les circuits électroniques.

2.2 Introduction à l'électronique

2.2.1 Outils cryptographiques

Il existe deux types de composants électroniques : analogique et numérique. L'électronique analogique étudie des signaux (e.g. tension, courant) à variation continue comme par exemple le signal sonore d'un chant d'un oiseaux. En revanche, lorsque le signal subit un découpage en valeurs discrètes bien définies, nous entrons dans le domaine de l'électronique numérique. Dans la suite nous parlons de circuits électroniques au sens numérique, que nous définissons ci-dessous.

Définition 6. ***Circuit électronique.*** *Un circuit électronique est conçu par un ensemble de portes logiques programmées afin de répondre à une fonction spécifique. Il prend en entrée un signal électrique représenté par sa forme binaire (composé de bits* 0 *et/ou* 1*) et retourne la valeur de la fonction appliquée à ce signal.*

Nous définissons les portes logiques d'un circuit électronique comme suit.

Définition 7. ***Portes logiques.*** *Les portes logiques sont des composants élémentaires d'un circuit électronique. Il existe sept portes logiques de base (AND, OR, XOR, NOT, NAND, NOR et XNOR) qui sont à la base des calculs mathématiques dans un circuit. Leur fonctionnement étant basé sur le passage du courant, elles traitent des informations en langage binaire.*

Nous rentrons plus en détails sur les différents types de portes logiques en Section 2.2.3. Les actions d'un circuit électronique sont rythmées par des *cycles d'horloge*, définis entre deux *fronts d'horloge* le long d'un *signal d'horloge*. Ces notions sont définies ci-dessous et illustrées en Figure 14.

Définition 8. ***Signal, cycle et front d'horloge.*** *Un signal d'horloge est un signal électrique oscillant. Sa période est appelée cycle d'horloge et permet aux calculs d'un circuit électronique de s'effectuer. On parle de front d'horloge montant lorsque le signal passe de* 0 *à* 1 *et de front d'horloge descendant lorsque le signal passe de* 1 *à* 0*.*

Remarque 3. *Entre chaque front d'horloge, avant d'atteindre la valeur* 0 *ou* 1*, le signal peut se trouver dans un état transitoire instable, appelé logic hazards en anglais. On considère que le signal met un certain temps à se propager d'une porte logique à une autre. En pratique, chaque porte logique possède un temps de propagation propre résultant du processus de fabrication,*

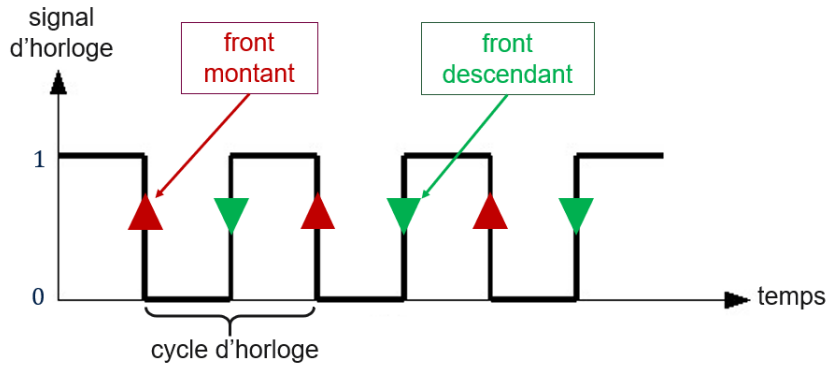


FIGURE 14 – Illustration d'un signal et d'un cycle d'horloge et représentation des fronts d'horloge

Nous nous intéressons ci-dessous à la relation entre l'électronique et la cryptographie. On donne la définition suivante :

Définition 9. ***Outils cryptographiques*** *[55] Les outils cryptographiques sont des circuits électroniques implémentant des algorithmes cryptographiques et capables de stocker des clés cryptographiques secrètes.*

Ces outils cryptographiques peuvent être réalisés de deux façons différentes. D'une part, sur un ensemble de composants électroniques inter-connectées à l'aide d'un circuit imprimé dont le but est de remplir une fonction. Par exemple, la boîte noire transactionnelle (*hardware security module (HSM)* en anglais) est un appareil électronique offrant un service de sécurité qui consiste à générer et stocker des clés cryptographiques à la demande de l'utilisateur (e.g. un vendeur de carte bancaire). D'autre part les circuits peuvent être programmés sur une puce isolée, comme par exemple les cartes à puce (e.g. carte SIM, carte vitale). Dans la suite, nous nous concentrons sur des outils cryptographiques implémentés sur une puce isolée.

Il y a essentiellement deux façons d'implémenter un circuit électronique sur une puce isolée. La première est de l'intégrer sur un *circuit intégré spécialisé* (en anglais, Application-Specific Integrated Circuit (ASIC)). Un ASIC est programmé pour répondre à une seule fonction ce qui lui apporte un avantage précieux pour les entreprises. En effet, la personnalisation du circuit donne une confidentialité au concepteur et une protection industrielle. Par exemple, les cricuits intégrés dans les téléphones portables sont des ASICs. La deuxième façon est d'utiliser un circuit logique programmable appelé Field-Programmable Gate Array (FPGA) en anglais. De la même famille que les ASICs, les FPGAs ont été inventés par la société Xilinx. Contrairement aux ASICs, le FPGA présente l'avantage d'être reprogrammable pour une autre fonction. Ainsi, les ingénieurs peuvent

définir le comportement des cellules du circuit et la façon de les connecter entre elles afin de répondre à la fonction désirée. Les FPGAs sont cependant plus chers et plus lents que les ASICs.

Dans la sous-section suivante, nous voyons les différentes étapes de conception et d'implémentation d'un circuit électronique, qu'il soit programmé sur un ASIC ou un FPGA.

2.2.2 Conception d'un circuit électronique

La conception d'un circuit électronique peut se résumer en l'application des quatre étapes suivantes :

Étape 1 : Spécifications. Il s'agit d'un fichier texte permettant de comprendre et modéliser le circuit afin de s'assurer de la fonctionnalité voulue. Les spécifications contiennent également des grilles de tests afin qu'un ingénieur puisse valider les étapes de calculs intermédiaires de son circuit.

Étape 2 : Description Register-Transfert Level (RTL). Cette étape décrit le comportement du circuit électronique grâce à un langage de description matérielle (e.g. VHDL, Verilog). Cette modélisation définit toutes les opérations et les envois de signaux en entrée et en sortie du circuit. Le code RTL fourni peut ensuite être simulé afin de vérifier sa fonctionnalité. Un stimulateur permet d'assurer cette vérification en simulant dans le temps chaque signal en fonction de stimuli(s) d'entrée.

Étape 3 : Synthèse logique. Cette étape va transformer la description RTL en une description au niveau des portes logiques. On parle en anglais de *netlist*, que l'on définit comme une liste de portes logiques et de toutes les connexions entre elles. Pour cette conversion, il est nécessaire de connaître au préalable la technologie utilisée (e.g. ASIC, FPGA) car une bibliothèque cible de portes logiques doit être sélectionnée en fonction de la plateforme choisie.

Étape 4 : Placement et routage. C'est le dernier processus qui consiste à obtenir une conception physique du circuit. Dans le cas d'un circuit ASIC, les portes logiques de la netlist sont placées de manière physique à un endroit précis sur le composant. On appelle cette étape le *placement*. De plus, les liens entre ces portes sont créées en introduisant des connexions métalliques entre les éléments. Cette étape s'appelle le *routage*. Le résultat se présente sous la forme d'une description géométrique de toutes les portes et fils du circuit, on parle de *circuit dessiné* (*layout* en anglais). Dans le cas d'un circuit FPGA, la procédure est légèrement moins complexe dans le sens où un lien direct peut être établi entre les éléments de la netlist et ceux du FPGA. Cela permet de repérer facilement quelle connexion physique du FPGA convient le mieux à une connexion donnée de la netlist. Le résultat de cette étape se présente sous forme d'un fichier constitué d'une succession de bits qui peut être utilisé pour configurer le FPGA. Ce fichier est appelé *bitstream* en anglais.

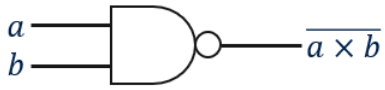
Dans la suite, nous rentrons plus en détails sur l'implémentation des portes logiques d'un circuit et leur technologie de fabrication. Ces nouvelles connaissances nous permettront de mieux comprendre d'où vient la quantité d'énergie consommée par un circuit, qui sera exploitée par un attaquant en cryptanalyse (voir Section 2.3.1).

2.2.3 Portes logiques et technologie CMOS

Comme vu dans les Définitions 6 et 7, les circuits électroniques sont construits grâce à des portes logiques. Ces dernières prennent un ou plusieurs signaux en entrée et les transforment en un ou plusieurs nouveaux signaux grâce à une fonction booléenne définie. On peut les représenter par un symbole et une table de vérité. Cette dernière a pour rôle de montrer la correspondance entre la sortie et toutes les combinaisons de valeurs que peuvent prendre la ou les entrées. Il existe

deux catégories de portes logiques, expliquées ci-dessous.

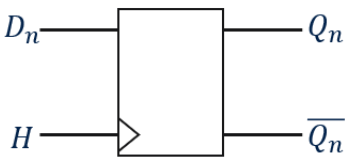
Les portes appelées *portes combinatoires* implémentent des fonctions booléennes. Leur nom vient du fait que les sorties de ces portes sont calculées par une combinaison logique des entrées. Un exemple connu est celui de la fonction NAND, illustrée en Figure 15. Cette porte prend deux entrées a et b et retourne 1 si au moins une des deux entrées est égale à 0, sinon la fonction retourne 0. Cette fonction est dite *universelle* car elle permet de reconstituer toutes les autres portes logiques. Il est donc possible d'implémenter un circuit entièrement avec des portes NAND. Mesurer les performances matérielles d'un circuit en terme de surface revient alors à compter le nombre de portes NAND, nécessaires à l'implémentation du circuit. On parle alors de *portes équivalentes* (ou Gates Equivalent (GE)).



a	b	$\overline{a \times b}$
0	0	1
0	1	1
1	0	1
1	1	0

FIGURE 15 – Symbole (à gauche) et table de vérité (à droite) de la porte logique combinatoire NAND

La deuxième catégorie de portes logiques sont les *portes séquentielles*, dont les sorties ne dépendent pas seulement des données en train d'être traitées en entrée, mais aussi des données impliquées précédemment dans le circuit. Ces portes sont en effet capables de mémoriser des valeurs en entrée à n'importe quel instant lors du cycle précédent. Un exemple connu est celui de la bascule D (pour *Data*), illustré en Figure 16. Cette porte permet de stocker un *bit d'information*[3] à chaque front d'horloge montant (ou descendant). Sur la Figure 16, si le front d'horloge H est montant, la sortie prend la valeur du bit d'entrée à l'instant n, noté D_n. Sinon, la sortie mémorise la valeur du bit d'entrée de l'instant précédent, représenté par Q_{n-1}. En remarque, dans la table de vérité, on note X un état quelconque, c'est-à-dire dont la valeur est ignorée pour le calcul du résultat en sortie.



D_n	H	Q_n	$\overline{Q_n}$
0	Front montant	0	1
1	Front montant	1	0
X	Front descendant	Q_{n-1}	$\overline{Q_{n-1}}$
X	Front descendant	Q_{n-1}	$\overline{Q_{n-1}}$

FIGURE 16 – Symbole (à gauche) et table de vérité (à droite) de la porte logique séquentielle bascule D

En combinant plusieurs portes séquentielles en parallèle, on peut stocker une valeur codée sur plusieurs bits. On crée alors un *registre*, que nous définissons ci-dessous.

3. En théorie de l'information, l'entropie de Shannon [80] est une mesure de l'incertitude d'un évènement en fonction de la connaissance que nous avons. Pour lever cette incertitude, nous devons connaître une quantité d'information correspondant à un nombre de bit d'information.

Définition 10. ***Registre.*** *Un registre est un emplacement de mémoire interne à un circuit électronique, permettant de stocker un ou plusieurs bits.*

En implémentation matérielle, savoir optimiser le nombre de registres dans un circuit électronique est un défi important. Le but est en effet de minimiser le coût de l'implémentation, en terme de surface et de temps d'exécution. Nous avons vu précédemment que les performances d'un circuit se mesurent en nombre de portes équivalentes NAND. Il est également nécessaire de prendre en compte le nombre de registres utilisés dans le circuit.

On appelle *circuit combinatoire* un circuit composé d'un ensemble de portes combinatoires et *circuit séquentiel* un circuit combinatoire associé à un registre. Ces deux types de circuits sont illustrés en Figure 17.

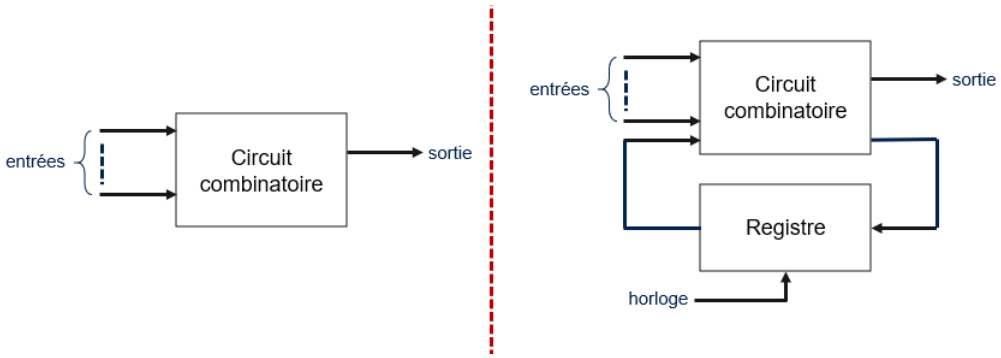


FIGURE 17 – Représentation d'un circuit combinatoire (à gauche) et d'un circuit séquentiel (à droite)

De plus, les portes logiques (combinatoires et séquentielles) sont implémentées grâce à des *transistors*. Un transistor est un composant à trois bornes composé de deux pattes métalliques et d'un *semi-conducteur*. Un semi-conducteur est un matériau ni conducteur ni isolant (e.g. silicium), qui peut être "dopé" pour devenir plus ou moins conducteur. Son rôle est de laisser passer le courant entre les deux pattes métalliques, ou au contraire de ne pas le transmettre. Il existe deux types de dopage, le dopage négatif de type N et le dopage positif de type P. En combinant deux matériaux dopés de façon différente, on peut faire des composants électriques comme des transistors. Les transistors les plus connus ont une structure *métal-oxyde-semiconducteur* (MOS). Plus encore, la technologie électronique emploie majoritairement des transistors MOS de type N (NMOS) et de type P (PMOS). Cette technologie est appelée Complementary Metal-Oxide Semiconductor (CMOS).

Dans la section suivante, nous étudions pourquoi les circuits CMOS sont particulièrement vulnérables aux attaques cryptographiques.

2.3 Les attaques par observations

2.3.1 Introduction

Une personne essayant d'extraire la clé secrète d'un outil cryptographique (voir Définition 9) est appelé un *attaquant* (ou un *adversaire*). Toute tentative d'extraction de la clé secrète de manière non autorisée est appelée une *attaque*. Les techniques employées afin d'atteindre cet objectif sont nombreuses. On peut les classer en deux catégories, les attaques mathématiques et les attaques physiques. Dans ce manuscrit, nous nous intéressons à ces dernières, que nous divisons en deux groupes, les attaques passives et les attaques actives :

— D'une part, lors d'une attaque passive, la clé secrète est retrouvée par l'observation d'émanations physiques mesurées sur l'outil cryptographique. Dans cette thèse, ces émanations

sont appelées des *fuites*. Il existe en effet des canaux auxiliaires qui donnent de l'information accessible à un attaquant. En 1996, Kocher [46] a ainsi introduit les premières attaques par observation, Side-Channel Analysis (SCA) en anglais, en exploitant le temps d'exécution ou la consommation électrique [47] d'un circuit électronique dans le but de retrouver des informations secrètes. Il existe d'autres types d'attaque à partir de l'accès au composant, comme par exemple l'exploitation des émanations d'émissions électromagnétiques [2, 34]. Les attaques par observation présentent l'avantage d'être faciles à mettre en oeuvre et de ne pas coûter cher à l'attaquant.

— D'autre part, une attaque active consiste à perturber le comportement de l'outil cryptographique afin d'exploiter un comportement non attendu menant à la clé secrète. Boneh *et. al.* [12] introduisent ainsi en 1997 les attaques par fautes, *fault attacks* en anglais, en injectant une faute au sein du circuit électronique dans le but de retrouver des informations secrètes. Afin d'atteindre cet objectif, on peut par exemple augmenter la température du circuit, injecter une perturbation électromagnétique ou à l'aide d'un laser.

Nous nous concentrons dans la suite sur les attaques passives. Les trois plus connues sont les attaques par analyse de temps [46], par analyse de consommation de courant globale [47] (mesure de la consommation du composant) et locale [34, 71] (mesure d'ondes électromagnétiques). Nous décrivons le fonctionnement des attaques par analyse de consommation de courant (globale et locale) sur des circuits électroniques CMOS implémentant des algorithmes cryptographiques.

2.3.2 Analyse de la consommation de courant d'un circuit CMOS

Une attaque par observation a pour but d'extraire les données secrètes à partir d'un outil cryptographique. Nous avons vu que ce dernier laisse par exemple fuir de la consommation de courant électrique, pouvant être exploitée par un attaquant. Cette fuite est due à la technologie CMOS. En effet, le montant total de consommation de courant dépend du nombre de portes logiques CMOS du circuit, des connexions entre elles et de la manière dont elles sont construites. À un temps précis, nous pouvons représenter ce qu'il se passe au niveau de la sortie d'une porte logique CMOS par quatre transitions différentes :

— La transition du signal 0 au signal 0, notée $0 \mapsto 0$
— La transition du signal 1 au signal 1, notée $1 \mapsto 1$
— La transition du signal 0 au signal 1, notée $0 \mapsto 1$
— La transition du signal 1 au signal 0, notée $1 \mapsto 0$

On distingue deux types de consommation de courant au niveau d'une porte logique CMOS. Lorsqu'il n'y a pas de changement d'état (*i.e.* $0 \mapsto 0$ et $1 \mapsto 1$), la porte consomme faiblement du courant, on parle de *consommation statique*. En revanche, lorsque la valeur du signal électrique change (*i.e.* $0 \mapsto 1$ et $1 \mapsto 0$), la porte consomme plus intensément de courant, on parle de *consommation dynamique*. Cette dernière consommation dépend donc des données exécutées dans le circuit CMOS et a un impact important sur la consommation totale de courant mesurée par un attaquant.

La section suivante décrit la mise en place d'une attaque par observation par analyse de consommation de courant et explique comment mesurer cette dernière à partir d'un circuit CMOS.

2.3.3 Mise en place d'une attaque par analyse de consommation de courant

La Figure 18 représente les composants impliqués dans la mise en place d'une attaque par analyse de consommation de courant. Nous décrivons le rôle de chacun d'eux dans la suite.

— **Outil cryptographique.** Il s'agit par exemple du circuit électronique CMOS à attaquer. Il contient l'algorithme cryptographique ciblé par l'attaque et dispose d'une interface afin de communiquer avec un ordinateur. Son rôle est d'exécuter les données envoyées par l'ordinateur et de lui renvoyer les données chiffrées.

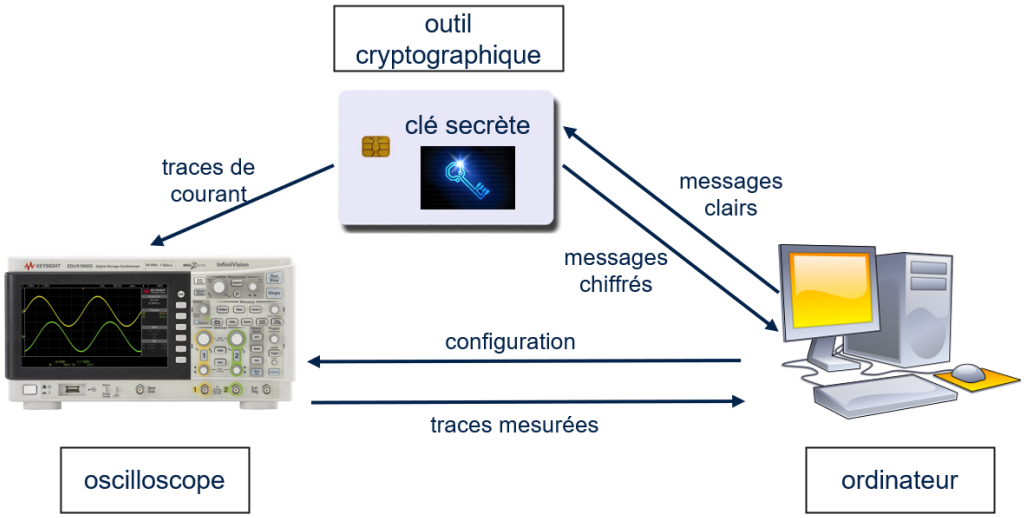


FIGURE 18 – Mise en place d'une attaque par analyse de consommation de courant sur un outil cryptographique

— **Appareil de mesure de courant.** Cette étape permet de mesurer, à l'aide par exemple d'un oscilloscope, la consommation de courant qui fuite lors de l'exécution de l'algorithme cryptographique sur le circuit électronique. Cette consommation est appelée *trace (de courant)* et est envoyée à l'ordinateur afin d'être stockée.

— **Interface.** Une interface, comme par exemple un ordinateur, contrôle toute la configuration de l'attaque et stocke toutes les données mesurées. Son rôle est de communiquer avec l'outil cryptographique afin de lui transmettre des messages clairs et de stocker les messages chiffrés correspondants. En parallèle, l'ordinateur configure un oscilloscope de manière à ce que ce dernier lui transmette les traces mesurées sur l'outil cryptographique.

Pour résumer, la liste des composants ci-dessus interagissent entre eux de la façon suivante.

— **Étape 1 :** L'ordinateur configure l'oscilloscope et envoie des entrées cryptographiques à l'outil pour lancer l'exécution de l'algorithme sur celui-ci.

— **Étape 2 :** Pendant l'exécution de l'algorithme, l'oscilloscope enregistre les traces de courant électrique mesurées sur l'outil cryptographique.

— **Étape 3 :** L'ordinateur reçoit et stocke les données chiffrées de l'algorithme cryptographique ainsi que les traces mesurées par l'oscilloscope.

Ces étapes sont répétées autant que nécessaires pour réaliser l'attaque par observation. Afin que cette dernière soit le plus facile possible, il est important d'assurer la qualité des traces mesurées en étape 3. Nous discutons de ce point ci-dessous.

Une porte logique CMOS d'un circuit électronique met moins d'une nanoseconde à retourner un signal électrique. Ainsi, la consommation de courant électrique d'une porte logique contient des signaux dits à *haute fréquence*. Ces derniers sont difficiles à mesurer en pratique car leur structure peut être altérée lors de leur propagation de la porte logique à l'oscilloscope. Leur interaction avec l'environnement est associée à la notion de *bruit* dans le circuit électronique (on parle de *switching noise* en anglais). De plus, du bruit peut également venir des composants électroniques utilisés lors de l'attaque (on parle d'*electronic noise* en anglais). Cela peut être par exemple dû à un générateur délivrant une tension peu stable (e.g. câble USB entre l'ordinateur et le circuit). Ainsi, la qualité des traces de courant mesurées dépend de la quantité du bruit présent dans celles-ci. Dans la section suivante, nous montrons qu'il est possible pour un attaquant de modéliser ces traces de courant.

2.3.4 Modèles de fuite

Nous avons vu que lors d'une attaque par observation, un attaquant peut par exemple exploiter la consommation de courant des données exécutées par l'outil cryptographique. Cette consommation peut contenir du bruit et peut être modélisée à l'aide d'un *modèle de fuite*, formalisé par Micali et Reyzin dans [62]. Nous donnons une définition ci-dessous.

Définition 11. ***Modèle de fuite.*** *Un modèle de fuite décrit comment les transitions à la sortie des portes logiques CMOS sont reliées à la consommation de courant mesurée.*

En particulier, nous introduisons le modèle de la distance de Hamming et celui du poids de Hamming.

Modèle de la distance de Hamming. En partageant la simulation du circuit CMOS en plusieurs intervalles de temps, une trace de courant peut être générée et décrite par le nombre de transitions effectives sur le circuit dans chaque intervalle de temps. Ainsi, le modèle de la distance de Hamming compte le nombre de transitions ($0 \mapsto 1$ et $1 \mapsto 0$) qui apparaissent dans un circuit CMOS dans un certain intervalle de temps. Il est supposé que ces transitions consomment la même quantité d'énergie. On note $\mathrm{HD}(v_1(t), v_0(t-1))$ la distance de Hamming entre la valeur v_0 et la valeur v_1 dans l'intervalle de temps $[t-1, t]$. Dans [13] Brier *et. al.* définissent la consommation de courant $\mathscr{L}$ mesurée sur le circuit à un temps t telle que :

$$\mathscr{L}(t) = a \times \mathrm{HD}(v_1(t), v_0(t-1)) + \varepsilon,$$

où a est une constante liée au circuit CMOS et ε est une variable aléatoire qui correspond à la quantité de bruit dans la trace.

Il est nécessaire de pouvoir faire des hypothèses sur deux valeurs consécutives (e.g. $v_0 \mapsto v_1$) du calcul attaqué pour appliquer le modèle de la distance de Hamming. Il existe des cas où l'une des deux valeurs consécutives est égale à 0. On peut alors utiliser le modèle du poids de Hamming afin de décrire une trace de courant (dans ce modèle, on fait l'hypothèse implicite que l'état de départ est constant ou, tout au moins, ne dépend pas de la clé secrète attaquée). Nous le décrivons ci-dessous.

Modèle du poids de Hamming. On regarde ici la présence ou non de courant contenue dans les transistors. Ce phénomène permet de stocker dans les circuits électroniques les bits d'une variable. Lorsqu'on change la valeur de cette dernière, les transistors se déchargent ($1 \mapsto 0$) ou se chargent ($0 \mapsto 1$). Le poids de Hamming d'une variable binaire v_1 à l'instant t, noté $\mathrm{HW}(v_1(t))$, est une fonction qui compte le nombre de bits non-nul de v_1, c'est à dire le nombre de transistors à charger pour stocker la suite de bits. On remarque alors que le modèle du poids de Hamming est un cas particulier du modèle de la distance de Hamming où $\mathrm{HD}(0, v_1(t)) = \mathrm{HW}(v_1(t))$. Ce modèle est très utilisé car la fuite exploitée par un attaquant est souvent liée à la façon dont les données manipulées sont stockées. Avec les mêmes notations que précédemment, Messerges *et. al.* définissent dans [59] la consommation de courant $\mathscr{L}$ mesurée sur le circuit CMOS à un temps t tel que :

$$\mathscr{L}(t) = a \times \mathrm{HW}(v_1(t)) + \varepsilon,$$

D'autres types de modélisation existent, on peut par exemple étudier la valeur du bit de poids le plus faible de v_1 (modèle LSb) ou de poids le plus fort (modèle MSb), ou encore directement la valeur de la donnée.

Dans l'étape 4 de la Section 2.3.6, nous montrons dans quel contexte un modèle de fuite est utilisé.

Nous pouvons distinguer deux types d'attaques par observation, les attaques par simple observation (en anglais, Simple Power Analysis (SPA)) et les attaques statistiques par observations (en anglais, Correlation Power Analysis (CPA) et Differential Power Analysis (DPA)). Nous décrivons leur mode de fonctionnement dans la suite.

2.3.5 Attaques par simple observation (SPA)

Dans cette partie, on considère qu'un algorithme cryptographique est implémenté sur un circuit électronique CMOS. Introduit par Kocher *et. al.* [47], les attaques par observation simple consistent comme leur nom l'indique à retrouver la clé cryptographique secrète à partir de l'observation d'une seule trace de courant, mesurée sur le circuit. Elles exploitent en particulier les différences de motifs au sein d'une trace de courant.

Nous illustrons une attaque SPA sur l'algorithme asymétrique RSA, décrit par Rivest *et. al.* [76] en 1978 et très utilisé pour l'échange de données confidentielles sur Internet. Le mécanisme de signature RSA est basé sur l'élévation du message chiffré (ici c) à une certaine puissance qui sert de clé secrète (ici d), modulo un entier (ici n). Cette opération est appelée *exponentiation modulaire*. Afin d'optimiser la place en mémoire nécessaire à l'exécution de cette opération sur le circuit, l'*algorithme d'exponentiation binaire* présenté en Algorithme 2 peut être appliqué. La consommation de courant observée par Kocher *et. al.* [47] lors de l'exécution de l'algorithme est présentée en Figure 19.

Algorithme 2 Algorithme d'exponentiation binaire

Entrée(s) : $c, n \in \mathbf{N}, c < n, d = \sum_{i=0}^{n-1} a_i 2^i, a_i \in \{0,1\}$
Sortie(s) : $x = c^d \ mod(n)$
1: **pour** $i \in \{n-2, ..., 0\}$ **faire**
2: $x = x \times x \ mod(n)$ ▷ Mise au carré.
3: **si** $a_i = 1$ **alors**
4: $x = x \times c \ mod(n)$ ▷ Multiplication.
retourner x

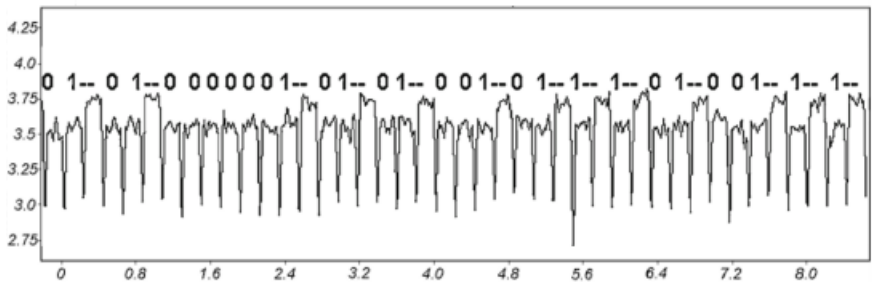


FIGURE 19 – Trace SPA - Simple observation de consommation de courant électrique lors de l'exécution de l'algorithme d'exponentiation binaire. Temps en ms en abscisse, consommation de courant en mA en ordonnée [47].

Dans l'Algorithme 2, la clé secrète d est décomposée dans sa forme binaire. On remarque que lorsqu'un bit de clé est égal à 1, un calcul supplémentaire doit s'effectuer par rapport au cas où un bit de clé est nul. On observe sur ces traces qu'il est possible de faire la distinction entre l'opération de la multiplication et celle du carré. Ainsi, il est possible pour un attaquant de retrouver la clé secrète bit à bit à partir de l'observation de la consommation de courant. La Figure 19 présente clairement les différences de consommation de courant lorsque le circuit traite un bit de clé non-nul ou un bit de clé nul. Dans cette figure, la consommation dépend des valeurs de la clé secrète

et on obtient $d = (01010000001010100101110100111)$.

Parfois la trace de courant observée lors d'une attaque SPA n'est pas suffisante pour retrouver la clé secrète manipulée par l'outil cryptographique. En effet, dans l'exemple précédent on observe que l'algorithme est dépendant de la valeur de la clé et que son déroulement va changer en fonction des valeurs possibles. C'est ce qui permet à l'attaquant de retrouver la clé bit à bit. Ce n'est généralement pas le cas pour l'AES où l'algorithme va toujours se dérouler de la même manière, seules les données manipulées vont changer. Par exemple, la Figure 20 illustre une seule trace de courant capturée depuis un circuit CMOS exécutant un chiffrement de l'AES-128. Une attaque SPA nous permet d'avoir des informations sur le type d'algorithme. En effet, sur la Figure 20, nous pouvons distinguer 9 motifs similaires dans l'encadré rouge, représentant les 9 premiers tours identiques de l'AES-128. Le dernier tour étant différent (*i.e.* l'opération *MixColumns* est manquante), son motif de consommation de courant se différencie des autres légèrement. Le motif précédant les 9 tours correspond au chargement de la clé secrète et du message à chiffrer dans les registres ainsi qu'à l'exécution de la première opération *AddRoundKey*. En revanche, l'attaque SPA ne permet pas à un attaquant de déduire la clé secrète de l'AES-128. Un exemple d'attaques plus appropriées sont les *attaques statistiques par observations*, que nous décrivons dans la suite.

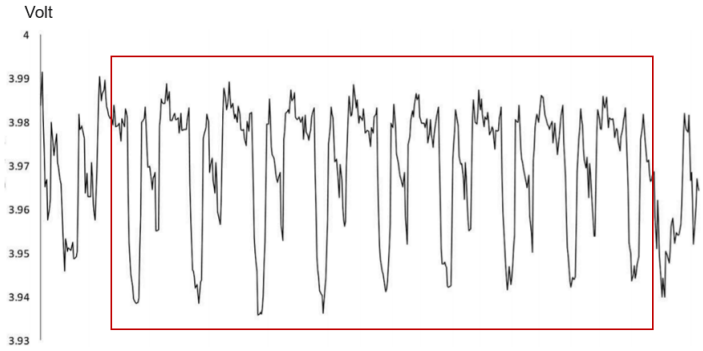


FIGURE 20 – Trace SPA - Consommation simple de courant électrique lors d'un chiffrement de l'AES-128 au cours du temps

2.3.6 Attaques statistiques par observations (DPA/CPA)

Contrairement aux attaques par simple observation, les attaques statistiques par observations requièrent plusieurs traces de courant afin de retrouver la clé secrète. Elles présentent également l'avantage d'être performantes en cas de présence de bruit important au sein des traces. Lors d'une attaque SPA, un attaquant exploite les motifs d'une trace de courant lors de l'exécution d'un algorithme cryptographique au fil du temps. Dans une attaque statistique, la forme de la trace au cours du temps n'est pas si importante car l'attaque se focalise sur la consommation de courant d'une opération précise, qui dépend de la donnée sensible manipulée à un moment fixé. Ainsi, les attaques statistiques par observations analysent un lien entre les traces de courant et les données cryptographiques. De plus, un attaquant ne peut pas reconstituer la clé secrète d'un coup, mais plutôt par morceaux. Cela est dû à la conception du circuit électronique, qui ne peut pas manipuler des données de taille 128 bits. Par exemple, un attaquant peut reconstituer une clé secrète de 128 bits (*i.e.* pour l'AES) en retrouvant 16 morceaux de 8 bits chacun. Cette méthode est appelée *"diviser pour mieux régner"*. La stratégie d'une attaque statistique par observations peut s'expliquer en 5 étapes. La première est expliquée ci-dessous.

Étape 1 : Choisir une opération précise. La première étape d'une attaque statistique par observations est de cibler la sortie d'une opération au sein de l'algorithme cryptographique exécuté sur le circuit. Cette opération doit manipuler des données reliées à la clé secrète. On note ainsi $f(x,k)$ la fonction ciblée prenant en paramètre une donnée x connue et la clé secrète k inconnue par

définition. Un attaquant va analyser la consommation de courant associée à la sortie de f appelée *valeur intermédiaire.*

Les 4 étapes suivantes sont illustrées dans la Figure 21 et décrites ci-dessous. Nous précisons dans la figure quelle partie de l'attaque s'effectue directement sur le composant électronique et quels calculs peuvent se réaliser au contraire en dehors (sur un ordinateur les calculs sont plus rapides que sur un circuit électronique).

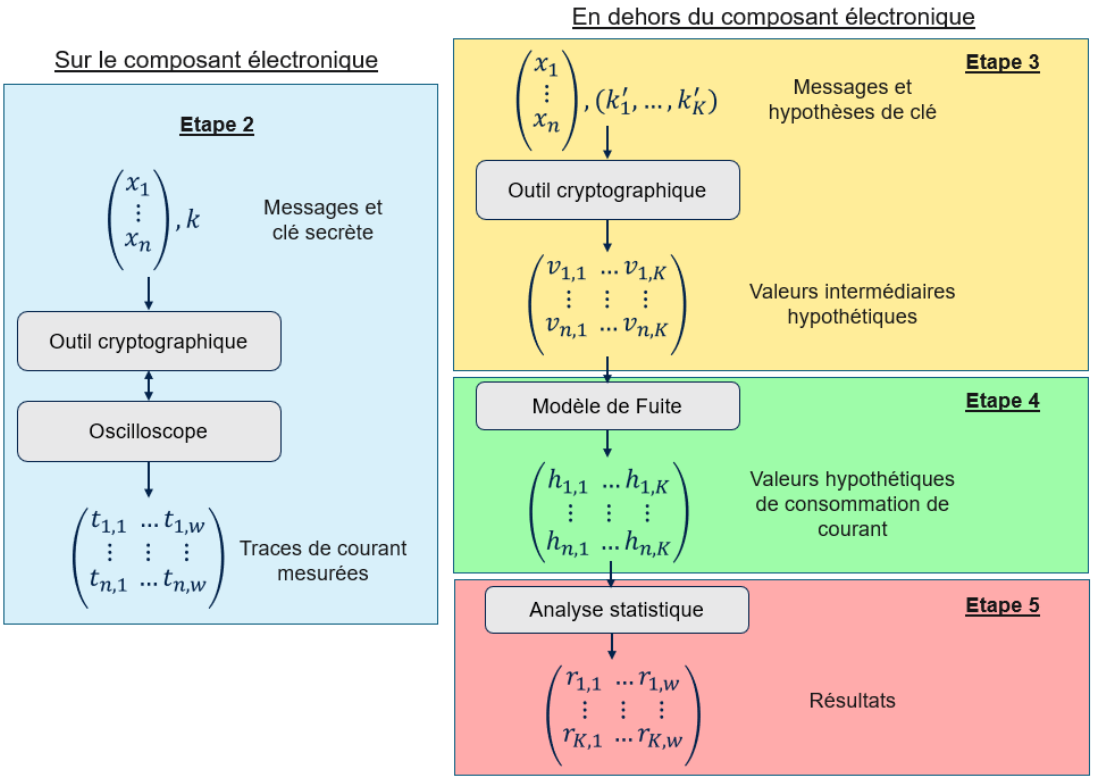


FIGURE 21 – Stratégie des attaques statistiques par observations - illustration des étapes 2 à 5 ci-dessous

Étape 2 : Mesure de la consommation de courant électrique. Cette deuxième étape correspond aux phases 1 à 3 décrites en Section 2.3.3. En effet, la consommation de courant électrique de l'outil cryptographique est mesurée lors du chiffrement (ou déchiffrement) de n messages.
Pour chacun des processus de chiffrement (ou déchiffrement), un attaquant doit être capable de connaître la donnée x impliquée dans le calcul de la valeur intermédiaire choisie en première étape. On note $\mathscr{X} = (x_1, ..., x_n)$ ces données connues telle que chaque x_i correspond à la donnée au $i^{ème}$ chiffrement (ou déchiffrement).
En parallèle, à chaque chiffrement (ou déchiffrement), l'attaquant enregistre et stocke les traces de courant mesurées par l'oscilloscope. On note $\mathscr{T}_i = (t_{i,1}, ..., t_{i,w})$ la mesure d'une trace de courant associée à la donnée x_i, où w correspond à la taille de la trace (*i.e.* le nombre de points de la trace). Au final, on peut regrouper les n traces mesurées dans une matrice $\mathscr{T}$ de taille $n \times w$:

$$\mathscr{T} = \begin{pmatrix} t_{1,1} & \dots & t_{1,w} \\ t_{2,1} & \dots & t_{2,w} \\ \vdots & \vdots & \vdots \\ t_{n,1} & \dots & t_{n,w} \end{pmatrix}$$

Remarque 4. *Il est important lors d'une attaque statistique par observations que les traces mesurées soient correctement alignées. Cela signifie que la consommation de courant de chaque colonne de la matrice $\mathscr{T}$ doit être associée à la même opération. La configuration de l'oscilloscope doit donc*

faire en sorte que toutes les traces enregistrées agissent sur la même séquence d'opérations à chaque chiffrement (ou déchiffrement).

Étape 3 : Calculer les valeurs intermédiaire hypothétiques. Cette troisième étape consiste à calculer toutes les valeurs intermédiaires en fonction de tous les choix possibles de la clé secrète k. On parle *d'hypothèses de clé* et on les note $k' = (k'_1, ..., k'_K)$, où K correspond au nombre total de choix possibles pour k'. Ainsi, étant donnés les vecteurs $\mathscr{X}$ et k', un attaquant peut calculer toutes les valeurs intermédiaires $v_{i,j} = f(x_i, k'_j)$ pour les $i \in [1..n]$ chiffrements et les $j \in [1..K]$ hypothèses de clé. Les résultats sont représentés par une nouvelle matrice $\mathscr{V}$ de taille $n \times K$:

$$\mathscr{V} = \begin{pmatrix} v_{1,1} & \dots & v_{1,K} \\ v_{2,1} & \dots & v_{2,K} \\ \vdots & \vdots & \vdots \\ v_{n,1} & \dots & v_{n,K} \end{pmatrix}$$

Chaque colonne j de $\mathscr{V}$ contient les valeurs intermédiaires x_i exécutées sur le circuit pendant les n chiffrements et calculées pour l'hypothèse de clé k'_j. Ainsi, le but de l'attaque est de retrouver quelle est la bonne colonne de $\mathscr{V}$ utilisée dans les n chiffrements pour lesquels la consommation de courant a été mesurée. Cela permettra à l'attaquant de déduire quel hypothèse k'_j correspond à la clé secrète originale k.

Étape 4 : Transformer les valeurs intermédiaires hypothétiques en valeurs hypothétiques de consommation de courant. Cette quatrième étape consiste à transformer la matrice $\mathscr{V}$ en une nouvelle matrice $\mathscr{H}$ contenant des valeurs hypothétiques de consommation de courant. L'attaquant va pour cela modéliser la consommation de courant des valeurs calculées dans l'étape précédente grâce à un modèle de fuite. Nous avons présenté les deux modèles les plus utilisés en cyptanalyse en Section 2.3.4. En utilisant par exemple le modèle du poids de Hamming (HW), chaque valeur intermédiaire hypothétique $v_{i,j}$ est transformée en une valeur hypothétique de consommation de courant $h_{i,j} = \mathrm{HW}(v_{i,j})$. On obtient une nouvelle matrice $\mathscr{H}$ de taille $n \times K$:

$$\mathscr{V} = \begin{pmatrix} v_{1,1} & \dots & v_{1,K} \\ v_{2,1} & \dots & v_{2,K} \\ \vdots & \vdots & \vdots \\ v_{n,1} & \dots & v_{n,K} \end{pmatrix} \xrightarrow{\mathrm{HW}(v_{i,j})} \mathscr{H} = \begin{pmatrix} h_{1,1} & \dots & h_{1,K} \\ h_{2,1} & \dots & h_{2,K} \\ \vdots & \vdots & \vdots \\ h_{n,1} & \dots & h_{n,K} \end{pmatrix}$$

Étape 5 : Analyse statistique entre les valeurs hypothétiques de consommation de courant et les traces de courant mesurées. Étant données les matrices $\mathscr{H}$ et $\mathscr{T}$, la dernière étape d'une attaque statistique consiste à comparer chaque colonne de $\mathscr{H}$ avec toutes les colonnes de $\mathscr{T}$. Cela signifie que les valeurs hypothétiques de consommation de courant pour chaque hypothèse de clé sont comparées avec les traces de courant mesurées au préalable, à toutes les positions. Le résultat se présente sous forme d'une matrice $\mathscr{R}$ de taille $K \times w$ telle que :

$$\mathscr{R} = \begin{pmatrix} r_{1,1} & \dots & r_{1,w} \\ r_{2,1} & \dots & r_{2,w} \\ \vdots & \vdots & \vdots \\ r_{K,1} & \dots & r_{K,w} \end{pmatrix}$$

Les valeurs $r_{i,j}$ représentent le résultat de la comparaison entre les colonnes h_i et t_j. Elles permettent à l'attaquant de distinguer quelle colonne de $\mathscr{H}$ correspond à la bonne valeur de la clé secrète originale k. Il existe plusieurs moyens d'effectuer la comparaison. Kocher *et. al.* [47] introduisent une attaque statistique basée sur la différences des moyennes, appelée DPA. Cette

méthode a ensuite été généralisée par Brier *et. al.* [13] par l'attaque statistique basée sur le coefficient de Pearson, appelée CPA, que nous décrivons ci-dessous.

Attaque statistique basée sur le coefficient de Pearson. Le coefficient de Pearson est le moyen le plus connu en statistiques pour déterminer s'il existe une relation linéaire entre deux variables. Sa valeur se situe dans l'intervalle $[-1,1]$. Il est utilisé dans le cadre des attaques statistiques par observations pour mesurer la dépendance entre les colonnes h_i et t_j pour $i \in [1..K]$ et $j \in [1..w]$. Les coefficients de corrélation obtenus sont représentés dans la matrice $\mathscr{R}$ et définis par la relation suivante :

$$r_{i,j} = \frac{\sum_{x=1}^{n}(h_{x,i} - \bar{h_i}) \times (t_{x,j} - \bar{t_j})}{\sqrt{\sum_{x=1}^{n}(h_{x,i} - \bar{h_i})^2 \times \sum_{x=1}^{n}(t_{x,j} - \bar{t_j})^2}},$$

où $\bar{h_i}$ et $\bar{t_j}$ correspondent à la moyenne de la colonne h_i et celle de la colonne t_j, respectivement. En termes d'interprétation, la bonne hypothèse de clé correspondra à la valeur absolue de $r_{i,j}$ la plus élevée. En effet, plus la valeur de $r_{i,j}$ se rapproche de -1 ou 1, plus la relation linéaire entre l'hypothèse de clé et les traces de courant est importante.

L'attaque CPA est résistante en cas de bruit important dans les traces de courant. En effet, un attaquant peut essayer de compenser une hausse de bruit par une augmentation du nombre de traces. Dans la suite, nous présentons une attaque CPA sur l'AES.

2.3.7 Illustration d'une attaque CPA sur l'AES

Nous donnons dans cette section des détails sur l'attaque CPA de l'AES réalisée par Popp *et. al.* [68]. Les auteurs travaillent avec un microcontrôleur de type 8051 qui exécute une implémentation logicielle de l'AES. Les microcontrôleurs 8051 étaient jusqu'à récemment très utilisés dans les cartes à puce et sont donc des plateformes adéquates pour étudier les attaques statistiques par analyse de consommation de courant.

Concernant la mise en place de l'attaque CPA, Popp *et. al.* se servent d'une interface RS232 (ancêtre du port USB) afin de communiquer avec le microcontrôleur. Ce dernier reçoit 1000 messages clairs de l'ordinateur via son interface RS32, les chiffre et envoie les résultats à l'ordinateur. Les données étant codées sur 8 bits sur le microcontrôleur, les attaquants retrouveront la clé secrète octet par octet, c'est à dire en employant la méthode *"diviser pour mieux régner"*.

Dans une première étape, Popp *et. al.* ciblent ainsi le premier octet de sortie de la première boîte-S de l'AES du premier tour. Cette valeur intermédiaire est une fonction du premier octet du message clair et du premier octet de la clé secrète. En parallèle des 1000 chiffrements, les attaquants mesurent grâce à un oscilloscope les traces de courant correspondantes aux chiffrements des messages clairs différents, au premier tour de l'AES. Dans cet exemple, les traces sont mesurées au cours du temps, pendant 100 micro-secondes (notées µs), cela correspond à 10000 points. Cette deuxième étape conduit à la matrice $\mathscr{T}$ de taille 1000×10000. On note SBox la fonction ciblée par l'attaque, x_i le premier octet d'un message clair correspondant au $i^{ème}$ chiffrement, et k le premier octet de la clé secrète utilisée.

La troisième étape consiste à calculer toutes les valeurs intermédiaires possibles pour chaque hypothèse sur le premier octet de la clé. Ainsi, une nouvelle matrice $\mathscr{V}$ est créée où chaque coordonnée $v_{i,j}$ est calculée telle que :

$$v_{i,j} = \text{SB}ox(x_i \oplus k_j),$$

où les $(x_1, ..., x_{1000})$ correspondent au premier octet des 1000 messages à chiffrer et j appartient à l'intervalle $[0,255]$. En effet, il existe 256 valeurs possible pour un octet de clé secrète (*i.e.* toutes

les valeurs de GF(2^8)). La matrice $\mathscr{V}$ obtenue est donc de taille 1000×256.

La quatrième étape de l'attaque modélise les valeurs de $\mathscr{V}$ de manière à obtenir une nouvelle matrice $\mathscr{H}$ de valeurs hypothétiques de consommation de courant. Le modèle de fuite considéré par Popp *et. al.* est la valeur du bit de poids le plus faible (modèle LSb) de l'octet manipulé. On représente donc les valeurs de la matrice $\mathscr{H}$ par :

$$h_{i,j} = \mathrm{LS}b(v_{i,j})$$

Enfin, la dernière étape consiste à calculer le coefficient de Pearson entre les colonnes de $\mathscr{H}$ et de $\mathscr{T}$. Les résultats se présentent sous forme d'une matrice $\mathscr{R}$ de taille 256×100, contenant tous les coefficients de corrélation. Une manière de visualiser ces résultats est d'afficher chaque ligne de $\mathscr{R}$ dans un graphe, prenant en abscisse le temps et en ordonnée la valeur du coefficient. Dans ce cas, chaque graphe correspond à une hypothèse d'octet de clé. Par exemple, la Figure 22 présente les graphes correspondants aux hypothèses d'octets de clé égales à 223 (a), 224 (b), 225 (c) et 226 (d) en notations décimales. Nous pouvons observer des pics sur le graphe (c). Ils correspondent aux valeurs les plus hautes de la matrice $\mathscr{R}$ et indiquent donc une relation linéaire importante entre l'hypothèse d'octet de clé 225 et les traces de courant mesurées. Les premiers pics apparaissent au bout de $13,8\mu$s et donne ainsi aux attaquants des informations précieuses. En effet, non seulement ces derniers peuvent déduire que le premier octet de clé secrète est égal à 225, mais également que le microcontrôleur calcule la sortie de la première boîte-S de l'`AES` à la position $13,8\mu$s dans les traces.

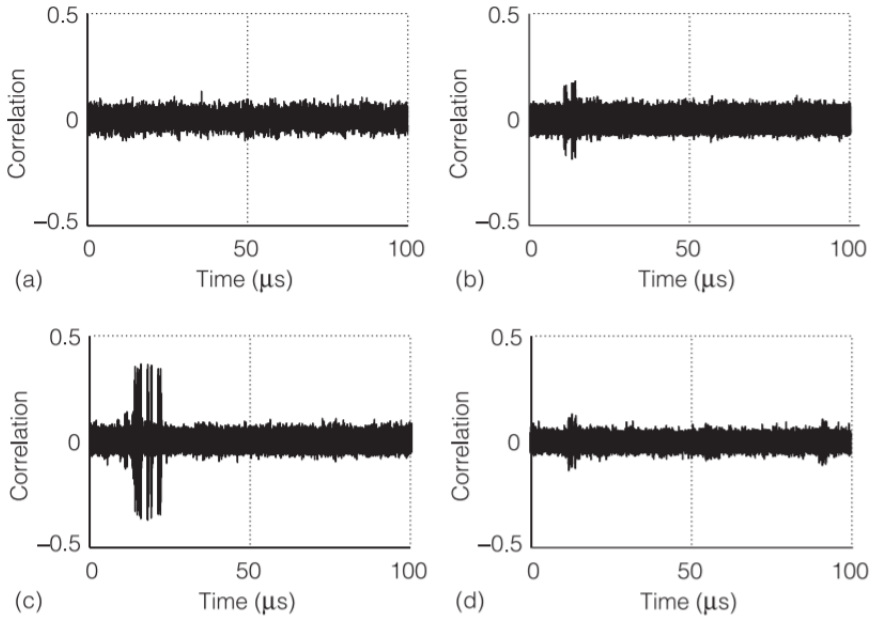


FIGURE 22 – Résultats d'une attaque CPA sur la sortie de la première boîte-S de l'`AES`- hypothèses d'octet de clé 223 (a), 224 (b), 225 (c) et 226 (d) [68]

De plus, il est possible d'observer plusieurs pics dans le graphe (c), indiquant aux attaquants que la valeur intermédiaire ciblée est impliquée dans plusieurs opérations au cours du temps. En remarque, il est également possible d'observer d'autres pics dans d'autres graphes, comme les traces (b) et (d). La moyenne de ces pics étant basse par rapport à celle des pics de (c), il n'est pas pertinent de les étudier. Pour éviter de confondre le pic lié à la bonne hypothèse d'octet de clé aux autres, il est important pour un attaquant d'estimer au mieux les coefficient de corrélation de la matrice $\mathscr{R}$. Plus le nombre de traces est important, meilleure est l'estimation. Ce phénomène dit de *ghost peaks* a été étudié dans l'article de Brier et al. [13] ainsi que celui de Clédière *et. al.* [14].

Pour conclure, nous avons vu dans cette section que les attaques par observations CPA exploitent les relations linéaires entre les émanations de courant d'un circuit CMOS et les données exécutées par l'algorithme cryptographique au sein du circuit. Dans la section suivante, nous étudions les différents moyens de défense contre ces attaques, visant à casser cette linéarité.

2.4 Les contre-mesures

Une attaque par observation peut retrouver la clé secrète car la consommation de courant de l'outil cryptographique dépend des valeurs intermédiaires manipulées lors de l'exécution de l'algorithme cryptographique. On appelle *contre-mesure* un moyen permettant de réduire ou supprimer cette dépendance. Les contre-mesures peuvent être catégorisées dans deux groupes : la *dissimulation* (*hiding* en anglais) et le *masquage* (*masking* en anglais). Introduite par Kocher *et. al.* [47], la dissimulation vise à modifier la consommation de courant du circuit de manière à réduire de manière significative la dépendance entre la consommation et les valeurs intermédiaires. Présentée par Kocher dans [46], la technique de masquage modifie directement les valeurs intermédiaires manipulées par l'outil cryptographique de manière à ce que la connaissance de la consommation de courant de ces valeurs modifiées n'apporte aucune information sur les données intermédiaires originales de l'algorithme. Les deux types de contre-mesures sont décrits ci-dessous.

2.4.1 La dissimulation

Afin de réduire la dépendance entre la consommation de courant d'un circuit et les valeur intermédiaires exécutées sur celui-ci, nous introduisons la notion du *rapport signal à bruit* (Signal-to-Noise Ratio (SNR) en anglais). Le SNR nous donne le rapport entre la valeur d'un signal contenant de l'information sensible et celle d'une information non significative (e.g. le bruit). Plus ce rapport est élevé, plus un attaquant aura de chance de réussir facilement son attaque. Ainsi, la contre-mesure de dissimulation peut jouer un rôle en diminuant le SNR. Mathématiquement, cela peut se faire de deux façons, soit en réduisant le signal à zéro, soit en augmentant vers l'infini la quantité de bruit. Cependant en pratique, les deux approches ne peuvent pas être implémentées de manière à atteindre la valeur idéale de zéro pour le SNR. Le signal peut être réduit fortement sans devenir nul et la quantité de bruit peut être plus importante sans tendre vers l'infini. Afin de satisfaire ces deux objectifs, nous présentons quelques méthodes ci-dessous.

Diminution du signal

Une manière de réduire le signal est d'égaliser la quantité d'énergie utilisée pour toutes les opérations de l'outil cryptographique. Si ce but est atteint, un attaquant ne sera plus capable de trouver des différences entre les traces mesurées (*i.e.* lors d'une attaque DPA par exemple). Le moyen le plus utilisé pour se rapprocher de cet objectif est d'utiliser des portes logiques CMOS telles que leur consommation de courant soit constante. Cette technique est appelée *logique double rail* (*dual-rail pre-charge (DRP) logic* en anglais) et est décrite dans [67]. Les travaux [63, 83, 54] présentent également d'autres méthodes pour réduire le signal dans un outil cryptographique.

Augmentation du bruit

Il existe plusieurs façon d'injecter du bruit dans les traces de courant mesurées [60, 41]. Il est tout d'abord possible d'exécuter un outil cryptographique supplémentaire en parallèle de celui que l'on souhaite protéger. Cet outil n'apporte rien de plus au résultat souhaité si ce n'est que de générer du bruit supplémentaire et rendre ainsi compliquée la tâche pour un attaquant d'accéder à l'information sensible. De plus, nous avons vu en Remarque 4 l'importance de l'alignement des traces pendant la mesure de la consommation de courant d'un circuit lors d'une attaque statistique par observations. Cela implique que la consommation de courant de chaque opération

doit être localisée au même endroit dans chaque trace. Sans cette condition, un attaquant devra prendre connaissance d'un plus grand nombre de traces [75]. Nous comprenons donc la motivation des fabricants de cartes à puce de désynchroniser les exécutions. Pour cela, du bruit peut être ajouté en rendant notamment le plus aléatoire possible le temps d'exécution des opérations d'un algorithme cryptographique. Les techniques les plus utilisées pour assurer cet objectif sont *l'ajout d'opérations factices* (*fake* ou *dummy operations* en anglais), *le mélange d'opérations* (*shuffling of operations* en anglais) et la *temporisation* (*jitter* en anglais).

Le principe de l'ajout d'opérations factices est d'insérer de manière aléatoire des opérations inutiles avant, pendant et/ou après l'exécution d'un algorithme cryptographique. Un entier aléatoire est ainsi généré à chaque exécution de l'algorithme afin de déterminer combien d'opérations factices sont ajoutées aux différents endroits. Il est important que le nombre total d'opérations ajoutées soit le même pour chaque exécution, sinon un attaquant peut avoir de l'information sur ce nombre en analysant le temps d'exécution de l'algorithme. Le changement aléatoire de position des opérations factices à chaque exécution de l'algorithme rend le travail d'un attaquant plus difficile. Cependant, en pratique, un ajout trop important d'opérations factices rend le temps d'exécution de l'algorithme trop important. Il est nécessaire de trouver un compromis entre le niveau de sécurité souhaité et le temps d'exécution de l'outil cryptographique. Dans [75], il est démontré que le nombre d'opérations factices à ajouter est linéaire en fonction du gain de sécurité attendu.

Le mélange d'opérations et la temporisation sont des contre-mesures alternatives à la précédente. La temporisation consiste à faire attendre l'algorithme cryptographique pendant un temps aléatoire avant chaque calcul sensible. Les traces sont alors désynchronisées. L'idée principale du mélange d'opérations est de changer l'ordre d'exécution des opérations d'un algorithme cryptographique. Cette technique est plus efficace que la méthode de temporisation puisque l'ordre est inconnu. Par exemple, dans le cas d'un `AES-128`, les 16 boîtes-S doivent être exécutées dans chaque tour. Comme ces substitutions sont indépendantes entre elles, il est possible de les exécuter dans un ordre arbitraire. Tout comme l'insertion d'opérations factices, cette technique permet de rendre plus indépendante la consommation de courant mesurée par un attaquant et les valeurs intermédiaires manipulées par l'outil attaqué. Le désavantage de cette contre-mesure vient du fait qu'il y a peu d'opérations cryptographiques sur lesquelles un mélange est applicable. De plus, certains algorithmes manipulent un nombre faible de boîtes-S (e.g. 4 boîtes-S pour le `SM4`).

Pour conclure, les techniques de diminution du SNR sont souvent combinées dans les implémentations. Nous présentons une deuxième contre-mesure dans la suite.

2.4.2 Le masquage

En comparaison à la technique de dissimulation, la contre-mesure de masquage est bien plus populaire dans la communauté cryptographique. Elle a été introduite par Chari *et. al.* [18] et Goubin et Patarin [39] en 1999, et permet d'éviter que, tout au long de l'algorithme, les valeurs intermédiaires dépendent directement des valeurs sensibles (e.g. sortie de la première boîte-S de l'`AES`). Afin que ces dernières ne soient accessibles par un attaquant, elles vont être cachées par des données aléatoires que l'on appelle des *masques*. Dans la suite, nous expliquons dans un premier temps comment la technique de masquage permet de sécuriser des implémentations contre les attaques par observations. Dans un second temps, nous appliquons la contre-mesure sur un algorithme cryptographique à chiffrement symétrique et nous observons les conséquences sur les performances de celui-ci.

Définition et résistance du masquage face aux attaques par observations

Le masquage consiste à diviser chaque valeur sensible v en d variables telle que les $(v_i)_{1\leq i\leq d}$ représentent les *masques* de v. Parmi ces masques, $(d-1)$ sont générés uniformément, aléatoirement et indépendant de v et le dernier est calculé tel que la combinaison de d masques, en fonction d'une opération $\star$, est égale à la variable initiale v. Nous obtenons :

$$v = v_1 \star ... \star v_d \tag{2.3}$$

La variable sensible v n'est plus manipulée seule, elle est remplacée par la combinaison de ses masques ci-dessus. L'opération $\star$ dépend du choix de masquage des opérations impliqué dans l'algorithme. En cryptographie, il existe une distinction entre le *masquage Booléen* et le *masquage arithmétique*. Le masquage Booléen s'applique lorsque la donnée intermédiaire est cachée à travers l'addition binaire, tel que $v = v_1 \oplus ... \oplus v_d$. Dans le masquage arithmétique, la valeur intermédiaire est cachée par une opération arithmétique, l'addition modulaire ou la multiplication modulaire, où le modulo est un entier défini par l'algorithme cryptographique. Pour simplifier les notations, dans la suite de cette thèse, on parle de *masquage additif* lorsque l'opération $\star$ correspond à l'addition binaire $\oplus$ et de *masquage multiplicatif* lorsque l'opération $\star$ correspond à la multiplication modulaire (pour un modulo donné), que nous notons $\otimes$.

Dans le contexte des attaques par observations, Chari *et. al.* [18] ont introduit la notion de *partage de secret*, où les d masques de v sont appelés des *morceaux*. On parle alors de *partage en d-morceaux* de v. Plus précisément, si l'opération $\star$ utilisée est $\oplus$ (respectivement $\otimes$), on dit que $(v_1, ..., v_d)$ est un *partage additif en d-morceaux* de v (respectivement *partage multiplicatif en d-morceaux* de v). Nous introduisons la notion d'*ordre* par la définition ci-dessous.

Définition 12. ***Attaque par observations à l'ordre*** t**.** *Soit t un entier naturel strictement positif. Une attaque par observations à l'ordre t est une attaque par observations qui s'applique sur une combinaison d'au plus t instants sur une trace de courant.*

Un partage en d-morceaux de v résiste à une attaque par observations à l'ordre $(d-1)$. En effet, si v est découpée en d morceaux, la connaissance de $(d-1)$ morceaux n'apporte aucune information sur v : comme ces morceaux sont statistiquement indépendants de v, leur consommation de courant est également indépendante de v. En conséquence, un attaquant sera incapable de retrouver la clé secrète en analysant les traces de courant de $(d-1)$ morceaux. Par ailleurs, plus l'ordre de l'attaque par observations est élevé, plus l'implémentation nécessite un nombre important de morceaux et moins il est facile de retrouver la clé secrète. Dans la suite, nous montrons comment le masquage s'applique en cryptographie symétrique.

Le masquage en cryptographie symétrique

Comme vu en Section 2.1.2, la conception d'un algorithme à chiffrement par blocs implique des opérations linéaires et non-linéaires définies sur un corps fini. Nous souhaitons ici masquer une donnée sensible v au sein d'un tel algorithme pour une résistance contre les attaques par observations à l'ordre 1. La stratégie classique est de masquer additivement v et de calculer la propagation des masques à travers les différentes transformations cryptographiques. On considère un partage additif en 2-morceaux (v_1, v_2) de v (*i.e.* $v_1 \oplus v_2 = v$) et f_L une fonction linéaire. On a l'équation suivante :

$$f_L(v) = f_L(v_1 \oplus v_2) = f_L(v_1) \oplus f_L(v_2). \tag{2.4}$$

Un attaquant est alors capable d'observer la mesure de consommation de $f_L(v_1)$ ou celle de $f_L(v_2)$, ce qui ne lui donne aucune information sur la valeur de $f_L(v)$ et garantit ainsi une protection contre les attaques par observations à l'ordre 1. Si le temps de calcul de la propagation des masques additifs v_1 et v_2 est rapide à travers des fonctions linéaire, il est cependant plus important à travers des opérations non-linéaires [36]. Un masquage multiplicatif de v est alors plus

approprié pour masquer des fonctions non-linéaires. Nous prenons l'exemple de la boîte-S de l'AES, basée sur le calcul de l'inverse multiplicatif d'un élément dans $GF(2^8)$: $INV_{AES}(v) = v^{254}$. On considère cette opération non-linéaire et un partage multiplicatif en 2-morceaux (v_1, v_2) de v (*i.e.* $v_1 \otimes v_2 = v$). On obtient alors :

$$\begin{aligned} INV_{AES}(v) = INV_{AES}(v_1 \otimes v_2) &= (v_1 \otimes v_2)^{254} \\ &= v_1^{254} \otimes v_2^{254} \\ &= INV_{AES}(v_1) \otimes INV_{AES}(v_2). \end{aligned} \tag{2.5}$$

Comme précédemment, un adversaire est alors capable d'observer la mesure de consommation de $INV_{AES}(v_1)$ ou celle de $INV_{AES}(v_2)$, ce qui ne lui donne aucune information sur la valeur de $INV_{AES}(v)$ et garantit ainsi une protection contre les attaques par observations à l'ordre 1.

Pour résumer, pour des questions d'optimisation de performances, un masquage additif est employé pour protéger le calcul d'une opération linéaire tandis qu'un masquage multiplicatif est préféré pour sécuriser le calcul d'une fonction non-linéaire. Akkar et Giraud [3] ont présenté en 2001 deux méthodes afin de masquer la fonction non-linéaire de l'AES :

— La première méthode est basée sur la conversion d'un masquage additif en un masquage multiplicatif (et inversement). Comme démontrée par Coron et Goubin [23], une telle conversion est problématique car elle génère un nombre supplémentaire important d'opérations, ce qui ralentit les performances de l'algorithme. De plus, un masquage multiplicatif est inefficace lorsque la valeur à masquer est nulle (voir explications sur le *problème du zéro* en Section 4.1). L'implémentation masquée de la conversion d'un masquage additif à un masquage multiplicatif (et inversement) est donc un sujet ouvert et très étudié par la communauté cryptographique [37, 84].

— La deuxième méthode est basée sur du masquage additif seulement. Afin de masquer additivement la table de substitution de l'AES, la solution proposée par les auteurs est de générer aléatoirement un masque m tel qu'un partage additif en 2-morceaux $(v \oplus m, m)$ de v soit créé. Pour toutes les valeurs possibles de m, la boîte-S est ainsi redéfinit telle que :

$$SBox(v \oplus m) = SBox(v) \oplus m \tag{2.6}$$

Cette technique de masquage fonctionne mais est coûteuse en surface. En effet, elle nécessite de stocker toutes les valeurs de $SBox(v) \oplus m$ pour tous les masques $m \in GF(2^8)$ possibles dans des registres.

De nombreuses implémentations masquées ont été proposées dans l'état de l'art pour l'algorithme de l'AES [3, 11, 18]. En comparaison, d'après les résultats de Mangard *et. al.* [55], le nombre de cycles d'horloges d'un AES masqué est deux fois plus important qu'un algorithme AES non-protégé sur la même plateforme (microcontrôleur de type 8051). En Section 2.2.3, nous avons vu que les performances d'une implémentation matérielle d'un circuit peuvent se mesurer en terme de surface (*i.e.* nombre de portes équivalentes NAND et nombre de masques aléatoires générés à stocker dans des registres). Il est également important de considérer le temps d'exécution (*i.e.* nombre de cycles d'horloges). En remarque, les performances d'une implémentation logicielle se mesurent en terme d'espace alloué à la mémoire (qui est limitée) et en temps d'exécution.

2.5 Conclusion

Ce chapitre met en valeur le rapport entre les mathématiques appliquées en cryptographie, l'électronique et la cryptanalyse. Nous avons décrit les mécanismes de chiffrement et de déchiffrement de l'algorithme symétrique AES, très étudié par la communauté cryptographique. De manière concrète, nous avons vu que les algorithmes à chiffrements par blocs sont exécutés sur des

circuits électroniques. Nos données privées, comme par exemple nos transactions bancaires sont protégées grâce au principe de confidentialité, assuré par notre carte bleue. L'expansion du marché des cartes à puce dans les années 1970 a ainsi favorisé l'évolution des technique de cryptanalyse, comme les attaques par observations dont nous détaillons le fonctionnement. Nous expliquons notamment d'où viennent les fuites de consommation de courant d'un circuit électronique, mesurées puis analysées par un attaquant. Nous avons décrit l'attaque CPA sur l'AES, basée sur le coefficient de Pearson, qui mesure les relations linéaires entre les fuites mesurées et les données cryptographiques contenant les données sensibles. Afin de casser la dépendance linéaire entre deux variables, nous avons présenté deux contre-mesures très étudiées dans l'état de l'art. D'une part la dissimulation permet d'altérer la consommation de courant du circuit. D'autre part le masquage modifie les données cryptographiques manipulées par le circuit électronique.

Nous avons vu que les algorithmes de chiffrement par blocs, et en particulier les fonctions non-linéaires utilisées (e.g. les boîtes-S), sont sensibles aux attaques par observations. Au niveau logiciel, les attaques par analyse de consommation de courant peuvent être contrées par du masquage sur les entrées. Cependant, comme démontré par Mangard *et. al.* dans [56], en présence de *glitches* dans les circuits électroniques, cela ne suffit pas à protéger une implémentation matérielle. Dans le chapitre suivant, nous décrivons le phénomène de glitches ainsi qu'une nouvelle contre-mesure, les *implémentations à seuil* [65], permettant de s'en protéger. Ce moyen de défense constitue un des axes principaux de recherche de ce manuscrit.

Chapitre 3

Introduction aux implémentations à seuil

Sommaire

Comme expliqué précédemment, les attaques par observations constituent une réelle menace pour les algorithmes cryptographiques embarqués. Afin de se prévenir d'elles, la communauté cryptographique développe au début du XXI^{e} siècle différentes contre-mesures. En particulier, un effort important est réalisé pour trouver des méthodes de protection pour des implémentations cryptographiques (e.g. `AES`) contre les attaques par analyse de consommation de courant (e.g. DPA, CPA). Introduit par Goubin et Patarin [39] et Chari *et. al.* [18], le *masquage* présente une contre-mesure particulièrement développée par les fabricants de cartes à puce. Comme expliqué en Section 2.4.2, cette protection vise à rendre la fuite de consommation observée statistiquement indépendante de la valeur sensible manipulée. Ainsi, de nombreux schémas de masquage [3, 11, 18] ont été proposés dans l'état de l'art. Cependant, en 2005, Mangard *et. al.* [56] démontrent la vulnérabilité de ces schémas face aux phénomènes de *glitches*. Dans [65], Nikova *et. al.* proposent en 2006 *les implémentations à seuil* comme une amélioration de la technique de masquage, prouvée résistante contre les attaques par observations en la présence de glitches.

Les techniques d'attaques ayant évoluées, il est devenu important de définir les capacités d'un adversaire dans ce qu'on appelle un *modèle d'attaque* ainsi que la notion de sécurité à atteindre vis-à-vis de ce modèle. On dit alors qu'une contre-mesure est résistante dans un modèle d'attaque particulier si une *preuve de sécurité* est donnée. Dans ce manuscrit, nous prouvons la sécurité de toutes nos propositions.

Dans un premier temps, la Section 3.1 introduit les modèles de sécurité impliqués dans ce manuscrit et présente le phénomène de glitches dans le contexte des attaques par observations. Puis, le principe des implémentations à seuil est présenté en Section 3.2. Nous étudions comment mettre en place cette contre-mesure afin d'assurer la protection de données sensibles dans un modèle de sécurité prenant en compte les glitches. La Section 3.3 se concentre ensuite sur l'une des propriétés des implémentations à seuil, *l'uniformité*, difficile à mettre en oeuvre sans alourdir

les performances. Nous nous intéressons ainsi à l'état de l'art et aux défis qui en ressortent. Enfin, la Section 3.4 décrit la méthode originale présentée par Nikova *et. al.* dans [65] afin de construire une implémentation à seuil d'une (n,m)-fonction (voir Définition 3).

3.1 Introduction aux glitches

3.1.1 Modèles d'attaque, présentation des glitches et conséquences sur la sécurité

Comme vu en Section 2.2.1, un algorithme cryptographique peut être implémenté par un circuit électronique, constitué de plusieurs registres entre lesquels des portes logiques CMOS combinatoires interagissent pour réaliser une (n,m)-fonction spécifique. D'un point de vue mathématique, ces portes logiques correspondent à des opérations définies sur des corps finis de caractéristique 2 (*i.e.* $\mathrm{GF}(2^n)$). On retrouve par exemple le Xor notée $\oplus$, la multiplication modulaire dans le corps notée $\otimes$, et l'inverse multiplicatif notée $\otimes^{-1}$ [1]. Ces opérations appartiennent à un ensemble mathématique que nous notons $\mathcal{O}$ dans la suite. Nous donnons ci-dessous une abstraction mathématique de la notion de circuit électronique donnée en Définition 6.

Définition 13. ***Circuit électronique en mathématiques.*** *[78] Soit* F *une* (n,m)*-fonction et soit* $\mathcal{O}$ *un ensemble d'opérations élémentaires. Un circuit électronique (idéal)* $\mathrm{C_F}$ *implémentant* F *grâce aux opérations de* $\mathcal{O}$ *est un graphe orienté où chaque noeud définit un élément de* $\mathcal{O}$ *et chaque arête représente une valeur intermédiaire du calcul.*

Dans le contexte des attaques par observations, un attaquant est capable d'exploiter une information physique (e.g. une consommation de courant électrique) qui fuite lors de l'exécution de calculs sur un circuit électronique. Comme défini en Section 2.3.3, du *bruit* peut être présent lors de la mesure de cette information, alors modélisée par un modèle de fuite (voir Définition 11) sur la valeur intermédiaire. Plus particulièrement, les capacités d'un attaquant sont souvent représentées par des *modèles d'attaques*. Parmi eux, le *modèle d'attaque par sondage* (*probing adversary model* en anglais) [42] est l'un des plus favorables à l'adversaire puisqu'il suppose que ce dernier a accès aux valeurs exactes d'un nombre fini (e.g. t) de valeurs intermédiaires. Nous donnons la définition formelle de ce modèle ci-dessous.

Définition 14. ***Modèle d'attaque par sondage à l'ordre*** t**.** *[78] Soient* I *un ensemble d'indices et* $\mathrm{C_F}$ *un circuit avec* $(\mathrm{V}_i)_{i\in \mathrm{I}}$ *valeurs intermédiaires (les arêtes). Soit* t *un entier strictement positif, un modèle d'attaque par sondage à l'ordre* t *correspond à une situation où un adversaire peut choisir n'importe quel sous-ensemble* J *de* I *constitué de* t *éléments et est capable d'observer les valeurs intermédiaires* $(\mathrm{V}_j)_{j\in \mathrm{J}}$ *pour toutes les entrées possibles du circuit* $\mathrm{C_F}$*.*

La pertinence de ce modèle pour mesurer la résistance d'un circuit contre les attaques par observations a été argumentée dans de nombreux articles dont [42], [69] et [31]. Une attaque réalisée par un adversaire dans le modèle d'attaque par sondage à l'ordre t conduit à la notion d'un circuit *sécurisé vis-à-vis du sondage à l'ordre* t. Nous donnons sa définition formelle ci-dessous.

Définition 15. ***Sécurité vis-à-vis du sondage à l'ordre*** t**.** *[17] Soit* t *un entier strictement positif. Un circuit est dit sécurisé vis-à-vis du sondage à l'ordre* t *si aucun adversaire dans le modèle d'attaque par sondage à l'ordre* t *contre ce circuit n'est capable d'extraire de l'information sensible.*

Par exemple, afin de mettre en défaut une implémentation sécurisée vis-à-vis du sondage à l'ordre 1, un attaquant doit combiner au moins 2 mesures lors d'une attaque par observations.

Un algorithme assurant une sécurité vis-à-vis du sondage à l'ordre t est résistant à la classe d'attaques par observations à l'ordre t (voir Définition 12). Dans l'état de l'art, différentes implémentations de l'`AES` basées sur des schémas de masquage ont été formellement prouvées comme étant sécurisées dans le modèle d'attaque par sondage à l'ordre 1 [42, 73, 74]. Cependant, en 2005, Mangard *et. al.* [56] montrent les vulnérabilités de ce modèle lorsque le phénomène de *glitches* est présent dans le circuit. Afin de mieux appréhender cette nouvelle notion dans cette thèse, il est nécessaire de faire la distinction entre les deux définitions d'un glitch données en électronique.

1. L'inverse multiplicatif est impliquée dans la partie non-linéaire de certains algorithmes cryptographiques (e.g. `AES`).

Un glitch peut être vu comme une défaillance électrique, provoquant une pointe de tension au sein d'un circuit électronique. Il peut aussi être considéré comme une erreur de programmation, un bug dans un jeu vidéo. Dans ce cas, il peut par exemple être exploité afin de tricher, comme le fait de pouvoir traverser les murs pour aller plus vite (voir Figure 23), ou encore pour créer un nouveau style de musique, basé sur les défauts sonores d'un dispositif électronique (e.g. l'œuvre de Carsten Nicolai qui est un artiste allemand de musique électronique). Nous ne considérons pas cette définition de glitch dans la suite.

FIGURE 23 – Exemple d'un glitch dans un jeu vidéo

Dans ce manuscrit, un glitch est vu comme un évènement inattendu. Comme dit précédemment, dans un circuit électronique, les éléments de $\mathcal{O}$ sont des portes logiques CMOS combinatoires (ou des combinaisons de telles portes). Entre ces portes, durant un même cycle, des valeurs intermédiaires transitent. Chacune d'elles correspond à un signal électrique. Le circuit présenté en Définition 13 est dit "idéal" car il considère que tous les signaux électriques prennent le même temps à transiter d'une porte logique à une autre. Cependant, ce circuit peut être vu de manière plus réaliste. En effet, comme expliqué dans la Remarque 3, il existe des états instables au sein du circuit qui peuvent conduire à des délais différents de propagation des signaux d'une porte logique à une autre. Ainsi, il est possible que plusieurs signaux arrivent à des temps différents à l'entrée d'une porte, qui retournera alors différentes sorties, autre que la sortie finale attendue. Ces retards peuvent permettre d'avoir accès à des valeurs intermédiaires qui peuvent fournir beaucoup d'information sur les données sensibles. Nous donnons ci-dessous la définition formelle des glitches utilisés dans cette thèse.

Définition 16. ***Glitches.*** *Dans un circuit, les transitions à la sortie d'une porte logique qui apparaissent avant que cette dernière atteigne un état stable et retourne la valeur correcte sont appelées des glitches.*

En d'autres termes, même si les entrées du circuit restent inchangées, la valeur qui transite entre deux portes peut changer durant un cycle, ce qui correspond à une période d'instabilité du circuit (voir Remarque 3). On parle d'*état interne à un temps* τ, noté $C_F(\tau)$. En conséquence, il est possible que lorsqu'un attaquant observe la valeur intermédiaire V_i, il obtienne une valeur non-attendue à un instant précis τ. On note cette valeur $V_i(\tau)$ dans la suite. Les capacités de ce nouvel adversaire sont représentées par un nouveau modèle d'attaque, défini ci-dessous.

Définition 17. ***Modèle d'attaque par sondage à l'ordre t en présence de glitches.*** *[78] Soient C_F un circuit et t un entier strictement positif. Un modèle d'attaque par sondage à l'ordre t en présence de glitches correspond à une situation où un adversaire peut choisir t instants $\tau_1, \tau_2, ..., \tau_t$ et peut observer les états internes $(C_F(\tau_i))_{i \leq t}$.*

Une attaque réalisée par un adversaire dans le modèle d'attaque à l'ordre t en présence de glitches conduit à la notion d'un circuit *sécurisé à l'ordre t en présence de glitches*. Nous donnons sa définition formelle ci-dessous.

Définition 18. ***Sécurité à l'ordre*** t ***en présence de glitches.*** *Soit t un entier strictement positif. Un circuit est dit sécurisé à l'ordre t en présence de glitches si aucun adversaire dans le modèle d'attaque par sondage à l'ordre t en présence de glitches contre ce circuit n'est capable d'extraire de l'information sensible.*

Remarque 5. *Un algorithme cryptographique sécurisé à l'ordre t en présence de glitches est aussi sécurisé vis-à-vis du sondage à l'ordre t. La réciproque est fausse (voir exemple 1 en Section 3.1.2).*

Dans la section suivante, nous illustrons la différence entre un circuit idéal et un circuit en présence de glitches. Les modèles d'attaques correspondants sont également illustrés.

3.1.2 Mise en situation des glitches

Dans un premier exemple, nous montrons qu'un circuit sécurisé vis-à-vis du sondage à l'ordre 1 peut être non sécurisé à l'ordre 1 en présence de glitches.

Exemple 1 : Soient x, β et β' trois éléments définis dans GF(2). Considérons un schéma de masquage où la valeur sensible x est successivement additivement masquée par les variables β puis β'. La Figure 24 représente un circuit idéal C_F implémentant la fonction $F : (x \oplus \beta, \beta, \beta') \mapsto x \oplus \beta'$. Le circuit est ainsi composé de deux portes logiques Xor (*i.e.* les noeuds) et trois valeurs intermédiaires $(V_i)_{i=\{1,2,3\}}$ (*i.e.* les arêtes). Dans le cadre d'une attaque par observation, nous nous concentrons sur les renseignements que possède un adversaire sur x en mesurant l'information physique qui est sondée sur la valeur intermédiaire V_3 durant un cycle.

Dans le modèle attaque par sondage à l'ordre 1 (Définition 14), l'adversaire mesure l'information $V_3 = x \oplus \beta'$. Dans ce cas, comme la valeur sensible x est masquée additivement par β', la consommation de courant électrique observée lors de la manipulation de V_3 ne pourra pas donner de renseignement sur x. Ainsi, dans ce schéma de masquage, le circuit est sécurisé vis-à-vis du sondage à l'ordre 1 (Définition 15).

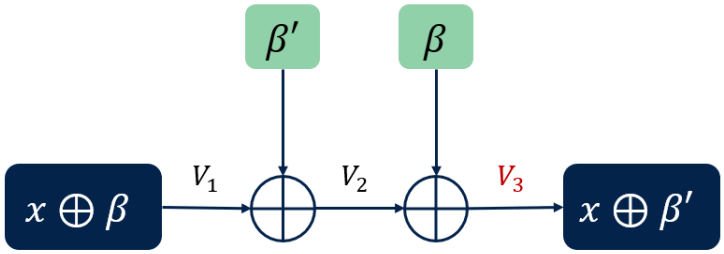


FIGURE 24 – Circuit idéal C_F implémentant la fonction $F : (x \oplus \beta, \beta, \beta') \mapsto x \oplus \beta'$

Considérons maintenant la présence de glitches au sein du circuit C_F. Il peut exister alors un temps τ tel que le signal β atteint la seconde porte Xor alors que le signal β' n'est pas encore arrivé à la première porte Xor. Le circuit C_F est maintenant illustré en Figure 25.

Dans cette nouvelle situation, la première porte Xor va prendre en entrée la valeur sensible masquée $x \oplus \beta$ et, par exemple, une valeur nulle à la place de β', contenue dans le registre précédent le registre où est stocké $x \oplus \beta$. Ainsi, la seconde porte Xor va traiter les variables $x \oplus \beta$ et β, conduisant à une situation où un adversaire dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches peut mesurer l'information $V_3(\tau) = x$ à l'instant τ. La nouvelle consommation de courant observée sera alors directement dépendante de la donnée sensible x, cassant ainsi la sécurité du schéma de masquage.

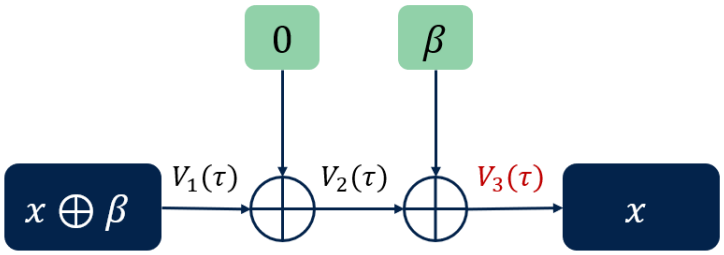


FIGURE 25 – Circuit C_F implémentant la fonction $F : (x \oplus \beta, \beta, \beta') \mapsto x \oplus \beta'$ en présence de glitches

Le circuit C_F est constitué de deux portes logiques. Dans un circuit composé d'un plus grand nombre d'opérateurs logiques, il est important de noter que les glitches s'y propagent en avalanche. En effet, un seul glitch présent à la sortie d'une porte logique conduira à l'observation de plusieurs glitches aux sorties de toutes les portes connectées à cette porte. Ce phénomène d'avalanche est dangereux pour la sécurité d'une implémentation car les glitches peuvent être dépendants des données manipulées dans le circuit électronique. Nous illustrons ce scénario dans un deuxième exemple.

Exemple 2 : Soient x, y, z, β et β' cinq éléments définis dans GF(2). On considère le circuit présenté en Figure 26, prenant en entrée les valeurs $(x \oplus \beta), (y \oplus \beta'), x, y$ et z et retournant la multiplication de x par y additivement masquée par z (*i.e.* $(x \otimes y) \oplus z$).

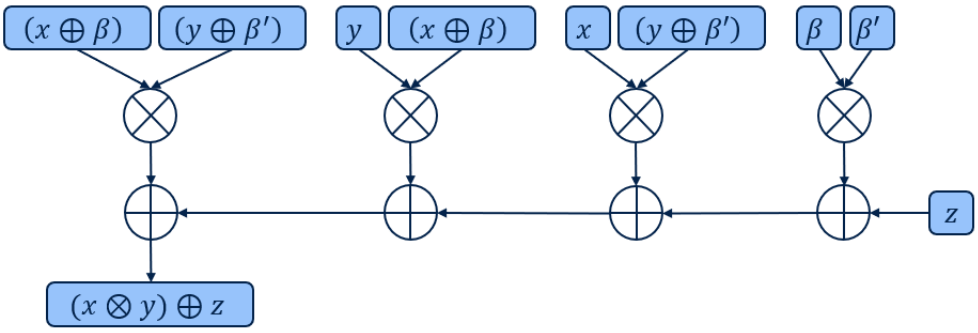


FIGURE 26 – Circuit implémentant la multiplication masquée $(x \otimes y) \oplus z$ [65]

Supposons qu'à un instant précis, un glitch apparaisse à l'entrée $(x \oplus \beta)$. La propagation de ce glitch dépend alors des valeurs de y et $(y \oplus \beta')$. Plus encore, la consommation de courant mesurée sur le circuit est fortement corrélée au nombre de portes logiques impactées par ce glitch. Dans le Tableau 1, en fonction des valeurs de y et $(y \oplus \beta')$, il est indiqué le nombre de portes Xor (*i.e.* $\oplus$) et de portes AND (*i.e.* $\otimes$) impactées par un glitch sur l'entrée $(x \oplus \beta)$. Des résultats similaires peuvent être obtenus en analysant l'effet d'un glitch sur n'importe quelle autre entrée du circuit [56].

y	$(y \oplus \beta')$	AND	Xor
0	0	0	0
0	1	1	1
1	0	1	2
1	1	2	2

TABLEAU 1 – Nombre de portes logiques AND et Xor impactées par un glitch sur l'entrée $(x \oplus \beta)$ du circuit en Figure 26 [65]

La raison pour laquelle les constructions utilisant la contre-mesure de masquage peuvent être

attaquées est qu'elles manipulent la valeur sensible masquée et le masque dans la même logique combinatoire. Par exemple, le circuit présenté en Figure 24 combine la valeur masquée $x \oplus \beta$ et le masque β dans le même cycle. En conséquence, la consommation de courant mesurée par un attaquant dépend directement de la valeur masquée et du masque, et correspond donc à une fonction de la valeur non-masquée (*i.e.* $F(x \oplus \beta, \beta) = x \oplus \beta \oplus \beta = x$). Dans la section suivante, nous présentons la contre-mesure des implémentations à seuil permettant, entre autre, d'éviter de manipuler la valeur masquée et le masque dans une même fonction. Nous obtiendrons une implémentation cryptographique sécurisée à l'ordre $t > 0$ en présence de glitches.

3.2 Les implémentations à seuil

3.2.1 Principes fondamentaux

Appliquées aux (n, m)-fonctions, les implémentations à seuil ont été introduites par Nikova *et. al.* [65] pour contrer les attaques par observations en présence de glitches. Elles consistent à la fois à l'application du *partage de secret* [77, 39] et à une adaptation du *calcul multipartite sécurisé* [7] sur les données manipulées lors du calcul. Nous donnons les principes de ces deux concepts ci-dessous.

Le partage de secret est abordé en Section 2.4.2. Il consiste au découpage en plusieurs morceaux de ce secret. Plus formellement, si x correspond à l'entrée d'une (n, m)-fonction F et X sa sortie, ces données sont séparées en d morceaux x_i et d' morceaux f_i et X_i, respectivement. La Figure 27 illustre la situation. Nous pouvons noter qu'il existe plusieurs manières de recombiner les morceaux, suivant une (ou plusieurs) opération sur un corps fini. Initialement, Nikova *et. al.* [65] ont défini un découpage additif sur GF(2^8) (voir Section 3.2.2), c'est-à-dire que la donnée sensible résulte de l'addition de tous ses morceaux.

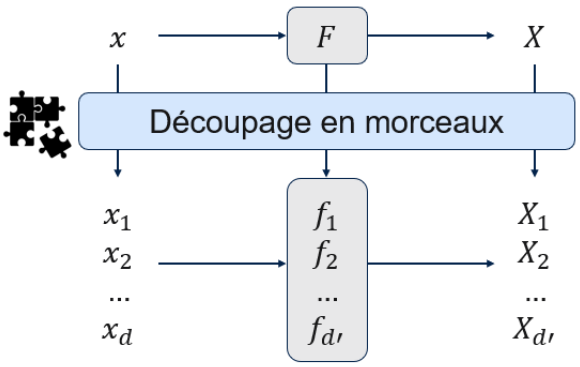


FIGURE 27 – Partage de secret appliqué à une (n, m)-fonction $F : x \mapsto X$

Le calcul multipartite sécurisé permet à chaque participant d'un réseau (ici un f_i) de calculer et connaître le résultat d'une fonction F en fonction de l'entrée x (ici X) sans révéler sa propre entrée (ici x_i) aux autres participants du réseau. Un participant n'a donc accès qu'à un seul morceau de x (*i.e.* le sien). La Figure 28 illustre le calcul multipartite sécurisé classique pour 3 participants.

Le calcul multipartite sécurisé des implémentations à seuil diffère de la méthode classique car les participants (ici les f_i) manipulent plusieurs entrées (ici plusieurs x_i), de telle sorte que chacun des participants n'ait pas la connaissance de toutes les entrées réunies (et donc de x). La Figure 29 présente cette situation pour 3 participants.

Pour conclure, une implémentation à seuil d'une (n, m)-fonction $F : x \mapsto X$ se base sur le partage de secret et le calcul multipartite sécurisé afin de construire un système de fonctions tel que chacune

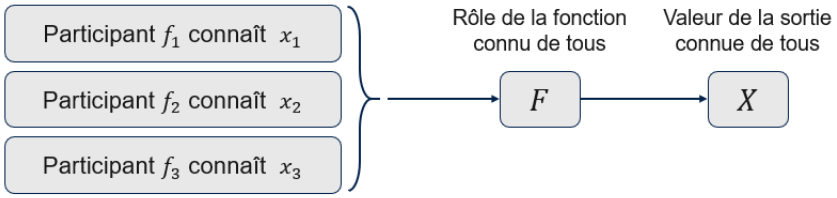


FIGURE 28 – Calcul Multipartite Sécurisé Classique

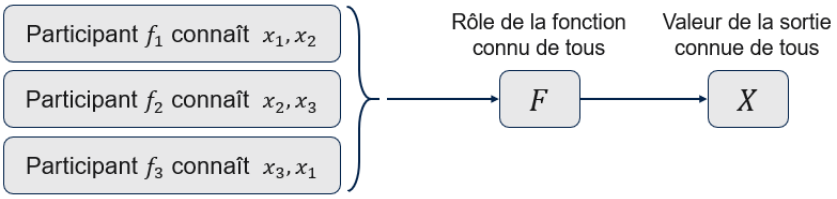


FIGURE 29 – Calcul Multipartite Sécurisé dans le contexte des Implémentations à Seuil

d'elles prend en entrée un ou plusieurs morceaux de x sans avoir la connaissance sur x, et retourne un morceau de X. La section suivante définit formellement les différentes notions utilisées lors d'une telle construction.

3.2.2 Notations et notions

Soient n, d deux entiers strictement positifs et m un entier. Dans cette section, nous discutons du découpage des éléments dans le corps fini GF(2^n) ainsi que celui des (n, m)-fonctions.

Découpage dans GF(2^n)**.** La technique du partage de secret [18] appliquée sur une donnée sensible $x \in \text{GF}(2^n)$ consiste au découpage de celle-ci en d morceaux aléatoires x_i, en fonction d'une opération mathématique précise définie dans GF(2^n). Nous appelons le tuple $\mathbf{x} = (x_1, \ldots, x_d) \in \text{GF}(2^n)^d$ le *partage en d-morceaux* de x. Nous introduisons également la *fonction de reconstruction de l'entrée* qui permet de retrouver la valeur de l'entrée x depuis son partage $\mathbf{x}$ en d-morceaux. Cette fonction est notée rec et définit de manière unique un partage de x tel que $rec(\mathbf{x}) = x$. Ci-dessous, nous présentons différents types de partage de x.

Le découpage le plus répandu et utilisé par Nikova *et. al.* dans [65] est le *partage additif en d-morceaux*, défini par l'addition binaire $\oplus$ dans GF(2^n) et par la fonction de reconstruction de l'entrée suivante :

$$rec_{add} : \mathbf{x} \in \text{GF}(2^n)^d \mapsto x = \bigoplus_{i=1}^{d} x_i \in \text{GF}(2^n)$$

Il existe également d'autres découpages. Par exemple, le *partage multiplicatif en d-morceaux*, défini par la multiplication $\otimes$ telle que :

$$rec_{mult} : \mathbf{x} \in (\text{GF}(2^n)^\star)^d \mapsto x = \bigotimes_{i=1}^{d} x_i \in \text{GF}(2^n)^\star$$

Une combinaison de plusieurs opérations mathématiques différentes dans GF(2^n) peut également conduire à un nouveau type de partage. Dans le chapitre suivant, nous définissons un découpage basé sur une multiplication et une addition binaire dans GF(2^n), que nous appelons *partage affine*. Dans la suite, nous étudions le partage en d'-morceaux d'une (n, m)-fonction.

Découpage des (n,m)-fonctions. Le partage de secret peut aussi s'appliquer sur une fonction vectorielle que l'on découpe en d' sous-fonctions. L'ensemble de ces dernières est appelé une *réalisation d'-masquée* d'une (n,m)-fonction F. On la note $\mathbf{F} : \mathrm{GF}(2^n)^{d'} \mapsto \mathrm{GF}(2^m)^{d'}$ telle que :

$$\mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_{d'}(\mathbf{x})), \tag{3.1}$$

où les f_i sont des sous-fonctions telles qu'il existe *une fonction de reconstruction de sortie* rec' définie par :

$$\mathrm{F}(x) = rec'(f_1(\mathbf{x}), ..., f_{d'}(\mathbf{x})).$$

Dans une implémentation à seuil d'une (n,m)-fonction F, la fonction de reconstruction de l'entrée et celle de la sortie ont souvent la même définition. En d'autres termes, la manière de combiner les morceaux x_i pour retrouver l'entrée x est souvent la même que celle pour combiner les sous-fonctions f_i pour retrouver la sortie $\mathrm{F}(x)$.

D'après l'Équation 3.1, les éléments d'une réalisation d'-masquée prennent en paramètre des morceaux de la donnée sensible. La section suivante introduit trois propriétés permettant de les construire.

3.2.3 Propriétés

Originellement, Nikova *et. al.* [65] ont expliqué comment construire une implémentation à seuil d'une (n,m)-fonction, sécurisée à l'ordre 1 en présence de glitches. Dans cette section, nous définissons les trois propriétés permettant d'obtenir une telle construction à l'ordre $t > 0$. Chacune des propriétés joue un rôle important dans la structure d'une implémentation à seuil :

- La *correction* permet d'assurer la **fonctionnalité** du schéma de telle sorte qu'il retourne la valeur attendue.
- La propriété *glitches-immune* permet d'assurer une **sécurité** à l'ordre t en présence de glitches.
- L'*uniformité* permet d'assurer un bon enchaînement entre chaque implémentation à seuil d'une (n,m)-fonction au sein d'un circuit. Cette propriété est vue comme un critère de **composition**.

Pour une entrée x, la propriété de *correction* assure que la réalisation d'-masquée $\mathbf{F}(\mathbf{x})$ de la (n,m)-fonction F correspond bien au partage de la sortie $\mathrm{F}(x)$ attendue. Plus formellement, on a :

Définition 19. ***(Correction.)*** *Soient d et d' deux entiers strictement positifs, $\mathbf{F}$ une réalisation d'-masquée de la (n,m)-function* F *et rec la fonction de reconstruction de l'entrée x de la fonction* F*. On dit que $\mathbf{F}$ est correcte si et seulement si, pour tout $x \in \mathrm{GF}(2^n)$, il existe une fonction de reconstruction de sortie rec' tel que $x = rec(\mathbf{x})$ implique $\mathrm{F}(x) = rec'(\mathbf{F}(\mathbf{x}))$, où $\mathbf{x}$ est le partage en d-morceaux de x associé à rec.*

En mathématiques, la définition classique de l'indépendance est donnée de manière *statistique* lorsque deux variables aléatoires n'ont aucune influence l'une sur l'autre. Prenons par exemple trois variables x, y et z telles que $z = x \oplus y$ où x et y sont statistiquement indépendantes. On a alors que z est statistiquement indépendante de y. Dans la deuxième propriété des implémentations à seuil, on utilise l'indépendance *fonctionnelle*. On dit qu'une variable est fonctionnellement indépendante d'une autre si sa valeur ne change pas quand celle de l'autre variable est modifiée. Dans notre exemple, z n'est pas fonctionnellement indépendante de y, car si y varie, la valeur de z est modifiée. Dans la suite, si nous ne précisons pas, l'indépendance est toujours vue de manière statistique. Nous définissons la deuxième propriété des implémentations à seuil ci-dessous.

Définition 20. ***(Glitches-immunité à l'ordre t.)*** *Soit un partage d'une entrée x d'une (n,m)-fonction. Une réalisation de cette fonction est glitches-immune à l'ordre t si n'importe quelle combinaison des entrées de t sous-fonctions de cette réalisation est fonctionnellement indépendante d'au moins un morceau du partage de x.*

Cette propriété assure une sécurité à l'ordre t contre les attaques par observations en présence de glitches. Elle impose de construire chaque sous-fonction de manière à ce qu'un attaquant ne puisse pas reconstruire la valeur sensible en observant les entrées de t sous-fonctions. Par exemple, à l'ordre 1, l'observation des entrées d'une sous-fonction ne permettra pas de retrouver la valeur de la donnée secrète, car cette sous-fonction est fonctionnellement indépendante d'au moins un morceau du partage de cette donnée. Ainsi, étant donné que dans une même fonction, toutes les informations permettant de reconstruire la donnée secrète ne sont donc pas accessibles, la présence de glitches ne représente pas de danger.

Afin d'appréhender la troisième propriété, nous devons prendre une vue plus générale de la situation. En effet, la propriété d'uniformité n'est ni nécessaire ni suffisante pour assurer la sécurité à l'ordre t en présence de glitches [61]. Elle joue un rôle important quand la sortie d'une réalisation d'-masquée est aussi l'entrée d'une autre réalisation d''-masquée (pour d' et d'' des entiers strictement supérieurs). On suppose une construction où un partage en d-morceaux $\mathbf{x}$ est l'entrée d'une réalisation d'-masquée $\mathbf{F} = (f_1, ..., f_{d'})$, dont la sortie correspond elle-même à l'entrée d'une autre réalisation d''-masquée $\mathbf{H} = (h_1, ..., h_{d''})$. La Figure 30 illustre la vue d'ensemble. La propriété d'uniformité d'une implémentation à seuil impose que si l'entrée $\mathbf{x}$ de $\mathbf{F}$ est uniforme, alors la sortie de $\mathbf{F}$ doit aussi être uniforme. De la même façon, comme l'entrée $\mathbf{F}$ de $\mathbf{H}$ est uniforme, alors la sortie de $\mathbf{H}$ doit aussi être uniforme. Cela implique qu'aucun biais (voir Définition 5) ne soit introduit dans la composition de réalisations.

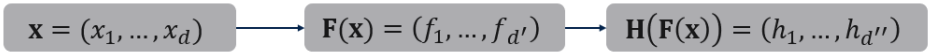


FIGURE 30 – Composition de deux réalisations d'-masquée $\mathbf{F}$ et d''-masquée $\mathbf{H}$

Plus formellement, la propriété doit assurer que toutes les réalisations d-masquées d'une implémentation soient uniformes. Dans [15] Carlet donne la définition suivante.

Définition 21. ***(Uniformité.)[15]*** *Soient d' un entier strictement positif et* F *une (n,m)-fonction. Une réalisation d'-masquée* $\mathbf{F}$ *de* F *avec une fonction de reconstruction de sortie rec' est uniforme si pour tout $\mathbf{z} \in \mathrm{GF}(2)^{d \times m}$, le nombre de $\mathbf{x} \in \mathrm{GF}(2)^{d \times n}$ pour lequel $\mathbf{F}(\mathbf{x}) = \mathbf{z}$ est égal à $2^{(d-1)\times(n-m)}$ fois le nombre de $x \in \mathrm{GF}(2)^n$ pour lequel $\mathrm{F}(x) = rec'(\mathbf{z})$.*

Dans la suite, nous illustrons les propriétés de correction, de glitches-immunité et d'uniformité présentées ci-dessus à travers deux exemples.

3.2.4 Illustration des propriétés

Nous supposons dans cette partie qu'Alice souhaite calculer le résultat de la multiplication de deux bits x et y sur un composant électronique. Elle sait qu'Ève est capable de faire des attaques par observations à l'ordre 1 en présence de glitches. Afin de se protéger d'Ève, Alice peut donc construire une réalisation masquée de la fonction $\mathrm{F} : (x, y) \in (\mathrm{GF}(2))^2 \mapsto x \otimes y \in \mathrm{GF}(2)$ satisfaisant les propriétés de correction, de glitches-immunité à l'ordre 1 et d'uniformité.

Dans [65], Nikova *et. al.* ont défini dans le théorème suivant le nombre minimal de morceaux d des partages de x et y nécessaires afin de créer une réalisation correcte et glitches-immune à l'ordre 1.

Théorème 1. *[65] Le nombre minimum de morceaux requis pour implémenter une réalisation d'un produit de v variables, satisfaisant la correction et la glitches-immunité à l'ordre* 1*, est donné par $d \geq v + 1$.*

Nous présentons ci-dessous deux exemples permettant de construire une implémentation à seuil de la fonction F.

Exemple 1 : à partir de la proposition d'Alice

Le Théorème 1 conduit Alice à découper les entrées x et y en au moins 3 morceaux. Elle choisit ainsi des partages additifs en 3-morceaux (x_1, x_2, x_3) de x et (y_1, y_2, y_3) de y en entrée et construit une réalisation 3-masquée $\mathbf{F} = (f_1, f_2, f_3)$ de F telle que :

$$f_1(x_1, y_1) = x_1 \otimes y_1 \tag{3.2}$$
$$f_2(x_1, x_2, y_1, y_2) = x_2 \otimes y_2 \oplus x_1 \otimes y_2 \oplus x_2 \otimes y_1$$
$$f_3(x_1, x_2, x_3, y_1, y_2, y_3) = x_3 \otimes y_3 \oplus x_1 \otimes y_3 \oplus x_3 \otimes y_1 \oplus x_2 \otimes y_3 \oplus x_3 \otimes y_2 \tag{3.3}$$

Correction. Nous vérifions tout d'abord la propriété de correction de $\mathbf{F}$. En additionnant f_1, f_2 et f_3 dans GF(2), on retrouve le produit de x par y. Il existe donc une fonction de reconstruction de sortie définie par $rec'(\mathbf{F}(\mathbf{x})) = \bigoplus_{i=1}^{3} f_i = \mathrm{F}(x) = x \otimes y$. D'après la Définition 19, la réalisation 3-masquée $\mathbf{F}$ de F est correcte.

Indépendance. Nous vérifions ensuite la propriété d'indépendance de $\mathbf{F}$. Cette dernière permet de s'assurer qu'une tierce personne ne soit pas capable d'observer de l'information sur les entrées x ou y en observant un élément de $\mathbf{F}$. D'après la Définition 20, pour $i = \{1, 2, 3\}$, nous devons vérifier que chaque sous-fonction f_i est fonctionnellement indépendante d'au moins un morceau de x et un morceau de y. Il est facile d'observer que l'Équation 3.3 ne satisfait pas cette condition, comme tous les morceaux de x et de y sont utilisés dans f_3. La réalisation $\mathbf{F}$ de F proposée par Alice n'est donc pas glitches-immune à l'ordre 1.

Une solution possible pour corriger l'erreur d'Alice est de déplacer les termes contenant x_1 et y_1 de l'équation 3.3 dans l'équation 3.2. Comme nous ne modifions pas le nombre total de termes de $\mathbf{F}$, cette modification n'altère pas son caractère de correction. Ainsi, nous obtenons une nouvelle réalisation 3-masquée $\mathbf{F}' = (f_1', f_2', f_3')$ définie par :

$$\begin{aligned} f_1'(x_1, x_3, y_1, y_3) &= x_1 \otimes y_1 \oplus x_1 \otimes y_3 \oplus x_3 \otimes y_1 \\ f_2'(x_1, x_2, y_1, y_2) &= x_2 \otimes y_2 \oplus x_1 \otimes y_2 \oplus x_2 \otimes y_1 \\ f_3'(x_2, x_3, y_2, y_3) &= x_3 \otimes y_3 \oplus x_2 \otimes y_3 \oplus x_3 \otimes y_2 \end{aligned} \tag{3.4}$$

Nous pouvons observer que f_1', f_2' et f_3' sont fonctionnellement indépendantes des paires de morceaux (x_2, y_2), (x_3, y_3) et (x_1, y_1), respectivement. En plus d'être correcte, d'après la Définition 20, la réalisation 3-masquée $\mathbf{F}' = (f_1', f_2', f_3')$ de la fonction F est glitches-immune à l'ordre 1. En conséquence, $\mathbf{F}'$ est sécurisée dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches.

Uniformité. Nous vérifions enfin la propriété d'uniformité de $\mathbf{F}'$. En reprenant les notations de la Définition 21, on a $\mathbf{z} = (f_1', f_2', f_3')$ et la fonction de reconstruction de sortie $rec'(\mathbf{z}) = \bigoplus_{i=1}^{3} f_i' = x \otimes y$. Nous devons donc vérifier que le nombre d'antécédents de la forme $(\mathbf{x}, \mathbf{y}) = (x_1, x_2, x_3, y_1, y_2, y_3) \in \mathrm{GF}(2)^{3\times 2}$ pour chaque image $\mathbf{z} \in \mathrm{GF}(2)^{3\times 1}$ est égal à $2^{(3-1)\times(2-1)}$ (*i.e.* 4) fois le nombre d'antécédents $(x, y) \in (\mathrm{GF}(2))^2$ pour chaque image $\mathrm{F}(x, y) = rec'(\mathbf{z}) = x \otimes y$.

- Si l'image $\mathrm{F}(x, y) = x \otimes y$ est égale à 0, il existe 3 paires d'antécédents $(x, y) = \{(0, 0), (1, 0), (0, 1)\}$. La partie gauche du Tableau 2 montre le nombre d'antécédents de la forme $(\mathbf{x}, \mathbf{y})$ (pour les valeurs de (x, y) fixées) pour chaque image $\mathbf{z}$ telle que $rec'(\mathbf{z}) = 0$ [2]. Ce nombre doit être égal à $4 \times 3 = 12$. À la lecture du Tableau 2 on constate que ce n'est pas le cas.
- Si l'image $\mathrm{F}(x, y) = x \otimes y$ est égale à 1, il existe 1 paire d'antécédent $(x, y) = (1, 1)$. La partie droite du Tableau 2 montre le nombre d'antécédents de la forme $(\mathbf{x}, \mathbf{y})$ pour chaque image $\mathbf{z}$ telle que $rec'(\mathbf{z}) = 1$. Ce nombre doit être égal à $4 \times 1 = 4$. À la lecture du Tableau 2 on constate que ce n'est pas le cas.

$(\mathbf{x},\mathbf{y})$ tel que $(x,y)=$	$\mathbf{z}=(f'_1,f'_2,f'_3)$							
	$rec'(\mathbf{z})=0$				$rec'(\mathbf{z})=1$			
	000	011	101	110	001	010	100	111
(0,0)	7	3	3	3	0	0	0	0
(0,1)	7	3	3	3	0	0	0	0
(1,0)	7	3	3	3	0	0	0	0
(1,1)	0	0	0	0	5	5	5	1

TABLEAU 2 – Nombre d'antécédents de la forme $(\mathbf{x},\mathbf{y})$ pour les valeurs de (x,y) données pour chaque image $\mathbf{z}$ telle que $rec'(\mathbf{z})=0$ (à gauche) et telle que $rec'(\mathbf{z})=1$ (à droite)

La réalisation 3-masquée $\mathbf{F}'$ n'est donc pas uniforme. Pour corriger ce problème, il existe une technique proposée par Moradi *et. al.* [64] dans l'état de l'art, appelée *remasquage* et présentée ci-dessous.

Proposition 1. ***(Remasquage d'une réalisation** d**-masquée [64] [9].)*** *Soient d' un entier strictement positif et $\mathbf{F}=(f_1,...,f_{d'})$ une réalisation d'-masquée d'une (n,m)-fonction* F. *Soient également $(d'-1)$ variables aléatoires $(r_1,...,r_{d'-1})$ statistiquement indépendantes et uniformément distribuées dans* $\mathrm{GF}(2^m)$. *Alors la réalisation d'-masquée $\mathbf{H}=(h_1,...,h_{d'})$ de* F *définie par :*

$$\begin{aligned} h_1 &= f_1 \oplus r_1 \\ h_2 &= f_2 \oplus r_2 \\ &... \\ h_{d'-1} &= f_{d'-1} \oplus r_{d'-1} \\ h_{d'} &= f_{d'} \oplus \bigoplus_{i=1}^{d'-1} r_i, \end{aligned}$$

est uniforme.

Le résultat de la Proposition 1 est basée sur la propriété suivante :

Propriété 1. *Soient a et b de variables aléatoires indépendantes définies sur le même espace de probabilité. Si a est indépendant de b et b est uniforme, alors $a \oplus b$ est uniforme et indépendant de a.*

Nous appliquons la technique de remasquage à $\mathbf{F}'$. Nous introduisons deux variables aléatoires r_1 et r_2 statistiquement indépendantes et uniformément distribuées dans GF(2). Soit $\mathbf{r}=(r_1,r_2,r_1\oplus r_2)$ le partage additif en 3-morceaux de $r=0$. Sans altérer le caractère correct et glitches-immune à l'ordre 1, nous obtenons une nouvelle réalisation 3-masquée $\mathbf{F}''=(f''_1,f''_2,f''_3)$ telle que :

$$\begin{aligned} f''_1(x_1,x_3,y_1,y_3,r_1) &= f'_1 \oplus r_1 = x_1\otimes y_1 \oplus x_1 \otimes y_3 \oplus x_3 \otimes y_1 \oplus r_1 \\ f''_2(x_1,x_2,y_1,y_2,r_2) &= f'_2 \oplus r_2 = x_2\otimes y_2 \oplus x_1 \otimes y_2 \oplus x_2 \otimes y_1 \oplus r_2 \\ f''_3(x_2,x_3,y_2,y_3,r_1,r_2) &= f'_3 \oplus r_1 \oplus r_2 = x_3\otimes y_3 \oplus x_2 \otimes y_3 \oplus x_3 \otimes y_2 \oplus r_1 \oplus r_2 \end{aligned} \tag{3.5}$$

Nous vérifions maintenant que $\mathbf{F}''$ soit uniforme. En reprenant les notations de la Définition 21, on a $\mathbf{z}=(f''_1,f''_2,f''_3)$ et la fonction de reconstruction de sortie $rec'(\mathbf{z})=\bigoplus_{i=1}^{3} f''_i = x\otimes y$. Nous devons donc vérifier que le nombre d'antécédents de la forme $(\mathbf{x},\mathbf{y},\mathbf{r})=(x_1,x_2,x_3,y_1,y_2,y_3,r_1,r_2,r_1\oplus r_2)\in \mathrm{GF}(2)^{3\times 3}$ pour chaque image $\mathbf{z}\in\mathrm{GF}(2)^{3\times 1}$ est égal à $2^{(3-1)\times(3-1)}$ (*i.e.* 16) fois le nombre d'antécédents $(x,y,r)\in(\mathrm{GF}(2))^3$ pour chaque image $\mathrm{F}(x,y,r)=rec'(\mathbf{z})=x\otimes y\oplus r=x\otimes y$.

2. Par exemple, l'image $\mathbf{z}=(0,0,0)$ a 7 antécédents $(\mathbf{x},\mathbf{y})=\{(0,0,0,0,0,0),(0,0,0,0,1,1),(0,0,0,1,0,1),(0,0,0,1,1,0),(0,1,1,0,0,0),(1,0,1,0,0,0),(1,1,0,0,0,0)\}$ tels que $(x,y)=(0,0)$.

- Si l'image $F(x,y,r) = x \otimes y$ est égale à 0, il existe 3 antécédents $(x,y,r) = \{(0,0,0), (1,0,0), (0,1,0)\}$. La partie gauche du Tableau 3 montre le nombre d'antécédents de la forme $(\mathbf{x},\mathbf{y},\mathbf{r})$ (pour les valeurs de (x,y,r_1,r_2) fixées) pour chaque image $\mathbf{z}$ telle que $rec'(\mathbf{z}) = 0$. Ce nombre doit être égal à $16 \times 3 = 48$. Il est facile de voir dans le Tableau 3 que c'est le cas. Il existe en effet 48 antécédents possibles pour chaque image $\mathbf{z}$.
- Si l'image $F(x,y,r) = x \otimes y$ est égale à 1, il existe 1 antécédent $(x,y,r) = (1,1,0)$. La partie droite du Tableau 3 montre le nombre d'antécédents de la forme $(\mathbf{x},\mathbf{y},\mathbf{r})$ pour chaque image $\mathbf{z}$ telle que $rec'(\mathbf{z}) = 1$. Ce nombre doit être égal à $16 \times 1 = 16$. Il est facile de voir dans le Tableau 3 que c'est le cas. Il existe en effet 16 antécédents possibles pour chaque image $\mathbf{z}$.

D'après la Définition 21, la réalisation 3-masquée $\mathbf{F}''$ est donc uniforme.

$(\mathbf{x},\mathbf{y},\mathbf{r})$ tel que $(x,y,r_1,r_2,r_1 \oplus r_2) =$	$\mathbf{z} = (f_1'', f_2'', f_3'')$							
	$rec'(\mathbf{z}) = 0$				$rec'(\mathbf{z}) = 1$			
	000	011	101	110	001	010	100	111
$(0,0,r_1,r_2,r_1 \oplus r_2)$	16	16	16	16	0	0	0	0
$(0,1,r_1,r_2,r_1 \oplus r_2)$	16	16	16	16	0	0	0	0
$(1,0,r_1,r_2,r_1 \oplus r_2)$	16	16	16	16	0	0	0	0
$(1,1,r_1,r_2,r_1 \oplus r_2)$	0	0	0	0	16	16	16	16

TABLEAU 3 – Nombre d'antécédents de la forme $(\mathbf{x},\mathbf{y},\mathbf{r})$ pour les valeurs de $(x,y,r_1,r_2,r_1 \oplus r_2)$ données pour chaque image $\mathbf{z}$ telle que $rec'(\mathbf{z}) = 0$ (à gauche) et telle que $rec'(\mathbf{z}) = 1$ (à droite)

La réalisation 3-masquée $\mathbf{F}''$ étant correcte, glitches-immune à l'ordre 1 et uniforme, nous venons de montrer comment construire une implémentation à seuil de la fonction $F(x,y) = x \otimes y$, sécurisée à l'ordre 1 en présence de glitches. Cette solution n'est pas unique, nous en présentons une deuxième ci-dessous, basée sur le travail de Ishaï *et. al.* [42], qui proposent en 2003 un schéma de masquage d'une multiplication entre deux bits (soit notre fonction F) dans un circuit. Leur schéma utilise la technique de remasquage, ce qui permet d'assurer une sécurité vis-à-vis d'un sondage à l'ordre t.

Exemple 2 : à partir des multiplications de Ishaï *et. al.*

Le schéma de Ishaï *et. al.* [42], que nous appelons *schéma ISW* dans la suite, est défini dans GF(2). Il prend en entrée deux partages additifs en d-morceaux $(x_1,...,x_d)$ et $(y_1,...,y_d)$ des deux bits d'entrée x et y à multiplier et retourne un partage additif en d-morceaux du résultat de la multiplication. Le schéma ISW suit les étapes suivantes :

— **Étape 1 :** Pour tout $1 \leq i < j \leq d$, un bit $r_{i,j}$ est tiré aléatoirement et uniformément dans GF(2) afin de faire du remasquage. Le bit $r_{j,i}$ est ensuite calculé tel que $r_{j,i} = (r_{i,j} \oplus x_i \otimes y_j) \oplus x_j \otimes y_i$ tel que $r_{j,i}$ et $r_{i,j}$ soient indépendants de x_i, x_j, y_i et y_j. L'utilisation des parenthèses indique l'ordre dans lequel les opérations sont exécutées, ce qui est primordial pour la sécurité du schéma ISW.

— **Étape 2 :** Les bits de sorties $(z_1,...,z_d)$ sont calculés tels que pour $1 \leq i \leq d$, $z_i = x_i \otimes y_i \oplus \bigoplus_{j \neq i} r_{i,j}$. La multiplication dans GF(2) de x par y est égale à z telle que $z = z_1 \oplus ... \oplus z_d$.

Une autre façon de représenter le schéma ISW est de le visualiser sous forme de matrice, comme présenté par Rivain *et. al.* dans [74]. Soit XY la matrice 3×3 contenant tous les termes (*i.e.* les bits) du développement de $(x_1 \oplus x_2 \oplus x_3) \otimes (y_1 \oplus y_2 \oplus y_3)$ telle que :

$$\mathrm{XY} = \begin{pmatrix} x_1 \otimes y_1 & x_1 \otimes y_2 & x_1 \otimes y_3 \\ x_2 \otimes y_1 & x_2 \otimes y_2 & x_2 \otimes y_3 \\ x_3 \otimes y_1 & x_3 \otimes y_2 & x_3 \otimes y_3 \end{pmatrix}.$$

Ishaï *et. al.* proposent de décomposer XY en l'addition de deux matrices A et B de tailles 3×3 (*i.e.* $XY = A \oplus B$) telle que :

$$A = \begin{pmatrix} x_1 \otimes y_1 & x_1 \otimes y_2 & x_1 \otimes y_3 \\ 0 & x_2 \otimes y_2 & x_2 \otimes y_3 \\ 0 & 0 & x_3 \otimes y_3 \end{pmatrix}, \qquad B = \begin{pmatrix} 0 & 0 & 0 \\ x_2 \otimes y_1 & 0 & 0 \\ x_3 \otimes y_1 & x_3 \otimes y_2 & 0 \end{pmatrix}.$$

Dans l'étape 1 du schéma ISW ci-dessus, pour tout $1 \leq i < j \leq 3$, un bit $r_{i,j}$ est tiré aléatoirement et uniformément dans GF(2). Soit Z une matrice de taille 3×3 dont les termes de la $i^{ème}$ colonne sont contenus dans l'élément z calculé à l'étape 2 du schéma ISW. Nous avons :

$$Z = A \oplus \begin{pmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{pmatrix} \oplus B^{\perp},$$

où $B^{\perp}$ représente la transposée de la matrice B. Nous obtenons le calcul de la matrice Z suivant :

$$Z = \begin{pmatrix} x_1 \otimes y_1 & x_1 \otimes y_2 & x_1 \otimes y_3 \\ 0 & x_2 \otimes y_2 & x_2 \otimes y_3 \\ 0 & 0 & x_3 \otimes y_3 \end{pmatrix} \oplus \begin{pmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & x_2 \otimes y_1 & x_3 \otimes y_1 \\ 0 & 0 & x_3 \otimes y_2 \\ 0 & 0 & 0 \end{pmatrix},$$

soit :

$$Z = \begin{pmatrix} x_1 \otimes y_1 & (x_1 \otimes y_2 \oplus r_{1,2}) \oplus x_2 \otimes y_1 & (x_1 \otimes y_3 \oplus r_{1,3}) \oplus x_3 \otimes y_1 \\ r_{1,2} & x_2 \otimes y_2 & (x_2 \otimes y_3 \oplus r_{2,3}) \oplus x_3 \otimes y_2 \\ r_{1,3} & r_{2,3} & x_3 \otimes y_3 \end{pmatrix}.$$

Pour des partages additifs en 3-morceaux (x_1, x_2, x_3) de x et (y_1, y_2, y_3) de y en entrée, le bit z_i ($i \in [1..3]$) de la réalisation 3-masquée $\mathbf{Z}$ de $F(x, y) = x \otimes y$ obtenu en suivant le schéma ISW présenté ci-dessus est défini par l'addition des termes de la $i^{ème}$ colonne de la matrice Z ci-dessus. On obtient ainsi la réalisation $\mathbf{Z} = (z_1, z_2, z_3)$ telle que :

$$z_1(x_1, y_1, r_{1,2}, r_{1,3}) = x_1 \otimes y_1 \oplus r_{1,2} \oplus r_{1,3} \tag{3.6}$$

$$z_2(x_1, x_2, y_1, y_2, r_{1,2}, r_{2,3}) = (x_1 \otimes y_2 \oplus r_{1,2}) \oplus x_2 \otimes y_1 \oplus x_2 \otimes y_2 \oplus r_{2,3}$$

$$z_3(x_1, x_2, x_3, y_1, y_2, y_3, r_{1,3}, r_{2,3}) = (x_1 \otimes y_3 \oplus r_{1,3}) \oplus x_3 \otimes y_1 \oplus (x_2 \otimes y_3 \oplus r_{2,3}) \oplus x_3 \otimes y_2 \oplus x_3 \otimes y_3 \tag{3.7}$$

La réalisation $\mathbf{Z}$ est correcte car l'addition de ses sous-fonctions retourne le résultat de la multiplication de x par y. De plus, le schéma ISW utilise la technique de remasquage ce qui assure l'uniformité de $\mathbf{Z}$. Cependant, comme z_3 est fonctionnellement dépendante de tous les morceaux de x et de y, $\mathbf{Z}$ n'est pas glitches-immune à l'ordre 1. Nous proposons de transformer le schéma ISW en une implémentation à seuil de la fonction F. Comme pour $\mathbf{F}$, une solution possible pour corriger le problème est de déplacer les termes contenant x_1 et y_1 de l'équation 3.7 dans l'équation 3.6. Cette modification n'altère pas le caractère de correction et d'uniformité de $\mathbf{Z}$. Ainsi, nous obtenons une nouvelle réalisation 3-masquée $\mathbf{Z}' = (z'_1, z'_2, z'_3)$ définie par :

$$\begin{aligned} z'_1(x_1, x_3, y_1, y_3, r_{1,2}, r_{1,3}) &= x_1 \otimes y_1 \oplus r_{1,2} \oplus r_{1,3} \oplus x_1 \otimes y_3 \oplus x_3 \otimes y_1 \\ z'_2(x_1, x_2, y_1, y_2, r_{1,2}, r_{2,3}) &= x_2 \otimes y_2 \oplus (r_{1,2} \oplus x_1 \otimes y_2) \oplus x_2 \otimes y_1 \oplus r_{2,3} \\ z'_3(x_2, x_3, y_2, y_3, r_{1,3}, r_{2,3}) &= x_3 \otimes y_3 \oplus r_{1,3} \oplus (r_{2,3} \oplus x_2 \otimes y_3) \oplus x_3 \otimes y_2 \end{aligned} \tag{3.8}$$

Nous pouvons observer que z'_1, z'_2 et z'_3 sont fonctionnellement indépendantes des paires de morceaux $(x_2, y_2), (x_3, y_3)$ et (x_1, y_1), respectivement. En plus d'être correcte et uniforme, la réalisation 3-masquée $\mathbf{Z}' = (z'_1, z'_2, z'_3)$ de la fonction F est donc glitches-immune à l'ordre 1. En conséquence, $\mathbf{Z}'$ est sécurisée dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches.

La construction de $\mathbf{Z}'$ est une deuxième implémentation à seuil possible de la fonction F.

Pour conclure cette section, nous avons défini les propriétés sur lesquelles les implémentations à seuil se construisent. En particulier nous avons décrit dans les systèmes d'équations 3.5 et 3.8 deux implémentations à seuil possibles $\mathbf{F}''$ et $\mathbf{Z}'$, respectivement, de la fonction booléenne calculant la multiplication entre deux bits. Si la correction et la propriété glitches-immune sont faciles à mettre en place et peu coûteuses en termes de performances, ce n'est pas toujours le cas pour l'uniformité. La section suivante aborde cette difficulté.

3.3 L'uniformité, un sujet de recherche ouvert

Protéger un algorithme symétrique (e.g. `AES`, `SM4`) avec les implémentations à seuil consiste à appliquer la contre-mesure sur chacune des fonctions impliquées dans le chiffrement d'un message. Comme illustré en Figure 31, une telle protection peut être vue comme une succession de circuits composés d'un certain nombre de cycles et de registres. Les défis à relever lorsque l'on souhaite réaliser une implémentation à seuil matérielle d'une (n,m)-fonction sont de minimiser le temps d'exécution (*i.e.* le nombre de cycles), la surface (*i.e.* le nombre de registres et de portes équivalentes NAND) et d'assurer les trois propriétés vues dans la section précédente. Parmi ces dernières, l'uniformité est la plus délicate à mettre en oeuvre. En effet, la technique de remasquage présentée en Proposition 1 permet d'assurer la propriété tout en introduisant de nouvelles variables aléatoires au sein du circuit. Cependant, ces nouveaux bits doivent être générés et stockés dans des registres, ce qui augmente le temps d'exécution et augmente la surface. Ainsi, réduire l'aléa supplémentaire tout en assurant l'uniformité d'une réalisation est un sujet encore actif et ouvert dans la communauté cryptographique [10, 22, 40, 64, 82].

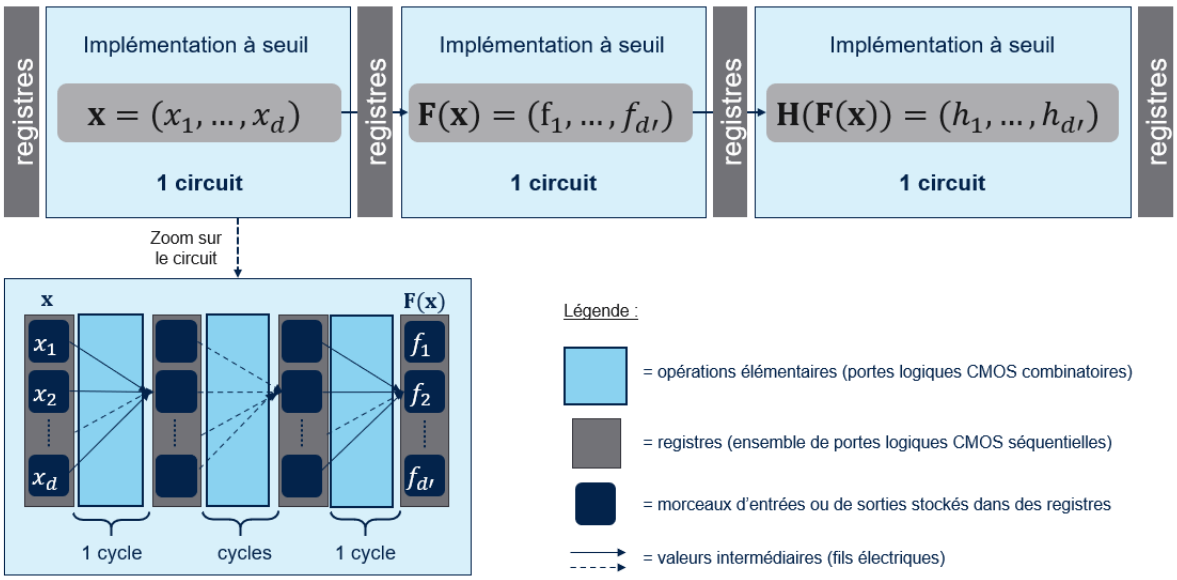


FIGURE 31 – Illustration d'implémentations à seuils matérielles **F** et **H** de 2 fonctions en électronique

La partie d'un bloc chiffré la plus délicate à sécuriser est sa fonction non-linéaire (*i.e.* boîte-S). D'après le Théorème 1 et le travail de Nikova *et. al.* [66], il est nécessaire de partager les données sensibles d'une fonction non-linéaire en 3 morceaux minimum afin d'assurer la propriété de glitches-immunité à l'ordre 1. De nombreuses études [10, 22, 40, 64, 82] se sont ainsi concentrées sur l'implémentation à seuil en 3 morceaux de la boîte-S de l'`AES`, dans le but de minimiser le nombre de portes équivalentes NAND (notée GE), la quantité de bits aléatoires ajoutés afin d'assurer l'uniformité et le nombre de cycles nécessaires pour protéger le calcul d'une boîte-S. Les

performances de leurs travaux sont affichées dans le Tableau 4.

Design	boîte-S [kGE]	Aléas [bits/boîte-S]	Nb cycles
Cnudde *et. al.* [22]	1.9	54	5
Moradi *et. al.* [64]	4.2	44	7
Gross *et. al.* [40]	2.6	18	6
Bilgin *et. al.* [10]	2.2	16	5
Sugawara [82]	3.5	0	4

TABLEAU 4 – Propositions de l'état de l'art - implémentations matérielles à seuil en 3-morceaux de la boîte-S de l'AES, sécurisées à l'ordre 1 en présence de glitches

Nous portons de l'attention à la dernière ligne du Tableau 4, où Sugawara [82] propose une réalisation 3-masquée de la boîte-S de l'AES sans ajouter de nouveaux bits aléatoires dans son implémentation. L'auteur base son travail sur la technique de Daemen [26], appelée *changing of the guards*. En effet, Daemen propose de remasquer les sous-fonctions d'une réalisation 3-masquée d'une (n,m)-fonction en utilisant des morceaux de données déjà existantes au sein de l'implémentation, au lieu d'en introduire de nouveaux. Nous donnons ci-dessous une illustration de sa méthode dans le cas de l'exécution en parallèle de la même boîte-S sur plusieurs (e.g. 3) entrées.

Soient (S_1, S_2, S_3) une réalisation 3-masquée correcte et glitches-immune à l'ordre 1 de la boîte-S. Pour $j \in \{1,2,3\}$, on considère des partages additifs en 3-morceaux $(x_{j,1}, x_{j,2}, x_{j,3})$ des entrées $x_j \in \mathrm{GF}(2)^8$. Pour assurer l'uniformité, l'idée de Daemen consiste à remplacer (S_1, S_2, S_3) par la réalisation 3-masquée $(S_1 \oplus x_{j-1,2} \oplus x_{j-1,3}, S_2 \oplus x_{j-1,3}, S_3 \oplus x_{j-1,2})$, où l'opération $(j-1)$ s'effectue modulo 3. On se réfère à la Figure 32 avec ces notations.

Ainsi, les éléments ajoutés aux sous-fonctions (S_1, S_2, S_3) font partis du partage en 3-morceaux du précédent octet manipulé dans l'implémentation. De cette manière, la technique de remasquage décrite en Proposition 1 est vérifiée. Dans ce manuscrit, pour assurer l'uniformité, nous privilégions la technique de Daemen afin de ne pas ajouter de génération de nouveaux aléas au sein de nos constructions.

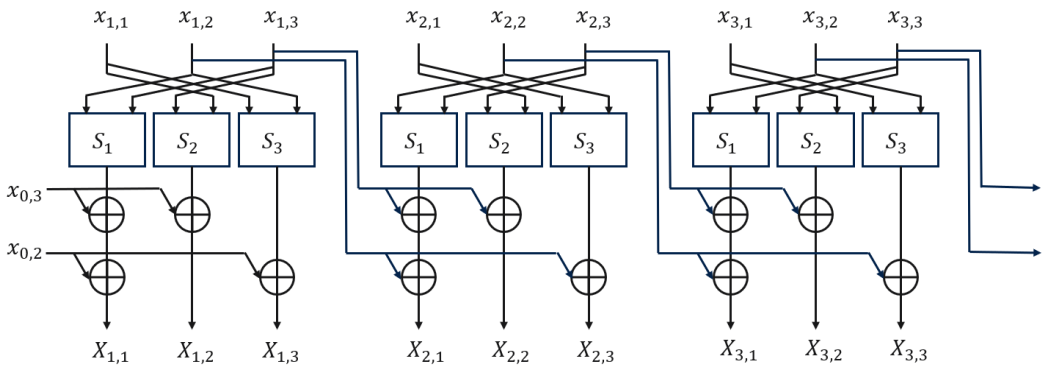


FIGURE 32 – Technique de Daemen [26] - *Changing of the guards*

Dans la section suivante, nous décrivons la méthode employée par Nikova *et. al.* [65] afin de construire une implémentation à seuil d'une (n,m)-fonction, sécurisée à l'ordre 1 en présence de glitches.

3.4 Méthode de Nikova *et. al.*

Nous terminons ce chapitre par la présentation pas à pas de la méthode de Nikova *et. al.* , permettant de construire une implémentation à seuil d'une (n,m)-fonction, sécurisée à l'ordre 1 en présence de glitches. Nous reprenons notre exemple avec la fonction $F : (x,y) \in (GF(2))^2 \mapsto x \otimes y \in GF(2)$. En application du Théorème 1, les données sensibles x et y sont masquées additivement en 3 morceaux (*i.e.* $x = x_1 \oplus x_2 \oplus x_3$ et $y = y_1 \oplus y_2 \oplus y_3$). La technique décrite par Nikova *et. al.* afin de construire une réalisation 3-masquée $\mathbf{H} = (h_1, h_2, h_3)$ de la fonction F est donnée par les étapes suivantes.

— **Étape 1 :** Développer $F(\bigoplus_{i=1}^{3} x_i \otimes \bigoplus_{i=1}^{3} y_i)$;

$$\begin{aligned} F(\bigoplus_{i=1}^{3} x_i \otimes \bigoplus_{i=1}^{3} y_i) &= (x_1 \oplus x_2 \oplus x_3) \otimes (y_1 \oplus y_2 \oplus y_3) \\ &= x_1 \otimes y_1 \oplus x_1 \otimes y_2 \oplus x_1 \otimes y_3 \\ &\oplus x_2 \otimes y_1 \oplus x_2 \otimes y_2 \oplus x_2 \otimes y_3 \\ &\oplus x_3 \otimes y_1 \oplus x_3 \otimes y_2 \oplus x_3 \otimes y_3. \end{aligned} \tag{3.9}$$

— **Étape 2 :** Construire h_1 avec tous les termes de l'Équation 3.9 exceptés ceux contenant les morceaux x_1 et y_1. On a :

$$h_1(x_2, x_3, y_2, y_3) = x_2 \otimes y_2 \oplus x_2 \otimes y_3 \oplus x_3 \otimes y_2 \oplus x_3 \otimes y_3.$$

— **Étape 3 :** Construire h_2 avec tous les termes restants de l'Équation 3.9 exceptés ceux contenant les morceaux x_2 et y_2. On a :

$$h_2(x_1, x_3, y_1, y_3) = x_1 \otimes y_1 \oplus x_1 \otimes y_3 \oplus x_3 \otimes y_1.$$

— **Étape 4 :** Construire h_3 avec tous les termes restants de l'Équation 3.9. Normalement les morceaux x_3 et y_3 ne doivent pas faire partis de f_3. On a :

$$h_3(x_1, x_2, y_1, y_2) = x_1 \otimes y_2 \oplus x_2 \otimes y_1.$$

La fonction de reconstruction de sortie rec' consiste à additionner les sous-fonctions de $\mathbf{H}$ tel que $rec'(\mathbf{H}(\mathbf{x})) = h_1 \oplus h_2 \oplus h_3 = x \otimes y$. Comme la Définition 19 est vérifiée, la réalisation $\mathbf{H}$ est correcte. De plus, chaque sous-fonction h_i est fonctionnellement indépendante des paires (x_i, y_i). La réalisation 3-masquée $\mathbf{H}$ satisfait donc la Définition 20, et est glitches-immune à l'ordre 1. En conséquence, $\mathbf{H}$ est sécurisée l'ordre 1 en présence de glitches.

Enfin, pour toutes les valeurs possibles des images de $\mathbf{H} = (h_1, h_2, h_3) \in GF(2)^3$, nous avons calculé le nombre d'antécédents $(\mathbf{x}, \mathbf{y}) \in ((GF(2))^2)^3$ possibles. En suivant le même raisonnement que pour $\mathbf{F}'$ en Équations 3.4 et dans le Tableau 2, nos résultats montrent qu'il existe plus d'antécédents pour l'image $(h_1, h_2, h_3) = (0,0,0)$ que pour une autre image. La réalisation $\mathbf{H}$ n'est donc pas uniforme. Afin de pallier à ce problème, Nikova *et. al.* emploient la technique de remasquage (Proposition 1) pour transformer $\mathbf{H}$ en une réalisation $\mathbf{H}' = (h'_1, h'_2, h'_3)$ uniforme telle que :

$$\begin{aligned} h'_1 &= x_2 \otimes y_2 \oplus x_2 \otimes y_3 \oplus x_3 \otimes y_2 \oplus x_3 \otimes y_3 \oplus r_1 \\ h'_2 &= x_1 \otimes y_1 \oplus x_1 \otimes y_3 \oplus x_3 \otimes y_1 \oplus r_2 \\ h'_3 &= x_1 \otimes y_2 \oplus x_2 \otimes y_1 \oplus r_1 \oplus r_2, \end{aligned} \tag{3.10}$$

où r_1 et r_2 sont deux bits aléatoires statistiquement indépendantes et uniformément distribuées dans GF(2). La réalisation 3-masquée $\mathbf{H}'$ obtenue avec la technique de Nikova *et. al.* est une implémentation à seuil possible de la fonction F.

3.5 Conclusion

Pour conclure, comme montré en 2005 par Mangard *et. al.* [56], nous avons présenté les *glitches* comme un phénomène inattendu et une faille de sécurité importante au sein d'un outil cryptographique, permettant à un attaquant de retrouver des informations sensibles. Cette vulnérabilité est due à des délais de propagation différents des signaux électriques entre les portes logiques d'un circuit. Comme les glitches peuvent apparaître n'importe quand, il est primordial de s'en protéger à tout instant lors de l'exécution de l'outil cryptographique. Nous avons ainsi décrit le principe des *implémentations à seuil* d'une (n,m)-fonction, proposé par Nikova *et. al.* [65] en 2006. Ce nouveau moyen de défense est basé sur un découpage en plusieurs morceaux des informations sensibles et des fonctions les manipulant. Ce découpage doit satisfaire trois propriétés afin d'être fonctionnel, sécurisé contre les glitches à l'ordre t et correctement composé, à savoir la *correction*, la *glitches-immunité à l'ordre t* et l'*uniformité*, respectivement. Nous avons illustré ces propriétés par la description d'une implémentation à seuil, sécurisée à l'ordre 1 en présence de glitches, de la fonction booléenne F calculant une multiplication entre deux bits. La construction obtenue $\mathbf{F}''$ est détaillée dans le système d'équations 3.5. Il est important de noter qu'il existe plusieurs manières de construire une implémentation à seuil d'une même fonction. Nous avons par exemple donné une autre implémentation à seuil de la fonction booléenne F, basée sur le schéma ISW [42], définie par $\mathbf{Z}'$ dans le système d'équations 3.8. De plus, nous avons mis en valeur la difficulté de mettre en place la propriété d'uniformité, pouvant être la cause d'une baisse de performances en temps d'exécution et de surface. Nous avons notamment décrit la technique de remasquage et la méthode de Daemen afin d'assurer l'uniformité de deux manière différentes. Enfin, nous avons présenté la méthode initiale de Nikova *et. al.* , permettant de construire une réalisation correcte et sécurisée à l'ordre 1 en présence de glitches de la fonction F. Associée à la technique de remasquage, la réalisation de Nikova *et. al.* peut facilement s'étendre en une implémentation à seuil de la fonction F comme définie par $\mathbf{H}'$ dans le système d'équations 3.10. Ce chapitre décrit donc 3 implémentations à seuil différentes et possibles pour la fonction F : $\mathbf{F}''$, $\mathbf{Z}'$ et $\mathbf{H}'$.

Dans le chapitre suivant, nous visons à contribuer à l'état de l'art présenté dans le Tableau 4 en présentant de nouvelles idées de protection contre les attaques par observations en présence de glitches. Nous proposons notamment notre propre implémentation à seuil du calcul d'un monôme (e.g. celui de la fonction inverse de l'`AES`) et de manière plus étendue d'une (n,m)-fonction, sécurisée à l'ordre 1 en présence de glitches.

Chapitre 4

Évaluation monomiale des fonctions polynomiales protégée par les implémentations à seuil

– Illustration sur l'AES–

Sommaire

Ce chapitre s'oriente autour de trois axes principaux. Tout d'abord, nous proposons une implémentation à seuil du calcul d'un monôme. Notre construction est mathématiquement prouvée comme étant sécurisée à l'ordre 1 en présence de glitches. Afin de l'illustrer, nous l'appliquons sur la fonction inverse de l'AES (*i.e.* $x^{254} \in \mathrm{GF}(2^8)$). Nous avons validé notre méthode avec une implémentation matérielle sur FPGA de manière à comparer nos performances à celles de l'état de l'art,

présentées dans le Tableau 4. Les critères pris en compte sont le nombre de cycles, le nombre de portes équivalentes et le nombre de bits aléatoires ajoutés afin de satisfaire la propriété d'uniformité. En complément, nous proposons une implémentation à seuil de toutes les opérations de l'AES, ce qui n'est pas toujours le cas dans la littérature cryptographique.

Un deuxième axe consiste en l'extension de notre construction monomiale de manière générique. Nous proposons une implémentation à seuil du calcul de plusieurs fonctions polynomiales. Nous présentons ainsi la première construction de l'état de l'art, sécurisée à l'ordre 1 en présence de glitches, pouvant s'appliquer sur une autre fonction que la boîte-S de l'AES.

Enfin, dans l'état de l'art, il n'existe qu'une seule alternative [72] à l'approche de Nikova *et. al.* [65] afin de créer une implémentation à seuil d'une (n,m)-fonction (voir la description en Section 3.4). Nous nous sommes ainsi concentrés sur une nouvelle façon de générer une construction sécurisée contre les glitches. Notre proposition est générique, elle peut s'appliquer sur n'importe quelle (n,m)-fonction. Elle présente l'avantage d'être plus facile à mettre en place que celle de Nikova *et. al.* , en connaissance de certains paramètres au préalable. De plus, nous nous plaçons dans un modèle d'attaque plus puissant que précédemment. Notre technique est en effet prouvée mathématiquement sécurisée à n'importe quel ordre en présence de glitches. Nous l'appliquons par exemple sur la multiplication (présente dans la fonction inverse de l'AES), pour une sécurité à l'ordre 2 en présence de glitches.

Les recherches présentées dans ce chapitre ont débouché sur deux articles. Le premier papier [49] a été publié lors de la conférence *Information Security Theory and Practice* en décembre 2019 (WISTP19), il présente les premières idées de recherches sur notre évaluation monomiale basée sur les implémentations à seuil. Le deuxième article [48] a été publié dans le Journal *Cryptography and Communications* en mai 2021. Il s'agit d'une extension du premier article, où des preuves mathématiques sont ajoutées et assurent la sécurité dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches de notre proposition. Nous avons également implémenté notre construction sur FPGA et donné des détails supplémentaires sur l'implémentation à seuil de l'AES, comme la structure sécurisée de la génération des clés de tour. Enfin, les recherches ont également permis d'innover avec la proposition de deux brevets, déposés en avril 2019 et juin 2020 avec Yanis Linge au sein de l'entreprise STMicroelectronics. Ces brevets ont été acceptés et seront publiés sous les noms *fault detection* et 2^{nd} *-order multiplicative masking of the inverse function with threshold implementation*. N'étant pas encore publique, nous ne les décriront pas dans ce manuscrit.

Dans un premier temps, la Section 4.1 décrit notre stratégie de masquage employée afin de sécuriser le calcul d'un monôme avec les implémentations à seuil. Puis, la Section 4.2 introduit la fonction de Dirac comme une solution de la littérature cryptographique permettant de pallier à une vulnérabilité due à notre masquage. Nous présentons également deux implémentations sécurisées de la fonction de Dirac. La Section 4.3 détaille notre implémentation à seuil du calcul d'un monôme, sécurisée à l'ordre 1 en présence de glitches. Notre proposition est ensuite appliquée à la fonction inverse de l'AES dans la Section 4.4. Nous décrivons également une construction de chaque opération de l'AES, avec une sécurité dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches. Dans la Section 4.5, nous montrons comment étendre notre technique sur plusieurs fonctions polynomiales. La Section 4.6 est dédiée à la validation matérielle de la fonctionnalité de notre méthode. Nous comparons également nos performances et celles des autres propositions de l'état de l'art. Enfin, nous proposons dans la Section 4.7 une nouvelle construction d'une implémentation à seuil d'une fonction polynomiale, différente de celle proposée par Nikova *et. al.* (voir Section 3.4) et sécurisée à n'importe quel ordre en présence de glitches.

4.1 Notre stratégie de masquage

Afin de protéger le calcul d'une fonction polynomiale définie sur $\mathrm{GF}(2^n)$, nous nous concentrons sur l'évaluation de chaque monôme de cette fonction. Soit q un entier strictement positif, on introduit alors la fonction $\mathrm{F}_{pow}(x)$ permettant de calculer le monôme x^q. Notre but est de créer un schéma de masquage pour la fonction F_{pow}, sécurisé à l'ordre 1 en présence de glitches (voir Définition 18). Pour cela, nous nous basons sur les principes des implémentations à seuil que nous avons décrits dans le chapitre précédent.

Au cours de l'exécution d'un algorithme cryptographique symétrique, une donnée sensible est manipulée par des opérations linéaires et non-linéaires. En Section 2.4.2, nous avons vu qu'il est facile de calculer la propagation d'un masque additif sur une fonction linéaire (voir Équation 2.4) et celle d'un masque multiplicatif sur une opération non-linéaire (voir Équation 2.5). Par exemple, dans le cas de la boîte-S de l'`AES` (voir Équation 2.2), un octet peut être multiplicativement masqué lors du calcul de son inversion, puis additivement masquée lors du calcul des opérations linéaires. Genelle *et. al.* proposent dans [36] une construction permettant de convertir un masquage additif en un masquage multiplicatif, et vice-versa, de manière à protéger une donnée sensible tout au long de l'exécution d'un algorithme combinant des opérations linéaires et des opérations non-linéaires s'exprimant sous la forme de fonctions puissances (e.g. $\mathrm{INV}_{\mathrm{AES}}(a) = a^{254}$). Leur construction est protégée dans le modèle d'attaque par sondage à l'ordre 1 sans prendre en compte la présence de glitches. Notre idée consiste à améliorer leur travail en minimisant le nombre de conversions et en assurant une sécurité à l'ordre 1 en présence de glitches. Pour cela, nous créons un découpage de notre donnée sensible, contenant à la fois un masque additif et un masque multiplicatif, tous deux accessibles à tout moment. Évidemment, nous devons trouver une combinaison des morceaux telle que nous retrouvons la valeur sensible. De cette façon, si nous devons appliquer à la donnée sensible une fonction linéaire ou non-linéaire, nous utiliserons respectivement un masquage additif ou multiplicatif, grâce au masque approprié.

Pour une protection à l'ordre 1, le Théorème 1 implique que nous devons découper la donnée sensible en au moins 3 morceaux afin de satisfaire les propriétés de correction et de glitches-immunité des implémentations à seuil. Notre découpage est appelé *partage affine* et est défini ci-dessous pour une valeur sensible $x \in \mathrm{GF}(2^n)$.

Définition 22. ***Partage affine en*** 3***-morceaux.*** *Soient* $x \in \mathrm{GF}(2^n)$ *un élément et* β, α *deux variables aléatoires et uniformément distribuées dans* $\mathrm{GF}(2^n)$ *et* $\mathrm{GF}(2^n)^\star$*, respectivement. Soit* $\tilde{x} \in \mathrm{GF}(2^n)$ *la variable définie par* $\tilde{x} = (x \oplus \beta) \otimes \alpha$*. Alors* $\mathbf{x} = (\tilde{x}, \alpha, \beta)$ *est appelé un partage affine en* 3*-morceaux de* x *et est associé à la fonction de reconstruction* $rec_{aff} : \mathbf{x} = (\tilde{x}, \alpha, \beta) \mapsto x = (\tilde{x} \otimes \alpha^{-1}) \oplus \beta$*. Les morceaux* α *et* β *sont appelés masque multiplicatif et masque additif de* x*, respectivement.*

Remarque 6. *Dans la Section 3.2.1, le principe de partage de secret sur lequel les implémentations à seuil se basent stipule que la donnée sensible* x *est découpée en morceaux telle que la combinaison de ces morceaux retourne* x*. Notre stratégie de masquage utilise le même principe, à la nuance que la donnée sensible* x *n'est pas elle-même découpée en morceaux mais est entièrement présente dans le premier morceaux du partage de* x*, masquée par un masque additif et un masque multiplicatif. Ces deux derniers correspondent aux autres morceaux du partage de* x*. La Figure 33 permet de mieux comprendre notre partage affine en* 3*-morceaux de* x*.*

Notre schéma de masquage de la fonction $\mathrm{F}_{pow}(x) = x^q$ consiste à représenter la valeur sensible $x \in \mathrm{GF}(2^n)$ par un partage affine selon la Définition 22. Cependant, le masquage multiplicatif utilisé pour ce découpage doit être implémenté avec attention pour ne pas créer une vulnérabilité au sein de la construction, qui pourrait être exploitée par un attaquant. C'est en 2002 que Trichina *et. al.* [84] et Golic *et. al.* [37] pointent une faille de sécurité, appelée *problème du zéro*, qui apparaît lorsqu'un masquage multiplicatif est appliqué sur une valeur sensible nulle. En effet, pour tout masque multiplicatif $\alpha \in \mathrm{GF}(2^n)^\star$, si un attaquant sait que $\mathrm{F}_{pow}(x \otimes \alpha) = 0$ alors il sera capable

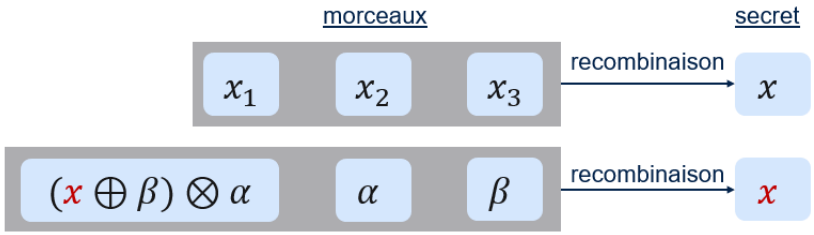


FIGURE 33 – Lien entre le principe du partage de secret de x décrit en Section 3.2.1 (en haut) et notre partage affine en 3-morceaux de la donnée sensible x (en bas)

d'avoir de l'information sur x (*i.e.* $x = \mathrm{F}_{pow}(0)$). Il est donc important d'éviter un tel biais statistique dans notre construction. Pour cela, nous nous basons sur une méthode introduite dans [35] qui implique la fonction de Dirac dont nous rappelons la définition dans le section suivante.

4.2 Fonction de Dirac

4.2.1 Définition

Nous définissons la fonction de Dirac comme suit.

Définition 23. ***Fonction de Dirac.*** *La fonction de Dirac, notée* δ*, est définie sur n'importe quel corps fini telle que* $\delta(x) = 1$ *si* $x = 0$ *et* $\delta(x) = 0$ *sinon.*

Afin de ramener le calcul $\mathrm{F}_{pow}(x) = x^q$ à un calcul sur des valeurs non-nulles, nous utilisons la propriété suivante.

Propriété 2. *Pour tout* $x \in \mathrm{GF}(2^n)$ *et pour tout entier strictement positif* q*, on a :*

$$(x \oplus \delta(x))^q = x^q \oplus \delta(x) \neq 0$$

Preuve. *Pour tout* $x \in \mathrm{GF}(2^n)$ *et pour tout entier strictement positif* q*, on a :*

— *Si* $x = 0$ *alors* $\delta(x) = 1$ *et l'équation suivante est vraie :*

$$(x \oplus \delta(x))^q = (0 \oplus 1)^q = 1^q = 0^q \oplus 1 = x^q \oplus \delta(x) = 1 \neq 0$$

— *Si* $x \neq 0$ *alors* $\delta(x) = 0$ *et l'équation suivante est vraie :*

$$(x \oplus \delta(x))^q = (x \oplus 0)^q = x^q = x^q \oplus 0 = x^q \oplus \delta(x) = x^q \neq 0$$

La Propriété 2 est donc vérifiée.

□

Si x correspond à une valeur sensible dans notre évaluation monomiale, alors x est remplacée par la variable non-nulle $(x \oplus \delta(x))$. Ce masquage est nécessaire. En effet, si un attaquant connaît la valeur (non-masquée) de $(x \oplus \delta(x))$, il est capable de déduire de l'information sur x : si $(x \oplus \delta(x)) = 1$ alors $x = 0$ ou $x = 1$, et si $(x \oplus \delta(x)) = y \neq 1$ alors $x = y$. Il est donc important de protéger le calcul de $(x \oplus \delta(x))$. Ce changement nous permet d'appliquer un masquage multiplicatif sur x, de manière sécurisée même contre le problème du zéro.

Avant d'étudier notre proposition, nous devons nous assurer qu'il est possible de sécuriser le calcul de la fonction de Dirac. Dans la suite, nous présentons deux méthodes différentes afin d'implémenter la fonction de Dirac. La première méthode est basée sur une table de correspondance alors que la deuxième consiste à sécuriser le calcul de la Dirac grâce aux implémentations à seuil. Nous présenterons les avantages et inconvénients de chaque construction. Dans les deux cas, à partir de la connaissance du partage affine de x, nous obtenons une réalisation 2-masquée de la fonction de Dirac $\delta(x)$, sécurisée à l'ordre 1 en présence de glitches.

4.2.2 Table de correspondance de la fonction de Dirac

D'après la Définition 22, la donnée sensible $x \in \mathrm{GF}(2^n)$ est découpée par un partage affine $\mathbf{x} = ((x \oplus \beta) \otimes \alpha, \alpha, \beta)$, où $\alpha \in \mathrm{GF}(2^n)^\star$ et $\beta \in \mathrm{GF}(2^n)$ sont respectivement les masques multiplicatif et additif. À partir de la connaissance des morceaux de $\mathbf{x}$, il est possible de représenter la fonction de Dirac par une table de correspondance.

En effet, grâce à la connaissance de α, la valeur de $x \oplus \beta$ peut être calculée. En introduisant un bit aléatoire $r \in \mathrm{GF}(2^n)$, une implémentation possible de la fonction de Dirac est de pré-calculer une table T d'une capacité de 2^n bits telle que :

$$\mathrm{T}[x \oplus \beta] = \begin{cases} r \oplus 1 & \text{si } x = 0 \\ r & \text{sinon.} \end{cases}$$

L'Algorithme 3 décrit la manière dont la table T est construite. La notation $[a]_b$ désigne un tableau de taille b dont tous les éléments sont égaux à $a \in \mathrm{GF}(2^n)$. La paire $(r, \mathrm{T}[x \oplus \beta])$ représente un partage additif en 2-morceaux de $\delta(x)$:

— Si $x = 0$: $r \oplus \mathrm{T}[x \oplus \beta] = r \oplus r \oplus 1 = 1 = \delta(x)$.

— Sinon : $r \oplus \mathrm{T}[x \oplus \beta] = r \oplus r = 0 = \delta(x)$.

La Définition 23 de la fonction de Dirac est bien vérifiée.

Algorithme 3 Pré-traitement de la table de correspondance de la fonction de Dirac δ

Entrée(s) : $r \in \mathrm{GF}(2^n), x \oplus \beta, \beta \in \mathrm{GF}(2^n)$
Sortie(s) : partage additif en 2-morceaux de $\delta(x)$
1: $\mathrm{T} \leftarrow [0]_{2^n}$
2: $\mathrm{T}[\beta] \leftarrow \mathrm{T}[\beta] \oplus 1$
3: $\mathrm{T} \leftarrow \mathrm{T} \oplus [r]_{2^n}$
4: **retourner** $(r, \mathrm{T}[x \oplus \beta])$

Pendant l'exécution d'un algorithme cryptographique entier (e.g. `AES`), le bit r ne varie pas tandis que le masque additif β évolue (voir Figure 36). On note β' la valeur du masque additif de la variable sensible x' pour laquelle on souhaite calculer la prochaine fonction de Dirac. En prenant en compte cette condition, la table T doit être modifiée à chaque calcul de la fonction de Dirac, de manière à ce que les valeurs contenues aux indices β et β' dans la table soient complémentées à 1. Cette modification est détaillée dans l'Algorithme 4 et peut être effectuée en 1 cycle en parallèle d'autres opérations entre deux couches de calculs de F_{pow}.

Algorithme 4 Modification de la table de correspondance lors du calcul de deux fonctions de Dirac successives

Entrée(s) : $\mathrm{T}, \beta, \beta'$
Sortie(s) : Table T modifiée
1: $\mathrm{T}[\beta] \leftarrow \mathrm{T}[\beta] \oplus 1$
2: $\mathrm{T}[\beta'] \leftarrow \mathrm{T}[\beta'] \oplus 1$
3: **retourner** T

Pour conclure, nous obtenons un partage additif en 2-morceaux $(\delta_1, \delta_2) = (r, \mathrm{T}[x \oplus \beta])$ de $\delta(x)$. Ce partage est glitches-immune à l'ordre 1 car la table n'est pas, par définition, un circuit logique. Le phénomène de glitches n'existe donc pas dans cette construction. La réalisation 2-masquée (δ_1, δ_2) de $\delta(x)$ s'effectue en 1 cycle et nécessite de stocker deux bits $\mathrm{T}[x \oplus \beta]$ et r, ainsi que 2^n bits pour la table.

Dans la suite, nous présentons une solution alternative pour implémenter la fonction de Dirac de façon sécurisée à l'ordre 1 en présence de glitches.

4.2.3 Implémentation à seuil de la fonction de Dirac

Dans le contexte de l'AES, nous travaillons sur le corps fini GF(2^8). Pour tout $x \in$ GF(2^8), la représentation algébrique de la fonction de Dirac se compose de 7 multiplications entre les bits complémentés de x, notés $\overline{x[i]}$ avec $i \in [1..8]$. On a donc :

$$\delta(x) = \overline{x[1]} \otimes \overline{x[2]} \otimes \overline{x[3]} \otimes \overline{x[4]} \otimes \overline{x[5]} \otimes \overline{x[6]} \otimes \overline{x[7]} \otimes \overline{x[8]}. \tag{4.1}$$

Notre idée est de sécuriser le calcul de chaque multiplication bit à bit $\otimes$ de l'Équation 4.1 grâce aux implémentations à seuil. Nous nous référons à l'implémentation à seuil $\mathbf{F}''$: $(\mathbf{x},\mathbf{y}) \in$ GF$(2)^{3\times 2} \mapsto \mathbf{x} \otimes \mathbf{y} \in$ GF$(2)^3$ de la fonction F : $(x,y) \in$ GF$(2)^{1\times 2} \mapsto x \otimes y \in$ GF(2) détaillée dans le système d'équations 3.5 et prouvée sécurisée à l'ordre 1 en présence de glitches. Nous savons que $\mathbf{F}''$ prend en entrée des partages additifs en 3-morceaux des deux bits à multiplier. Or, comme introduit en Section 4.1, notre stratégie d'évaluation monomiale manipule des partages affines en 3-morceaux afin de sécuriser le calcul de F_{pow}. Il est donc important de savoir comment transformer les partages affines en 3-morceaux des deux bits d'entrée en des partages additifs en 3-morceaux afin de les multiplier. Pour cela, nous utilisons l'approche de Daemen [26] (sa technique est détaillée en Section 3.3 avec la Figure 32), qui consiste à exploiter des morceaux déjà présents dans l'implémentation (e.g. x'). En particulier, grâce aux deux partages affines en 3-morceaux $\mathbf{x} = ((x \oplus \beta) \otimes \alpha, \alpha, \beta)$ de x et $\mathbf{x}' = ((x' \oplus \beta') \otimes \alpha', \alpha', \beta')$ de x', nous créons un partage additif en 3-morceaux $\mathbf{x}_{add} = (x \oplus \beta \oplus \beta', \beta', \beta)$. La fonction de reconstruction pour $\mathbf{x}$ et $\mathbf{x}'$ est donnée par rec_{aff} dans la Définition 22. La fonction de reconstruction pour $\mathbf{x}_{add}$ est définie par l'addition des 3 morceaux de $\mathbf{x}_{add}$ soit $x \oplus \beta \oplus \beta' \oplus \beta \oplus \beta' = x$. La Figure 34 illustre notre transformation. Nous supprimons le masque multiplicatif de $\mathbf{x}$ (ici α) et nous rajoutons un masque additif provenant de $\mathbf{x}'$ (ici β'). Cette méthode fonctionne car x et x' sont indépendants et le nouveau masque additif β' est uniforme par définition et indépendant de x'.

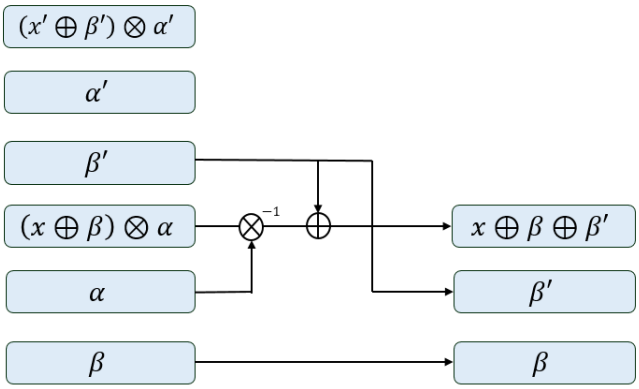


FIGURE 34 – Transformation d'un partage affine $\mathbf{x}$ de $x \in$ GF(2^n) en un partage additif en 3-morceaux $\mathbf{x}_{add}$ du même x

Nous utilisons l'implémentation à seuil $\mathbf{F}''$ de la multiplication bit à bit afin de sécuriser les 7 multiplications de l'Equation 4.1. Pour assurer l'uniformité de $\mathbf{F}''$, nous avons vu qu'il est nécessaire d'introduire 2 bits aléatoires, uniformes et indépendants des morceaux des deux bits à multiplier (voir Equation 3.5). Nous devons donc définir 14 bits aléatoires, uniformes et indépendants des bits de x afin d'assurer l'uniformité des 7 implémentations à seuil des 7 multiplications de la fonction de Dirac de x. Soit $i \in [1..7]$, nous introduisons pour cela 7 partages additifs en 3-morceaux $\mathbf{r}_i$ de la valeur 0, définis par 7 mots ri codés sur 2 bits (*i.e.* les 14 bits) tels que

$\mathbf{r}_i = (ri[0], ri[1], ri[0] \oplus ri[1])$.

La Figure 35 résume notre proposition d'implémentation à seuil de la fonction $\delta(x)$. Nous obtenons un partage additif en 3-morceaux $\mathbf{F}_\delta$ de $\delta(x)$, de la même forme que $\mathbf{F}''$. La construction de $\mathbf{F}_\delta$ s'effectue au moyen de 7 applications de $\mathbf{F}''$ en 3 cycles. Le premier cycle réalise 4 implémentations à seuil $\mathbf{F}''$ prenant en entrée les partages additifs en 3-morceaux $\mathbf{x}_i$ et $\mathbf{x}_{i+1}$ des bits complémentés $\overline{x[i]}$ et $\overline{x[i+1]}$ de x, pour $i = 2 \times j + 1$ tel que $j \in [0..3]$. Puis les résultats sont manipulés à travers 2 implémentations à seuil $\mathbf{F}''$ dans un deuxième cycle. Un dernier cycle calcule $\mathbf{F}_\delta$ au moyen d'une implémentation à seuil $\mathbf{F}''$. La réalisation 3-masquée $\mathbf{F}_\delta$ est donc obtenue par la composition de 7 implémentations à seuil implémentations à seuil $\mathbf{F}''$. Nous avons la propriété suivante :

Propriété 3. *[9] Soit w un entier strictement positif et soient w implémentations à seuil $\mathbf{F}_1, ..., \mathbf{F}_w$ de (n,m)-fonctions $\mathrm{F}_1, ..., \mathrm{F}_w$, respectivement. La réalisation $\mathbf{F}$ définie par la composition $\mathbf{F}_1 \circ ... \circ \mathbf{F}_w$ est une implémentation à seuil de la fonction $\mathrm{F} = \mathrm{F}_1 \circ ... \circ \mathrm{F}_w$.*

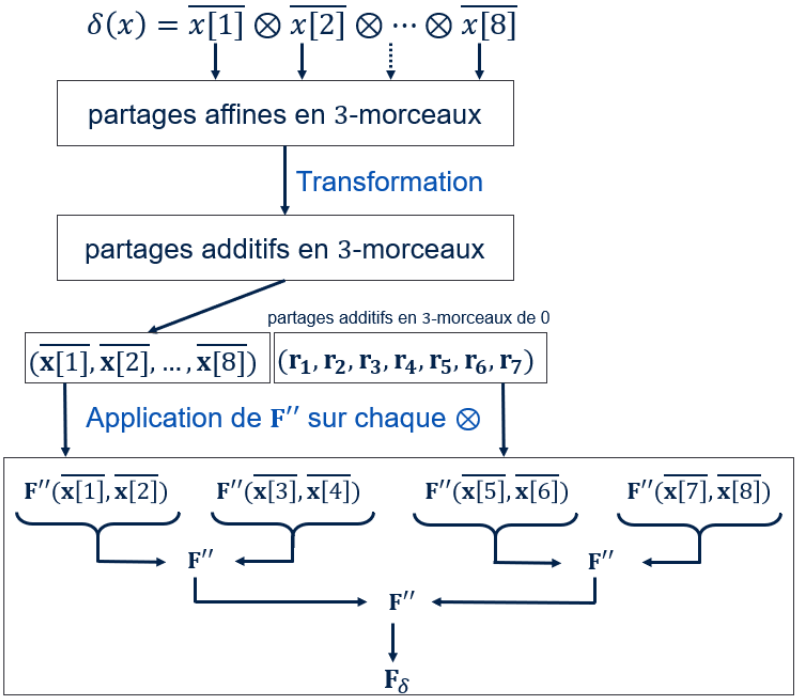


FIGURE 35 – Construction de l'implémentation à seuil $\mathbf{F}_\delta$ de la fonction de Dirac d'un octet x en 3 cycles

La sécurité de la construction de $\mathbf{F}_\delta$ est donnée par la proposition suivante :

Proposition 2. *La réalisation 3-masquée $\mathbf{F}_\delta$ de la fonction de Dirac $\delta : x \in \mathrm{GF}(2)^8 \mapsto \delta(x) \in \mathrm{GF}(2)$ décrite par la construction en 3 cycles en Figure 35 est une implémentation à seuil de la fonction $\delta(x)$. Elle est sécurisée à l'ordre 1 en présence de glitches.*

Preuve. *Soient $\mathbf{x}_i$ un partage additif en 3-morceaux du bit complémenté $\overline{x[i]}$ de x, pour $i \in [1..8]$. La réalisation 3-masquée $\mathbf{F}_\delta$ est définie par la relation suivante :*

$$\mathbf{F}_\delta = \mathbf{F}''(\mathbf{F}''(\mathbf{F}''(\mathbf{x}_1, \mathbf{x}_2), \mathbf{F}''(\mathbf{x}_3, \mathbf{x}_4)), \mathbf{F}''(\mathbf{F}''(\mathbf{x}_5, \mathbf{x}_6), \mathbf{F}''(\mathbf{x}_7, \mathbf{x}_8))).$$

soit la composition de 7 implémentations à seuil de la forme $\mathbf{F}''$. D'après la Propriété 3, $\mathbf{F}_\delta$ est une implémentation à seuil de la fonction $\delta(x)$, sécurisée à l'ordre 1 en présence de glitches.

□

Remarque 7. *Afin d'être en raccord avec la construction sécurisée de la fonction de Dirac sous forme de table de correspondance (*i.e. *pour n'avoir que deux morceaux de* $\mathbf{F}_\delta$*), il est possible de transformer* $\mathbf{F}_\delta = (f_{\delta_1}, f_{\delta_2}, f_{\delta_3})$ *en une réalisation* 2*-masquée* $\mathbf{F'}_\delta = (f'_{\delta_1}, f'_{\delta_2})$ *telle que :* $f'_{\delta_1} = f_{\delta_1} \oplus f_{\delta_2}$ *et* $f'_{\delta_2} = f_{\delta_3}$*, où* $f'_{\delta_1} \oplus f'_{\delta_2} = \delta(x)$*. Cette transformation implique l'ajout d'un cycle supplémentaire.* $\mathbf{F'}_\delta$ *se calcule dans ce cas en* 4 *cycles en tout.*

En Section 4.2.2, nous avons proposé d'implémenter la fonction de Dirac δ par une table de correspondance. Cette dernière stocke 2^n+2 bits en mémoire, soit 258 dans le contexte de l'`AES` et s'exécute une seule fois et en 1 cycle. En comparaison avec cette construction, l'implémentation à seuil $\mathbf{F}_\delta$ de δ présente l'avantage de ne pas être pré-traitée ni d'allouer de l'espace en mémoire. Cependant, elle implique l'utilisation de 4 cycles (1 cycle pour la transformation en Figure 34 et 3 autres pour implémenter $\mathbf{F}_\delta$) afin d'être glitches-immune. En fonction des critères de performances à optimiser, un développeur peut choisir l'une ou l'autre construction.

Dans la section suivante, nous voyons quel rôle joue la fonction de Dirac dans notre proposition de calcul sécurisé de la fonction $\mathrm{F}_{pow}(x) = x^q$.

4.3 Notre évaluation monomiale sécurisée à l'ordre 1 en présence de glitches

Dans cette partie nous proposons une réalisation 3-masquée $\mathbf{F}_{pow}$ de la fonction $\mathrm{F}_{pow}(x) = x^q$ en 4 cycles. Notre construction est illustrée en Figure 36 et suit les étapes suivantes :

Étape 1 : Notre implémentation prend en entrée un partage affine de x (voir Définition 22) et un partage additif en 2-morceaux (δ_1, δ_2) de $\delta(x)$. Ce dernier peut être calculé par une table de correspondance (voir Section 4.2.2) ou à travers 7 multiplications sécurisées (voir Section 4.2.3). Nous définissons le partage en 5-morceaux $\mathbf{x_1} = ((x \oplus \beta) \otimes \alpha, \alpha, \beta, \delta_1, \delta_2)$, pour $\beta \in \mathrm{GF}(2^n)$ et $\alpha \in \mathrm{GF}(2^n)^\star$ et tel que $\mathbf{x_1}$ est associé à la fonction de reconstruction $rec(\mathbf{x_1}) = ((x \oplus \beta) \otimes \alpha \otimes \alpha^{-1} \oplus \beta, \delta_1 \oplus \delta_2) = (x, \delta(x))$. En sortie, nous souhaitons obtenir un nouveau partage en 5-morceaux, noté $\mathbf{x}_2$, de x^q.

Étape 2 : Nous souhaitons calculer $\mathrm{F}_{pow}(x)$ de manière sécurisée. Précédemment, nous avons vu en Section 2.4.2 qu'il est plus approprié d'utiliser un masquage multiplicatif plutôt qu'un masquage additif afin de protéger la fonction (non-linéaire) F_{pow}. De plus, afin d'éviter un biais statistique induit par le problème du zéro (voir Section 4.1), la valeur sensible x est associée à sa fonction de Dirac $\delta(x)$ telle que F_{pow} est appliquée à la variable $(x \oplus \delta(x))$ (au lieu de x). Cette deuxième étape consiste donc à transformer le partage $\mathbf{x_1}$ de $(x, \delta(x))$ en un partage en 5-morceaux $\mathbf{x_2}$ de x^q.

Le partage affine de x est tout d'abord converti en un partage multiplicatif en 2-morceaux de $(x^q \oplus \delta(x))$. La conversion est illustrée dans la partie supérieure de la Figure 36. Nous devons "injecter" les morceaux du partage de $\delta(x)$ (*i.e.* δ_1, δ_2) dans la construction tout en supprimant le masque additif β. Cela doit se réaliser en respectant les propriétés des implémentations à seuil. En particulier, afin de garantir la glitches-immunité à l'ordre 1, il est nécessaire de ne pas manipuler les deux morceaux de $\delta(x)$ dans un même cycle. Ainsi, nous obtenons en 2 cycles le partage multiplicatif en 2-morceaux $((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q)$ de $(x^q \oplus \delta(x))$. La fonction de reconstruction associée consiste à multiplier le premier morceaux par l'inverse du deuxième. Puis, nous étendons ce partage en ajoutant un troisième morceau $(\beta \otimes \alpha^q)$ que nous utiliserons dans l'étape suivante. Au final, nous obtenons le partage en 5-morceaux $\mathbf{x_2} = ((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, \delta_1, \delta_2)$ de x^q.

Étape 3 : Cette dernière étape permet de convertir $\mathbf{x_2}$ en un partage en 5-morceaux $\mathbf{x_3}$ constitué d'un partage affine de x^q et du partage additif de $\delta(x)$. Ajouté à β, nous utilisons ce dernier partage afin de fabriquer un nouveau masque additif β'. Comme pour la deuxième étape, le caractère de glitches-immunité à l'ordre 1 est obtenu en découpant le calcul en 2 cycles. La conversion est

illustrée dans la partie inférieure de la Figure 36. Nous obtenons finalement le nouveau partage $\mathbf{x_3} = ((x^q \oplus \beta') \otimes \alpha^q, \alpha^q, \beta', \delta_1, \delta_2)$ de $(x^q, \delta(x))$, où $\beta' = \delta(x) \oplus \beta$.

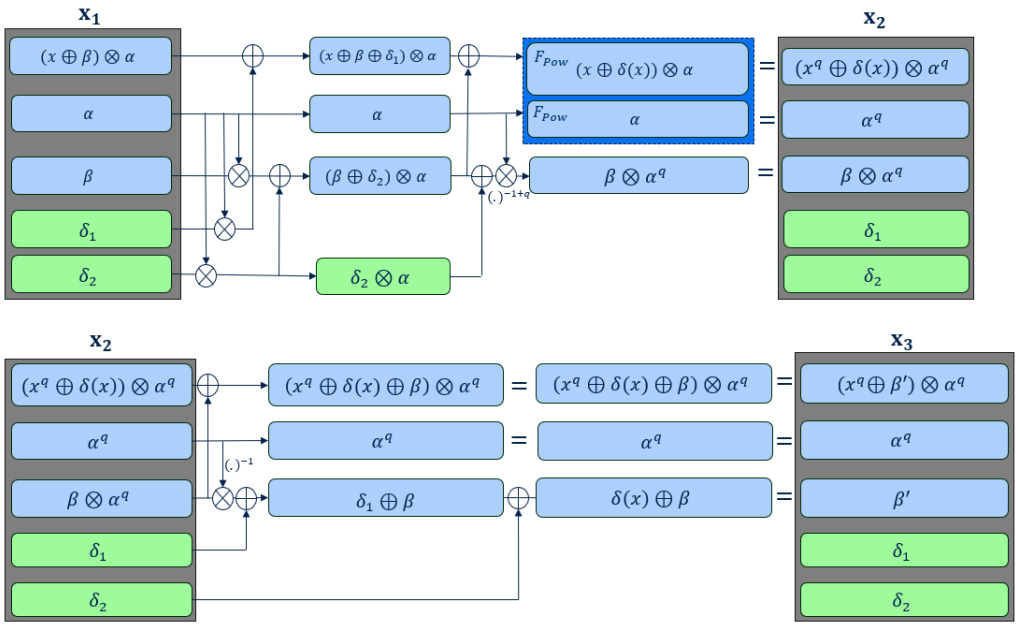


FIGURE 36 – Notre réalisation 3-masquée $\mathbf{F}_{pow}$ de la fonction $\mathrm{F}_{pow}(x) = x^q$ en 4 cycles

Remarque 8. *Afin de garantir l'uniformité dans notre construction, il est important que la variable de* α^q *reste uniformément distribuée dans* $\mathrm{GF}(2^n)^\star$ *quand* α *l'est. Cette condition ne peut s'appliquer que sur des fonctions* F_{pow} *bijective* i.e. *si et seulement si* q *est premier avec l'ordre du sous-groupe multiplicatif de* $\mathrm{GF}(2^n)$.

La sécurité de notre construction repose sur la proposition suivante.

Proposition 3. *Soient* α *et* β *deux variables indépendantes distribuées uniformément dans* $\mathrm{GF}(2^n)^\star$ *et* $\mathrm{GF}(2^n)$, *respectivement. Soit un entier strictement positif* q *tel que* $pgcd(q, 2^n - 1) = 1$. *Alors la réalisation* $\mathbf{F}_{pow}$ *de la fonction* $\mathrm{F}_{pow}(x) = x^q$ *décrite en Figure 36 est correcte, glitches-immune à l'ordre* 1 *et uniforme. C'est donc une implémentation à seuil de* F_{pow}, *sécurisée à l'ordre* 1 *en présence de glitches.*

Nous donnons la preuve de la Proposition 3 ci-dessous.

Preuve. *Nous prouvons ici que la réalisation* 3*-masquée* $\mathbf{F}_{pow}$ *de* $\mathrm{F}_{pow}(x) = x^q$ *décrite en Figure 36 est correcte, glitches-immune à l'ordre* 1 *et uniforme. Cette réalisation est définie par la composition de deux réalisations* 5*-masquées* $\mathbf{F}_{12}$ *et* $\mathbf{F}_{23}$ *telles que* $\mathbf{F}_{pow} = \mathbf{F}_{23} \circ \mathbf{F}_{12}$ *et :*

- $\mathbf{F}_{12} : \mathbf{x_1} \mapsto \mathbf{x_2}$ *implémente la fonction* $\mathrm{F} : (x, \delta(x)) \mapsto x^q \oplus \delta(x) \oplus \delta(x) = x^q$
- $\mathbf{F}_{23} : \mathbf{x_2} \mapsto \mathbf{x_3}$ *implémente la fonction* $\mathrm{F} : (x^q \oplus \delta(x), \delta(x)) \mapsto x^q \oplus \delta(x) \oplus \delta(x) = x^q$

Dans un premier temps, nous démontrons les propriétés de $\mathbf{F}_{12}$.

Correction : *La fonction de reconstruction de sortie de* $\mathbf{x_2} = ((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, \delta_1, \delta_2)$ *existe et est définie par :*

$$rec(\mathbf{x_2}) = (x^q \oplus \delta(x)) \otimes \alpha^q \otimes (\alpha^q)^{-1} \oplus \delta_1 \oplus \delta_2 = x^q \oplus \delta(x) \oplus \delta(x) = x^q$$

La Définition 19 étant vérifiée, la réalisation 3*-masquée* $\mathbf{F}_{12}$ *est correcte.*

Glitches-immunité à l'ordre 1 *: Dans la partie supérieure de la Figure 36, pour les deux cycles, chaque morceau de sortie est fonctionnellement indépendant d'au moins un morceau d'entrée du cycle précédent. La Définition 20 étant vérifiée, la réalisation* 3*-masquée* $\mathbf{F}_{12}$ *est glitches-immune à l'ordre* 1*.*

Uniformité : D'après la Définition 21, nous devons vérifier que le nombre d'antécédents de la forme $\mathbf{x}_1 \in \mathrm{GF}(2)^{5\times n}$ *pour chaque image* $\mathbf{x}_2 \in \mathrm{GF}(2)^{5\times m}$ *est égal à* $2^{(5-1)\times(n-m)}$ *fois le nombre d'antécédents* $x \in \mathrm{GF}(2^n)$ *pour chaque image* $\mathrm{F}(x,\delta(x)) = x^q$*. Comme* $n = m$*, ce nombre est égal à* $2^{2\times 0} = 1$*. Nous rappelons que par définition,* $x^q \oplus \delta(x) \neq 0$*. Nous avons :*

- *Si l'image* $\mathrm{F}(x,\delta(x)) = x^q = 0$ *il existe* 1 *antécédent* $(x,\delta(x)) = (0,1)$*. Pour* $\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star, \delta_1 \in \{0,1\}$ *et* $\delta_2 = \delta_1 \oplus 1$*, le Tableau 5 indique le nombre d'antécédents de la forme* $\mathbf{x}_1 = ((x\oplus\beta)\otimes\alpha,\alpha,\beta,\delta_1,\delta_2)$ *telle que* $rec(\mathbf{x}_2) = 0$*. On observe qu'il existe* 1 *unique antécédent* $\mathbf{x}_1$ *pour chaque image* $\mathbf{x}_2$*.*

Antécédent $\mathbf{x}_1 = ((x\oplus\beta)\otimes\alpha,\alpha,\beta,\delta_1,\delta_2)$	Image $\mathbf{x}_2 = ((x^q\oplus\delta(x))\otimes\alpha^q,\alpha^q,\beta\otimes\alpha^q,\delta_1,\delta_2)$	
	$rec(\mathbf{x}_2) = 0 \Rightarrow x = 0$	
$\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star$ $\delta_1 \in \{0,1\}, \delta_2 = \delta_1 \oplus 1$	$(\alpha^q,\alpha^q,\beta\otimes\alpha^q,0,1)$	$(\alpha^q,\alpha^q,\beta\otimes\alpha^q,1,0)$
$x = 0 \Rightarrow (\beta\otimes\alpha,\alpha,\beta,0,1)$	1	0
$x = 0 \Rightarrow (\beta\otimes\alpha,\alpha,\beta,1,0)$	0	1

TABLEAU 5 – Nombre d'antécédents $\mathbf{x}_1$ pour toute image $\mathbf{x}_2$ telle que $rec(\mathbf{x}_2) = 0$

- *Si l'image* $\mathrm{F}(x,\delta(x)) = x^q \in \mathrm{GF}(2^n)^\star$*, il existe* 1 *antécédent* $(x,0)$ *tel que* $x \in \mathrm{GF}(2^n)^\star$*. Pour* $\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star, \delta_1 \in \{0,1\}$ *et* $\delta_2 = \delta_1$*, le Tableau 6 indique le nombre d'antécédents de la forme* $\mathbf{x}_1 = ((x\oplus\beta)\otimes\alpha,\alpha,\beta,\delta_1,\delta_2)$ *telle que* $rec(\mathbf{x}_2) = x^q \in \mathrm{GF}(2^n)^\star$*. On observe qu'il existe* 1 *unique antécédent* $\mathbf{x}_1$ *pour chaque image* $\mathbf{x}_2$*.*

Antécédent $\mathbf{x}_1 = ((x\oplus\beta)\otimes\alpha,\alpha,\beta,\delta_1,\delta_2)$	Image $\mathbf{x}_2 = ((x^q\oplus\delta(x))\otimes\alpha^q,\alpha^q,\beta\otimes\alpha^q,\delta_1,\delta_2)$	
	$rec(\mathbf{x}_2) = x^q \in \mathrm{GF}(2^n)^\star \Rightarrow x \in \mathrm{GF}(2^n)^\star$	
$\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star$ $\delta_1 \in \{0,1\}, \delta_2 = \delta_1$	$(x^q\otimes\alpha^q,\alpha^q,\beta\otimes\alpha^q,0,0)$	$(x^q\otimes\alpha^q,\alpha^q,\beta\otimes\alpha^q,1,1)$
$x \in \mathrm{GF}(2^n) \Rightarrow ((x\oplus\beta)\otimes\alpha,\alpha,\beta,0,0)$	1	0
$x \in \mathrm{GF}(2^n) \Rightarrow ((x\oplus\beta)\otimes\alpha,\alpha,\beta,1,1)$	0	1

TABLEAU 6 – Nombre d'antécédents $\mathbf{x}_1$ pour toute image $\mathbf{x}_2$ telle que $rec(\mathbf{x}_2) = x^q \in \mathrm{GF}(2^n)^\star$

La Définition 21 étant vérifiée, la réalisation 3*-masquée* $\mathbf{F}_{12}$ *est uniforme.*

La réalisation $\mathbf{F}_{12}$ *est donc une implémentation à seuil de la fonction* $\mathrm{F}(x,\delta(x)) = x^q$*. Dans un second temps, nous démontrons ci-dessous les propriétés de* $\mathbf{F}_{23}$*.*

Correction : La fonction de reconstruction de sortie de $\mathbf{x_3} = ((x^q\oplus\beta')\otimes\alpha^q,\alpha^q,\beta',\delta_1,\delta_2)$ *existe et est définie par :*

$$rec(\mathbf{x_3}) = (x^q\oplus\beta')\otimes\alpha^q\otimes(\alpha^q)^{-1}\oplus\beta' = x^q$$

La Définition 19 étant vérifiée, la réalisation 3*-masquée* $\mathbf{F}_{23}$ *est correcte.*

Glitches-immunité à l'ordre 1 *: Dans la partie inférieure de la Figure 36, pour les deux cycles, chaque morceau de sortie est fonctionnellement indépendant d'au moins un morceau d'entrée du cycle précédent. La Définition 20 étant vérifiée, la réalisation* 3*-masquée* $\mathbf{F}_{23}$ *est glitches-immune à l'ordre* 1*.*

Uniformité : D'après la Définition 21, nous devons vérifier que le nombre d'antécédents de la forme $\mathbf{x}_2 \in \mathrm{GF}(2)^{5\times n}$ *pour chaque image* $\mathbf{x}_3 \in \mathrm{GF}(2)^{5\times m}$ *est égal à* $2^{(5-1)\times(n-m)}$ *fois le nombre d'antécédents* $(x^q \oplus \delta(x), \delta(x))$ *pour chaque image* $\mathrm{F}(x^q \oplus \delta(x), \delta(x)) = x^q$. *Comme* $n = m$, *ce nombre est égal à* $2^{2\times 0} = 1$. *Par définition,* $x^q \oplus \delta(x) \neq 0$. *Nous avons :*

- *Si l'image* $\mathrm{F}(x^q \oplus \delta(x), \delta(x)) = x^q = 0$, *il existe* 1 *unique antécédent* $(x^q \oplus \delta(x), \delta(x)) = (1,1)$ (i.e. *où* $x = 0$). *Pour* $\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star, \delta_1 \in \{0,1\}$ *et* $\delta_2 = \delta_1 \oplus 1$, *le Tableau 7 indique le nombre d'antécédents de la forme* $\mathbf{x}_2 = ((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, \delta_1, \delta_2)$ *telle que* $rec(\mathbf{x}_3) = 0$. *On observe qu'il existe* 1 *unique antécédent* $\mathbf{x}_2$ *pour chaque image* $\mathbf{x}_3$.

Antécédent $\mathbf{x}_2 = ((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, \delta_1, \delta_2)$	Image $\mathbf{x}_3 = ((x^q \oplus \delta(x) \oplus \beta) \otimes \alpha^q, \alpha^q, \delta(x) \oplus \beta, \delta_1, \delta_2)$	
	$rec(\mathbf{x}_3) = 0 \Rightarrow x = 0$	
$\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star$ $\delta_1 \in \{0,1\}, \delta_2 = \delta_1 \oplus 1$	$((1 \oplus \beta) \otimes \alpha^q, \alpha^q, 1 \oplus \beta, 0, 1)$	$((1 \oplus \beta) \otimes \alpha^q, \alpha^q, 1 \oplus \beta, 1, 0)$
$x = 0 \Rightarrow (\alpha^q, \alpha^q, \beta \otimes \alpha^q, 0, 1)$	1	0
$x = 0 \Rightarrow (\alpha^q, \alpha^q, \beta \otimes \alpha^q, 1, 0)$	0	1

TABLEAU 7 – Nombre d'antécédents $\mathbf{x}_2$ pour chaque image $\mathbf{x}_3$ telle que $rec(\mathbf{x}_3) = 0$

- *Si l'image* $\mathrm{F}(x^q \oplus \delta(x), \delta(x)) = x^q \in \mathrm{GF}(2^n)^\star$, *il existe* 1 *unique antécédent* $(x^q \oplus \delta(x), \delta(x)) = (x^q, 0)$ (i.e. *où* $x \in \mathrm{GF}(2^n)^\star$). *Pour* $\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star, \delta_1 \in \{0,1\}$ *et* $\delta_2 = \delta_1$, *le Tableau 8 indique le nombre d'antécédents de la forme* $\mathbf{x}_2 = ((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, \delta_1, \delta_2)$ *telle que* $rec(\mathbf{x}_3) = x^q \in \mathrm{GF}(2^n)^\star$. *On observe qu'il existe* 1 *unique antécédent* $\mathbf{x}_2$ *pour chaque image* $\mathbf{x}_3$.

Antécédent $\mathbf{x}_2 = ((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, \delta_1, \delta_2)$	Image $\mathbf{x}_3 = ((x^q \oplus \delta(x) \oplus \beta) \otimes \alpha^q, \alpha^q, \delta(x) \oplus \beta, \delta_1, \delta_2)$	
	$rec(\mathbf{x}_3) = x^q \in \mathrm{GF}(2^n)^\star \Rightarrow x \in \mathrm{GF}(2^n)^\star$	
$\beta \in \mathrm{GF}(2^n), \alpha \in \mathrm{GF}(2^n)^\star$ $\delta_1 \in \{0,1\}, \delta_2 = \delta_1$	$((x^q \oplus \beta) \otimes \alpha^q, \alpha^q, \beta, 0, 0)$	$((x^q \oplus \beta) \otimes \alpha^q, \alpha^q, \beta, 1, 1)$
$x \in \mathrm{GF}(2^n)^\star \Rightarrow (x^q \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, 0, 0)$	1	0
$x \in \mathrm{GF}(2^n)^\star \Rightarrow (x^q \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q, 1, 1)$	0	1

TABLEAU 8 – Nombre d'antécédents $\mathbf{x}_2$ pour chaque image $\mathbf{x}_3$ telle que $rec(\mathbf{x}_3) = x^q \in \mathrm{GF}(2^n)^\star$

La Définition 21 étant vérifiée, la réalisation 3*-masquée* $\mathbf{F}_{23}$ *est uniforme.*

La réalisation 3*-masquée* $\mathbf{F}_{23}$ *est donc une implémentation à seuil de la fonction* $\mathrm{F}(x \oplus \delta(x), \delta(x)) = x^q$.

Comme la réalisation 3*-masquée* $\mathbf{F}_{pow}$ *est composée des deux implémentations à seuil* $\mathbf{F}_{23}$ *et* $\mathbf{F}_{12}$, *d'après la Propriété 3,* $\mathbf{F}_{pow}$ *est une implémentation à seuil. Elle permet de calculer de la fonction* $\mathrm{F}_{pow}(x) = x^q$ *et est sécurisée à l'ordre* 1 *en présence de glitches.*

□

Remarque 9. *Notre évaluation sécurisée de* F_{pow} *présente l'avantage de ne pas avoir besoin de sécuriser le calcul de la puissance (*i.e. x^q*). Le fait d'utiliser une implémentation à seuil de* F_{pow} *est suffisant pour garantir la sécurité, car la donnée* x^q *est toujours masquée (*i.e. *par un partage multiplicatif dans* $\mathbf{x}_2$ *et par un partage affine dans* $\mathbf{x}_3$*).*

Dans la section suivante, nous appliquons notre construction sécurisée de $\mathrm{F}_{pow}(x)$ sur la fonction inverse de l'AES, soit la fonction $\mathrm{INV}_{\mathrm{AES}}(x) = x^{254} \in \mathrm{GF}(2^8)$ présentée en Équation 2.2. De plus, nous proposons une implémentation à seuil pour chaque opération de l'AES présentée en Section 2.1.3.

4.4 Illustration de notre construction sur l'AES

Nous nous référons à la Section 2.1.3 pour une description complète de l'AES. Nous notons ARK, SB, SR et MC les opérations *AddRoundKey*, *SubBytes*, *ShiftRows* et *MixColumns*, respectivement. Nous appliquons notre stratégie de masquage présentée en Section 4.1 afin de protéger efficacement la valeur sensible à travers ces opérations. Dans la Figure 37 nous représentons la structure de notre implémentation à seuil d'un tour d'AES. Les variables $\mathbf{x}$ et $\mathbf{rk}$ sont les partages affines de l'état de données manipulé (pour le premier tour il s'agit du message clair x) et de la clé de tour rk correspondante, respectivement, et les éléments $\mathbf{x}_1, \mathbf{x}'_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ correspondent aux partages intermédiaires entre les opérations d'AES. Les fonctions $\mathrm{INV}_{\mathrm{AES}}$ et $\mathrm{AF}_{\mathrm{AES}}$ sont détaillées dans l'Équation 2.2 et permettent de calculer les 16 boîtes-S de l'opération *SubBytes*. Notre évaluation monomiale décrite en Section 4.3 est notamment illustrée sur la fonction $\mathrm{INV}_{\mathrm{AES}}$.

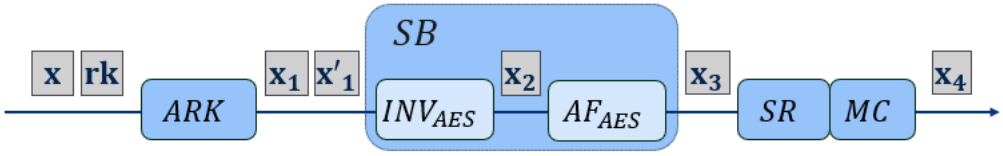


FIGURE 37 – Structure de notre implémentation à seuil d'un tour d'AES

Dans la suite, nous décrivons la façon dont les opérations de l'AES sont implémentées pour traiter les différents partages en 3-morceaux avec une sécurité à l'ordre 1 en présence de glitches.

4.4.1 Notre implémentation à seuil d'un chiffrement par bloc de l'AES

Pour simplifier notre description, nous détaillons les implémentations à seuil des étapes de l'AES pour un message et la clé de tour correspondante en entrée, tous deux constitués de 16 éléments dans $\mathrm{GF}(2^8)$.

Initialisation. Soit x un octet du message et soit rk l'octet correspondant (i.e. de même indice) de la première clé de tour. On note respectivement β_x et β_{rk} leurs masques additifs, uniformément distribués dans $\mathrm{GF}(2^8)$ et indépendants l'un de l'autre. Soit également $\alpha \in \mathrm{GF}(2^8)^\star$ un masque multiplicatif. Les octets x et rk sont représentés par leurs partages affines en 3-morceaux $\mathbf{x} = ((x \oplus \beta_x) \otimes \alpha, \alpha, \beta_x)$ et $\mathbf{rk} = ((rk \oplus \beta_{rk}) \otimes \alpha, \alpha, \beta_{rk})$, respectivement. Dans notre proposition, tous les partages en 3-morceaux manipulent un même masque multiplicatif. Ce n'est pas le cas du masque additif que nous faisons évoluer à chaque masquage d'un état d'AES. Nous décrivons notre implémentation sécurisée pour chaque opération de l'AES pour un tour ci-dessous.

Étape 1 : *AddRoundKey*. L'opération ARK est sécurisée en ajoutant les premiers (respectivement les troisièmes) morceaux de $\mathbf{x}$ et $\mathbf{rk}$ ensemble. Nous obtenons en 1 cycle le partage affine $\mathbf{x_1} = ((x \oplus rk \oplus \beta) \otimes \alpha, \alpha, \beta)$, où $\beta = \beta_x \oplus \beta_{rk}$. Cette transformation est appliquée aux 16 octets de l'état d'AES et peut être parallélisée car β_x et β_{rk} sont indépendants.

Proposition 4. *La réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{ARK}} : (\mathbf{x}, \mathbf{rk}) \mapsto \mathbf{x_1}$ *de la fonction* $\mathrm{F} : (x, rk) \in \mathrm{GF}(2^n)^2 \mapsto x \oplus rk$ *est correcte, glitches-immune à l'ordre* 1 *et uniforme. Elle est sécurisée à l'ordre* 1 *en présence de glitches.*

Preuve. *Nous démontrons ci-dessous les propriétés de la réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{ARK}}$.

Correction : La fonction de reconstruction de sortie de $\mathbf{x_1}$ *existe et est définie par :*

$$rec_{aff}(\mathbf{x_1}) = ((x \oplus rk \oplus \beta) \otimes \alpha \otimes \alpha^{-1} \oplus \beta = x \oplus rk$$

La Définition 19 étant vérifiée, la réalisation 3*-masquée* $\mathbf{F}_{\mathrm{ARK}}$ *est correcte.*

Glitches-immunité à l'ordre 1 *: Le calcul du premier (respectivement du troisième) morceau de* $\mathbf{x_1}$ *en sortie est indépendant fonctionnellement du troisième (respectivement du premier) morceau de* $\mathbf{x}$ *et* $\mathbf{rk}$ *en entrée. Le deuxième morceau de* $\mathbf{x_1}$ *est une copie directe du deuxième morceau de* $\mathbf{x}$*. La Définition 20 étant vérifiée, la réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{ARK}}$ *est glitches-immune à l'ordre* 1*.*

Uniformité : Comme β_x *et* β_{rk} *sont uniformément distribués dans* $\mathrm{GF}(2^8)$ *et indépendants l'un de l'autre, le caractère d'uniformité est une conséquence directe de la Propriété 1.*

La réalisation 3*-masquée* $\mathbf{F}_{\mathrm{ARK}}$ *est donc correcte, glitches-immune à l'ordre* 1 *et uniforme. C'est une implémentation à seuil de la fonction* ARK*, sécurisée à l'ordre* 1 *en présence de glitches.*

□

Étape 2 : ***SubBytes.*** L'opération SB consiste en l'application de la boîte-S de l'AES sur chacun des octets d'un état d'AES après l'opération ARK (*i.e.* de la forme $(x \oplus rk)$). Cette boîte-S, notée $\mathrm{SB}ox$, est définie en Équation 2.2 par la fonction inverse $\mathrm{INV}_{\mathrm{AES}}(x) = x^{254} \in \mathrm{GF}(2^8)$ et la fonction affine $\mathrm{AF}_{\mathrm{AES}}$. Comme 254 est premier avec $2^8 - 1$ (*i.e.* l'ordre du sous-groupe multiplicatif de $\mathrm{GF}(2^8)$), on peut appliquer la Proposition 3. On considère alors notre construction d'implémentation à seuil de la fonction $\mathrm{F}_{pow} = x^{254}$ présentée en Section 4.3. Pour plus de lisibilité, nous notons $q = -1$ dans la suite. Notre réalisation 3-masquée de la boîte-S de l'AES sur l'octet $(x \oplus rk)$ est construite par l'application de la Figure 36 sur la fonction $\mathrm{INV}_{\mathrm{AES}}$ dans laquelle nous intégrons le calcul sécurisé de la fonction $\mathrm{AF}_{\mathrm{AES}}$. Un partage additif en 2-morceaux (δ_1, δ_2) de la fonction de Dirac de $(x \oplus rk)$ est créé comme présenté en Sections 4.2.2. Notre réalisation prend alors en entrée un partage en 5-morceaux $\mathbf{x}'_1 = ((x \oplus rk \oplus \beta) \otimes \alpha, \alpha, \beta, \delta_1, \delta_2)$ constitué du partage affine $\mathbf{x}_1$ de $(x \oplus rk)$ et du partage additif de $\delta(x \oplus rk)$. Notre construction est illustrée en Figure 38.

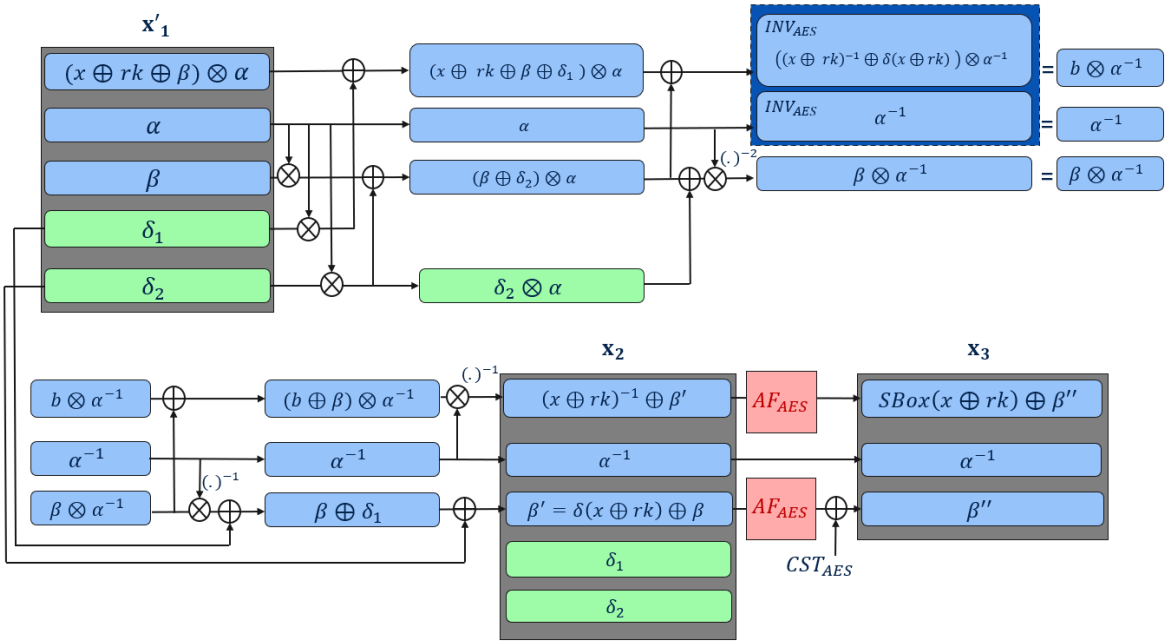


FIGURE 38 – Adaptation de notre évaluation monomiale sur la fonction $\mathrm{INV}_{\mathrm{AES}}(x) = x^{254}$ suivie par le calcul sécurisé de la fonction $\mathrm{AF}_{\mathrm{AES}}$ - Illustration de notre implémentation à seuil de la boîte-S de l'AES en 5 cycles

Dans un premier temps, en adaptant la description de la partie supérieure de la Figure 36, nous obtenons le partage additif en 3-morceaux $((x \oplus rk)^{-1} \oplus \beta', \alpha^{-1}, \beta')$ de $\mathrm{INV}_{\mathrm{AES}}(x \oplus rk)$, où $\beta' = \delta(x \oplus rk) \oplus \beta$. La fonction de reconstruction de sortie associée à ce partage additif consiste en la somme entre

le premier et le troisième morceau. Nous parlons d'adaptation de notre proposition car le masque multiplicatif α^{-1} a été retiré du premier morceau. En effet, un masquage additif seulement de la valeur sensible est nécessaire dans le but de protéger la fonction affine $\mathrm{AF}_{\mathrm{AES}}$ qui suit. Nous évitons ainsi une conversion de notre partage affine en un partage additif plus tard. Nous obtenons le partage additif en 5-morceaux $\mathbf{x}_2 = ((x \oplus rk)^{-1} \oplus \beta', \alpha^{-1}, \beta', \delta_1, \delta_2)$ de $(x \oplus rk)^{-1}$. La fonction de reconstruction de sortie de $\mathbf{x}_2$ correspond à la somme du premier et du troisième morceau. Puis, le calcul sécurisé de $\mathrm{AF}_{\mathrm{AES}}$ nous conduit au partage additif en 3-morceaux $\mathbf{x}_3 = (\mathrm{SB}ox(x \oplus rk) \oplus \beta'', \alpha^{-1}, \beta'')$ de $\mathrm{SB}ox(x \oplus rk)$, où $\beta'' = \mathrm{AF}_{\mathrm{AES}}(\beta') \oplus \mathrm{CST}_{\mathrm{AES}} = \mathscr{M}_{\mathrm{AES}}(\beta')$. Ce calcul peut être fusionné avec le $4^{ème}$ cycle de la construction de $\mathbf{x}_2$ sans introduire de vulnérabilité dans l'implémentation car $\mathrm{AF}_{\mathrm{AES}}$ est une bijection appliquée séparément sur le premier et troisième morceau de $\mathbf{x}_2$. La fonction de reconstruction de sortie associée à $\mathbf{x}_3$ consiste en la somme entre le premier et le troisième morceau. Au final, la réalisation 3-masquée de la boîte-S de l'AES s'effectue en 5 cycles (4 pour $\mathrm{INV}_{\mathrm{AES}}(\mathrm{AF}_{\mathrm{AES}}(x \oplus rk))$ et 1 cycle pour la fonction de Dirac sous forme de table de correspondance).

Nous pouvons remarquer que la valeur du masque additif évolue entre chaque étape. Les masques additifs β, β' et β'' de $\mathbf{x}'_1, \mathbf{x}_2$ et $\mathbf{x}_3$, respectivement, sont différents. Cette évolution présente une sécurité supplémentaire de notre construction.

Remarque 10. *Si nous implémentons le partage additif en* 3*-morceaux* $\mathbf{x}_\delta$ *de la fonction de Dirac en suivant la technique décrite en Section 4.2.3, la réalisation* 3*-masquée de la boîte-S de l'AES s'effectue en* 8 *cycles (*4 *pour* $\mathrm{INV}_{\mathrm{AES}}(\mathrm{AF}_{\mathrm{AES}}(x \oplus rk))$ *plus* 4 *pour la fonction de Dirac en* 7 *multiplications sécurisées à l'ordre* 1 *en présence de glitches). Dans la Figure 38,* δ_1 *et* δ_2 *consisteront respectivement à la somme des deux premiers morceaux et au troisième morceau de* $\mathbf{F}_\delta$*, comme présenté en Remarque 7. En revanche, ce choix d'implémentation ne nécessite pas de pré-traitement ou de stockage supplémentaire.*

Proposition 5. *La réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{SB}} : \mathbf{x}'_1 \mapsto \mathbf{x}_3$ *de la fonction* $\mathrm{F} : x \mapsto \mathrm{SB}ox(x)$ *est correcte, glitches-immune à l'ordre* 1 *et uniforme. Elle est sécurisée à l'ordre* 1 *en présence de glitches.*

Preuve. *Soit la réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{INV}} : \mathbf{x}'_1 \mapsto \mathbf{x}_2$ *de la fonction* $\mathrm{F} : x \mapsto \mathrm{INV}_{\mathrm{AES}}(x)$*. Montrer que* $\mathbf{F}_{\mathrm{INV}}$ *est une possible implémentation à seuil de la fonction inverse de l'AES est une conséquence directe de la Proposition 3.*

Soit la réalisation 3*-masquée* $\mathbf{F}_{\mathrm{AF}} : \mathbf{x}_2 \mapsto \mathbf{x}_3$ *de la fonction* $\mathrm{F} : x \mapsto \mathrm{AF}_{\mathrm{AES}}(x)$*. La fonction de reconstruction de sortie de* $\mathbf{x}_3$ *est définie par l'addition du premier et du troisième morceau de* $\mathbf{x}_3$*. La réalisation* $\mathbf{F}_{\mathrm{AF}}$ *est donc correcte. De plus, la fonction affine* $\mathrm{AF}_{\mathrm{AES}}$ *est appliquée à au premier et deuxième morceau de* $\mathbf{x}_2$*, séparément, ce qui garantit la glitches-immunité à l'ordre* 1 *de* $\mathbf{F}_{\mathrm{AF}}$*. Enfin, comme* $\mathrm{AF}_{\mathrm{AES}}$ *est une bijection,* $\mathbf{F}_{\mathrm{AF}}$ *est uniforme. La réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{AF}}$ *est donc une implémentation à seuil de la fonction* $\mathrm{AF}_{\mathrm{AES}}$*.*

Comme la réalisation 3*-masquée* $\mathbf{F}_{\mathrm{SB}}$ *est définie par* $\mathbf{F}_{\mathrm{SB}} = \mathbf{F}_{\mathrm{AF}} \circ \mathbf{F}_{\mathrm{INV}}$*, d'après la Propriété 3,* $\mathbf{F}_{\mathrm{SB}}$ *est une implémentation à seuil de la boîte-S de l'AES, sécurisée à l'ordre* 1 *en présence de glitches.*

□

Étape 3 : *ShiftRows.* L'opération SR est illustrée en Figure 12. Elle consiste en un décalage cyclique vers la gauche de chacune des 4 lignes de l'état d'AES de 0, 1, 2 et 3 positions, respectivement. Pour notre implémentation sécurisée, cela implique simplement un ré-indexage des partages additifs en 3-morceaux (*i.e.* de la forme $\mathbf{x}_3$) de chaque élément de l'état d'AES. Cette étape ne nécessite pas de cycle supplémentaire et peut être fusionnée avec l'étape suivante sans induire de vulnérabilité.

Étape 4 : *MixColumns.* Nous considérerons ici une colonne de 4 octets de l'état d'AES en entrée de notre implémentation. L'opération MC consiste en la multiplication de la $i^{ème}$ colonne de l'état d'AES par la matrice Mat présentée en Figure 13. On note Mat_j la $j^{ème}$ ligne de Mat. Sans perte de généralité, nous décrivons notre réalisation 3-masquée du calcul de la première colonne de l'état

d'AES, notée $(\mathrm{SB}ox(x_j \oplus rk_j))_{j\in[0,3]}$ après l'application des opérations ARK, SB et SR. Dans notre implémentation, chaque octet $\mathrm{SB}ox(x_j \oplus rk_j)$ de la colonne est associé à son partage additif en 3-morceaux de la forme de $\mathbf{x_3}$, soit $(\mathrm{SB}ox(x_k \oplus rk_j) \oplus \beta''_j, \alpha^{-1}, \beta''_j)$. Ce dernier est donné en entrée à notre implémentation. De plus, comme la valeur sensible est masquée additivement, nous ignorons le masque multiplicatif α^{-1} dans le calcul de MC.

Notre proposition consiste donc simplement à multiplier la matrice Mat avec les vecteurs colonnes $(\mathrm{SB}ox(x_j \oplus rk_j) \oplus \beta''_j)_{j\in[0,3]}$ et $(\beta''_j)_{j\in[0,3]}$. Ce calcul nécessite un cycle supplémentaire. Soient $\odot$ le produit scalaire dans $\mathrm{GF}(2^8)^4$ et $(y_j)_{j\in[0,3]}$ le vecteur colonne obtenu à l'issu du calcul de MC sur la première colonne de l'état d'AES. Ainsi, chaque élément y_j pour $j \in [0,3]$ est représenté par le partage additif en 3-morceaux de la forme $((y_j \oplus \beta'''_j), \alpha^{-1}, \beta'''_j)$, où $y_j = Mat_j \odot (\mathrm{SB}ox(x_i \oplus rk_i) \oplus \beta''_i)^{\perp}_{i\in[0,3]}$ et $\beta'''_j = Mat_j \odot (\beta''_i)^{\perp}_{i\in[0,3]}$, où $(v)^{\perp}$ représente la transposée du vecteur v. La fonction de reconstruction de sortie associée à ce partage consiste en la somme entre le premier et le troisième morceaux. Enfin, afin de préparer la prochaine opération de l'AES, à savoir ARK, il est nécessaire de transformer ce partage en un partage affine. Cette conversion s'effectue en 1 cycle en multipliant le premier et le deuxième morceaux ensemble. Nous obtenons le partage affine $\mathbf{x_4} = ((y_j \oplus \beta'''_j) \otimes \alpha^{-1}, \alpha^{-1}, \beta'''_j)$ de la sortie de fonction MC. La fonction de reconstruction de sortie est définie par $rec_{aff}(\mathbf{x_4}) = (y_j \oplus \beta'''_j) \otimes \alpha^{-1} \otimes \alpha^{-1} \oplus \beta'''_j = \mathrm{MC}(\mathrm{SB}ox(x_j \oplus rk_j))$.

Proposition 6. *La réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{MC}} : \mathbf{x_3} \mapsto \mathbf{x_4}$ *de la fonction* $\mathrm{F} : x \mapsto \mathrm{MC}(x)$ *est correcte, glitches-immune à l'ordre* 1 *et uniforme. Elle est sécurisée à l'ordre* 1 *en présence de glitches.*

Preuve. *La propriété de correction et d'uniformité de la réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{MC}}$ *est une conséquence directe de la linéarité dans* $\mathrm{GF}(2^8)$ *de la transformation définie par le produit matriciel de* Mat*. De plus, la construction du premier (respectivement du troisième) morceau de* $\mathbf{x_4}$ *en sortie est indépendant fonctionnellement du troisième (respectivement du premier) morceau de* $\mathbf{x_3}$ *en entrée. Le deuxième morceau de* $\mathbf{x_4}$ *est une copie directe du deuxième morceau de* $\mathbf{x_3}$*. La réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{MC}}$ *est donc glitches-immune à l'ordre* 1*. Pour conclure, la réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{MC}}$ *est une implémentation à seuil de la fonction* MC*, sécurisée à l'ordre* 1 *en présence de glitches.*

□

Notre implémentation à seuil de tous les tours de l'AES fonctionne de manière similaire aux étapes 1 à 4, à part pour le dernier tour qui omet l'opération MC. Les octets de clé de tour sont représentés par un partage affine dont le masque multiplicatif est égal à celui du partage affine de l'octet sensible manipulé pour le tour. En effet, nous avons vu au début du premier tour que le masque multiplicatif impliqué dans **rk** est égal à α. À la fin de ce premier tour, le masque multiplicatif a évolué en la valeur α^{-1}. Nous avons ainsi la remarque suivante.

Remarque 11. *Pour chaque tour impair, le masque multiplicatif sera égal à* α *alors que pour chaque tour pair il sera égal à* α^{-1}*.*

Pour conclure cette partie, d'après les Propositions 4, 5 et 6, notre proposition d'implémentation à seuil de l'exécution de tout l'AES est sécurisée à l'ordre 1 en présence de glitches. Notre construction se fait en 7 cycles comme détaillé dans le Tableau 9.

Fonctions	Nombre de cycles
ARK	1
SB	16×5 (voir Remarque 12)
SR et MC	1

TABLEAU 9 – Nombre de cycle pour chaque opération d'un tour de chiffrement de l'AES

Remarque 12. *Les* 5 *cycles de l'opération SubBytes s'exécutent pour chacun des* 16 *octets en parallèle.*

Notre implémentation à seuil de la génération de chaque clé de tour est détaillée dans la suite.

4.4.2 Notre implémentation à seuil de l'algorithme de cadencement de clé de l'AES

Notre construction prend en entrée la clé cryptographique originale k. Nous nous référons à l'Algorithme 1 pour les détails du fonctionnement de l'algorithme de cadencement de clé de l'AES. En particulier, nous rappelons que pour $i \in [0,10]$, les clés de tour rk_i sont égales à la concaténation de quatre mots w de 4 octets telles que :

$$rk_i = w[4\times i] \| w[4\times i+1] \| w[4\times i+2] \| w[4\times i+3],$$

où la première clé de tour rk_0 est égale à la clé cryptographique originale k de l'AES telle que

$$rk_0 = k = w[0] \| w[1] \| w[2] \| w[3],$$

et le mot $w[j]$ est défini pour $j \in [4,43]$ tel que :

$$w[j] = \begin{cases} w[j-4] \oplus \mathrm{SW}(\mathrm{RW}(w[j-1])) \oplus \mathrm{R}con[j/4] & \text{si } j = 0 \mod 4 \\ w[j-4] \oplus w[j-1] & \text{sinon.} \end{cases} \tag{4.2}$$

Les fonctions *SubWord* et *RotWord* sont définies dans la Section 2.1.3 et sont représentées ci-dessus par les notations SW et RW, respectivement. Le tableau Rcon de 10 constantes est défini dans l'Algorithme 1.

Sans perte de généralité, nous présentons notre implémentation à seuil du calcul de la clé de tour $rk_1 = w[4] \| w[5] \| w[6] \| w[7]$. Les octets des mots $w[0]$, $w[1]$, $w[2]$ et $w[3]$ sont représentés par un partage affine avec le même masque multiplicatif $\alpha \in \mathrm{GF}(2^8)^\star$ et 4 masques additif indépendants et aléatoirement distribués dans $\mathrm{GF}(2^8)$.

Calcul de $w[4]$. Pour cette partie, nous nous concentrons sur la première ligne de l'Équation 4.2 pour $j = 4$, constituée des fonctions RW, SW et de deux Xors. Nous détaillons les calculs de ces opérations ci-dessous.

Étape 1 : *RotWord.* Comme *ShiftRows*, cette opération est un simple ré-indexage des partages affine en 3-morceaux du mot $w[3]$. Cette étape ne nécessite pas de cycle.

Étape 2 : *SubWord.* Nous appliquons ici notre implémentation à seuil de la boîte-S de l'AES, illustrée en Figure 38 et présentée dans l'étape 2 de la Section 4.4.1, sur chaque partage affine des octets de $\mathrm{RW}(w[3])$ et le partage additif de la fonction de Dirac associée. Le mot $\mathrm{SW}(\mathrm{RW}(w[3]))$ est composé de 4 octets notés x_i, pour $i \in [0,3]$. Après cette deuxième étape, chaque octet x_i est représenté par un partage affine de la forme $\mathbf{x}_i = ((x_i \oplus \beta_i) \otimes \alpha^{-1}, \alpha^{-1}, \beta_i)$. Nous utilisons la valeur inverse du masque multiplicatif car nous souhaitons que la clé de tour rk_1 (tour impair) ait un masque multiplicatif égal à α^{-1} (voir Remarque 11). Il est donc nécessaire de stocker les valeurs de α et α^{-1} afin d'utiliser le bon masque multiplicatif en fonction du tour.

Étape 3 : *Addition avec* Rcon[0]. Le mot Rcon[0] est public et composé de 4 octets notés Rcon_i[0] pour $i \in [0,3]$. Nous n'avons donc pas besoin de protéger cette étape. Nous multiplions chaque octet de Rcon[0] par α^{-1} et nous ajoutons chacun des résultats au premier morceau des partages affines $\mathbf{x}_i$. Après cette troisième étape, chaque octet de $\mathrm{SW}(\mathrm{RW}(w[3])) \oplus \mathrm{R}con[0]$ est représenté par un partage affine de la forme $\mathbf{x}_i = ((x_i \oplus \mathrm{R}con_i[0] \oplus \beta_i) \otimes \alpha^{-1}, \alpha^{-1}, \beta_i)$.

Étape 4 : *Addition avec* $w[0]$. Cette opération est implémentée de manière similaire à l'opération ARK dans l'étape 1 de la Section 4.4.1, avec la seule différence que le masque multiplicatif α de $w[0]$ doit être remplacé par α^{-1}. Pour cela, nous effectuons en 1 cycle la multiplication des premiers et deuxièmes morceaux des partages affines des octets de $w[0]$ par la valeur $(\alpha^{-1})^2$. Après cela, les premiers et troisièmes morceaux des partages affines en 3-morceaux de chaque octet de $w[0]$ sont respectivement ajoutés aux premiers et troisièmes morceaux des partages affines de chaque octet de $\mathrm{SW}(\mathrm{RW}(w[3])) \oplus \mathrm{R}con[0]$. Le résultat est un partage affine de chaque octet de $w[4]$. Soient y_i un octet de $w[0]$ et $\mathbf{y}_i = ((y_i \oplus \beta'_i) \otimes \alpha, \alpha, \beta'_i)$ le partage affine de y_i pour $i \in [0,3]$. Nous illustrons cette étape dans la Figure 39 pour les premiers octets de $w[0]$ (*i.e.* y_0) et $\mathrm{SW}(\mathrm{RW}(w[3])) \oplus \mathrm{R}con[0]$ (*i.e.* x_0), conduisant au partage affine du premier octet de $w[4]$. De manière plus générale, pour $i \in [0,3]$, le partage affine d'un octet z_i de $w[4]$ est de la forme $\mathbf{z}_i = ((z_i \oplus \beta''_i) \otimes \alpha^{-1}, \alpha^{-1}, \beta''_i)$ où $z_i = y_i \oplus x_i$ et $\beta''_i = \beta'_i \oplus \beta_i \oplus \mathrm{R}con_i[0]$.

Remarque 13. *Les fonctions de reconstruction de sortie de tous nos partages affines en* 3*-morceaux sont définies par* $rec_{aff} : (\tilde{x}, \alpha, \beta) \mapsto (\tilde{x} \otimes \alpha^{-1}) \oplus \beta$ *(voir Définition 22).*

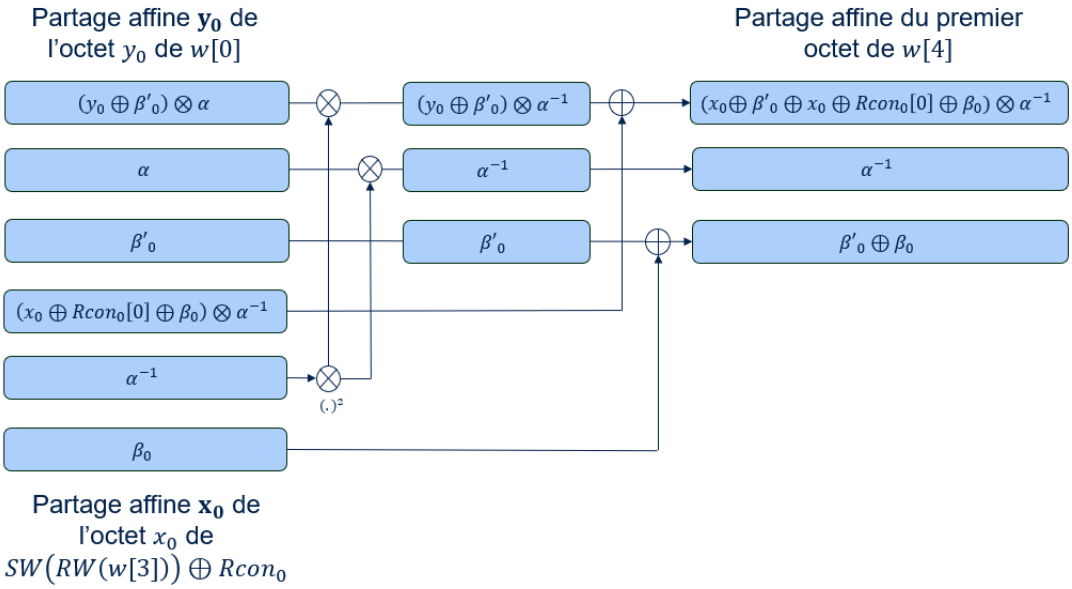


FIGURE 39 – Notre implémentation à seuil de l'étape 4 du calcul du premier octet de $w[4]$

Calculs de $w[5]$, $w[6]$ *et* $w[7]$. Les 3 autres mots de rk_1 sont obtenus en suivant la deuxième ligne de l'Équation 4.2 pour $j = 4$. Pour chaque $w[j]$, $j = \{5,6,7\}$, une addition est effectuée entre $w[j-4]$ et $w[j-1]$. Comme les masques multiplicatifs des partages affines des octets de $w[j-4]$ et $w[j-1]$ sont différents, il est nécessaire d'appliquer le premier cycle de la construction en Figure 39 pour rétablir le bon masque multiplicatif en fonction du tour (*i.e.* α^{-1} pour les tours pair et α pour les tours impairs). Puis, les partages affines des octets de $w[j]$ sont obtenus en ajoutant respectivement les premiers et troisièmes morceaux des partages affines des octets de $w[j-4]$ avec les premiers et troisièmes morceaux des partages affines des octets de $w[j-1]$. C'est une application directe du deuxième cycle de la construction en Figure 39.

Les 9 autres clés de tour se construisent de la même façon que rk_1. Pour les mêmes raisons invoquées en Section 4.4.1 (*i.e.* la Proposition 3 pour SW et la conservation de la sécurité en présence de glitches à travers les transformations linéaires pour les Xors), nous pouvons argumenter que notre implémentation à seuil de l'algorithme de cadencement de clé de l'AES en 9 cycles (voir Tableau 10) est sécurisée à l'ordre 1 en présence de glitches.

Remarque 14. *Les* 5 *cycles de l'opération SubWord s'exécutent pour chacun des* 4 *octets en parallèle.*

Fonctions	Nombre de cycles
RW	0
SW	4×5 (voir Remarque 14)
$\oplus \mathrm{R}con[j-1]$	1
$\oplus w[j-4]$	2
$w[j-4] \oplus w[j-1]$	1

TABLEAU 10 – Nombre de cycles pour chaque opération d'un tour de cadencement de clé de l'AES

Dans la section suivante, nous proposons une extension de notre évaluation monomiale sur des fonctions polynomiales.

4.5 Extension de notre proposition sur des fonctions polynomiales

Notre implémentation à seuil de la fonction F_{pow} présentée en Section 4.3 peut être appliquée à chaque monôme d'une fonction polynomiale définie sur GF(2^n). Le but est de sécuriser dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches l'implémentation d'une telle fonction. Nous décrivons dans cette section une contribution de nos recherches à l'état de l'art : l'aspect générique, permettant d'appliquer notre contre-mesure à plusieurs schémas de substitution-permutation. En effet, les propositions de l'état de l'art ne s'appliquent actuellement qu'à des algorithmes spécifiques comme l'AES.

Soit $\mathrm{F}(x) = \bigoplus_{i=0}^{2^n-2} x^{q_i} \in \mathrm{GF}(2^n)$ une fonction polynomiale telle que chaque q_i et la somme des q_i sont tous premiers avec $2^n - 1$. Soit également α et β deux variables indépendantes distribuées uniformément dans GF(2^n) et GF(2^n)*. Notre implémentation sécurisée de F prend en entrée un partage affine $\mathbf{x} = ((x \oplus \beta) \otimes \alpha, \alpha, \beta)$ de x et retourne en sortie un partage affine $\mathbf{F}(\mathbf{x}) = ((\mathrm{F}(x) \oplus \beta_\mathrm{F}) \otimes \alpha_\mathrm{F}, \alpha_\mathrm{F}, \beta_\mathrm{F})$ de la valeur $\mathrm{F}(x)$. Notre proposition d'extension est constituée de 6 cycles et est présentée dans les étapes suivantes.

Étape 1 : L'implémentation à seuil d'un monôme de F nécessite de calculer au préalable un partage additif en 2-morceaux de sa fonction de Dirac. Ce calcul n'est effectué qu'une seule fois pour sécuriser toute l'évaluation de la fonction polynomiale. Nous nous référons à la technique présentée en Section 4.2.2 afin de générer un tel partage en 1 cycle, via une table de correspondance.

Étape 2 : Ensuite, chaque monôme de F est évalué suivant notre technique présentée en Figure 36. Cette étape prend 4 cycles et peut être calculée en parallèle pour chaque monôme. Le nombre de portes logiques (*i.e.* d'opérations élémentaires) et de registres augmente linéairement en fonction du nombre de monômes de F évalués en parallèle.

Étape 3 : Enfin, nous recombinons les partages affines de chaque monôme obtenus dans l'étape précédente. Ce calcul s'effectue en 1 cycle et nous conduit à un partage affine $\mathbf{F}(\mathbf{x})$ de F. En particulier, les deuxièmes (respectivement les troisièmes) morceaux des partages affines sont multipliés (respectivement ajoutés) entre eux afin de produire le deuxième (respectivement troisième) morceau de $\mathbf{F}(\mathbf{x})$. Puis, les premiers morceaux des partages affines sont multipliés au deuxième morceau de tous les autres partages affines et les résultats sont ajoutés entre eux afin de former le premier morceau de $\mathbf{F}(\mathbf{x})$. Pour mieux comprendre cette étape, nous donnons un exemple ci-dessous.

Proposition 7. *La réalisation* 3*-masquée* $\mathbf{F}(\mathbf{x})$ *de la fonction* $\mathrm{F}(x) = \bigoplus_{i=0}^{2^n-2} x^{q_i}$ *décrite dans cette section est correcte, glitches-immune à l'ordre* 1 *et uniforme. C'est une implémentation à seuil de* F*, sécurisée à l'ordre* 1 *en présence de glitches.*

Preuve. *La preuve de la Proposition 7 est similaire à la preuve de la Proposition 3 mais pour la fonction* $\mathrm{F}(x) = \bigoplus_{i=0}^{2^n-2} x^{q_i} \in \mathrm{GF}(2^n)$ *et la fonction de reconstruction de sortie de décrite en Définition 22 et appliquée sur* $\mathbf{F}(\mathbf{x})$.

La Figure 40 illustre notre implémentation à seuil de la fonction polynomiale $\mathrm{F}(x) = x^8 \oplus x^{23}$ pour $x \in \mathrm{GF}(2^8)$, représenté par son partage affine $\mathbf{x} = ((x \oplus \beta) \otimes \alpha, \alpha, \beta)$. Premièrement, à partir de $\mathbf{x}$, nous calculons la table de correspondance de $\delta(x)$ en 1 cycle en suivant la méthode présentée en Section 4.2.2. Nous appliquons ensuite notre implémentation à seuil de la Figure 36 en 4 cycles sur les monômes x^8 et x^{23}. Ces deux calculs sont faits en parallèle et nous donnent deux partages affines **x8** de x^8 et **x23** de x^{23}, de la même forme que $\mathbf{x}$. En remarque, les valeurs $8, 23$ et $(8+23)$ sont premières avec (2^8-1). Enfin, afin d'obtenir un partage affine $\mathbf{F}(\mathbf{x})$ de $\mathrm{F}(x)$, nous recombinons **x8** et **x23** en 1 cycle de la manière suivante :

1. Nous multiplions le premier morceau de **x8** avec le deuxième morceau de **x23** et le premier morceau de **x23** avec le deuxième morceau de **x8**. Les résultats sont ajoutés et nous obtenons le premier morceaux de $\mathbf{F}(\mathbf{x})$ égal à $(x^8 \oplus \beta_{x^8} \oplus x^{23} \oplus \beta_{x^{23}}) \otimes (\alpha^8 \otimes \alpha^{23})$.
2. Nous multiplions entre eux les deuxièmes morceaux de **x8** et **x23**. Nous obtenons le deuxième morceau de $\mathbf{F}(\mathbf{x})$ égal à $\alpha_F = (\alpha^8 \otimes \alpha^{23})$.
3. Nous ajoutons entre eux les troisièmes morceaux de **x8** et **x23**. Nous obtenons le troisième morceau de $\mathbf{F}(\mathbf{x})$ égal à $\beta_F = (\beta_{x^8} \oplus \beta_{x^{23}})$.

Nous obtenons donc en 6 cycles un partage affine $\mathbf{F}(\mathbf{x}) = ((\mathrm{F}(x) \oplus \beta_F) \otimes \alpha_F, \alpha_F, \beta_F)$ de $\mathrm{F}(x)$.

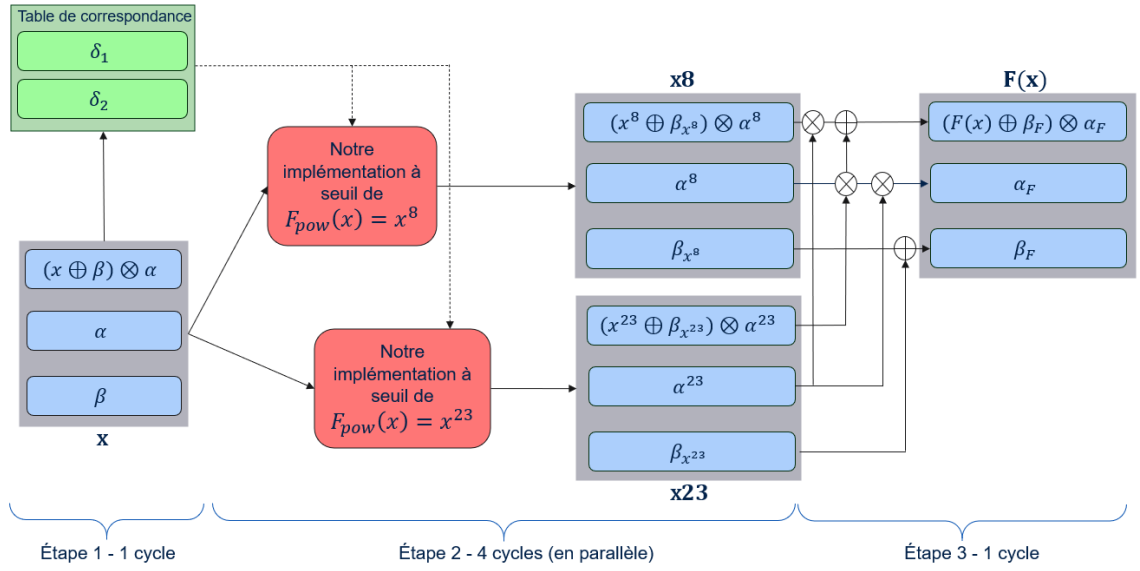


FIGURE 40 – Notre implémentation à seuil en 6 cycles de la fonction polynomiale $\mathrm{F}(x) = x^8 \oplus x^{23} \in \mathrm{GF}(2^8)$

En conclusion, nous proposons une évaluation de n'importe quelle fonction polynomiale [1] en 6 cycles.

Dans la section suivante, nous implémentons matériellement notre implémentation à seuil de la fonction inverse de l'AES de manière à prouver son caractère fonctionnel (*i.e.* la propriété de correction) autrement que mathématiquement, comme démontré jusqu'ici. Cette démarche nous permet également de comparer nos performances par rapport à celles des propositions l'état de l'art.

1. telle que chaque puissance de tous les termes de la fonction et la somme de ces puissances soient premiers avec l'ordre du sous groupe multiplicatif.

4.6 Implémentation matérielle de notre construction de la boîte-S de l'AES sur FPGA et comparaison avec l'état de l'art

Une de nos contributions à l'état de l'art est une construction de la boîte-S de l'AES totalement basée sur les implémentations à seuil. En effet, en Section 4.2.3, nous proposons une implémentation à seuil de la fonction de Dirac d'un octet. Comme notre technique est nouvelle dans la littérature cryptographique, nous avons tenu à vérifier son caractère fonctionnel et comparer nos performances aux autres propositions existantes.

Pour cela, nous avons implémenté notre construction présentée en Figure 38, telle que la fonction de Dirac soit sécurisée avec les implémentations à seuil (voir Figure 35), en Verilog sur le FPGA *PYNQ-Z2*. Ce circuit logique programmable est fabriqué par la société *Xilinx Zynq SoC* pour le programme de *l'Université Xilinx* afin de soutenir le support PYNQ ("Python Productivity for Zynq" en anglais) et le développement de systèmes intégrés. En effet, en utilisant le langage Python et les bibliothèques, les concepteurs peuvent tirer parti des avantages de la logique programmable et des microprocesseurs dans Zynq pour construire des systèmes intégrés plus efficaces. Nous avons utilisé l'outil Yosis afin de synthétiser notre code Verilog. Nous nous référons à l'étape 3 en Section 2.2.2 pour des explications sur la synthèse logique d'un circuit électronique. Nous avons ensuite simulé notre implémentation RTL de la boîte-S de l'AES à l'aide du logiciel *Vivado Design Suite*.

Dans la suite, nous présentons la structure de notre code RTL. Nous montrons ensuite la fonctionnalité de notre construction dans un exemple. Nous finirons par comparer nos performances avec celles de l'état de l'art.

4.6.1 Structure de notre code RTL

Comme présenté dans la Figure 38, notre code RTL prend en entrée un partage affine (voir Définition 22) de la sortie de l'opération *AddRoundKey*, soit l'addition entre le message clair x et la clé secrète k (pour le premier tour), et retourne un partage additif de la boîte-S de l'AES. Afin d'appliquer notre évaluation monomiale sur $(x \oplus k)^{254}$ dans le corps de l'AES, un partage additif de la fonction de Dirac $\delta(x \oplus k)$ doit être calculé. Nous avons testé notre implémentation à seuil de la fonction de Dirac illustrée dans la Figure 35. Une première étape consiste en la transformation du partage affine de $x \oplus k$ en un partage additif en 3-morceaux de $x \oplus k$ (voir Figure 34). Puis, les 7 implémentations à seuil $\mathbf{F}''$ des 7 multiplications des bits complémentés de $x \oplus k$ sont calculées de manière à retourner le partage additif en 3-morceaux de $\delta(x \oplus rk)$. Ce dernier partage est ensuite transformé en un partage additif en 2-morceaux de $\delta(x \oplus rk)$ (voir Remarque 7). Afin d'assurer l'uniformité de la construction nous avons introduit 7 mots aléatoires ri ($i \in [1..7]$) codés sur 2 bits tels que le partage additif en 3-morceaux $(ri[0], ri[1], ri[0] \oplus ri[1])$ de 0 soit utilisé pour chaque implémentation à seuil $\mathbf{F}''$. La structure de notre code RTL est illustrée en Figure 41.

Pour 14 bits aléatoires ajoutés par boîte-S, notre implémentation à seuil occupe une surface de 3.4 GE et s'exécute en 8 cycles. Dans la suite, nous simulons notre code RTL avec des valeurs arbitrairement choisies en entrée.

4.6.2 Exemple de simulation de notre code RTL

La Figure 42 représente un exemple de simulation de notre code RTL sous Vivado, où les notations sont en hexadécimale. Notre code RTL prend en entrée le partage affine de $(x \oplus k)$ suivant :

$$((x \oplus k \oplus bool\_mask) \otimes mult\_mask, mult\_mask, bool\_mask),$$

tel que :

— le message clair $x = 20$

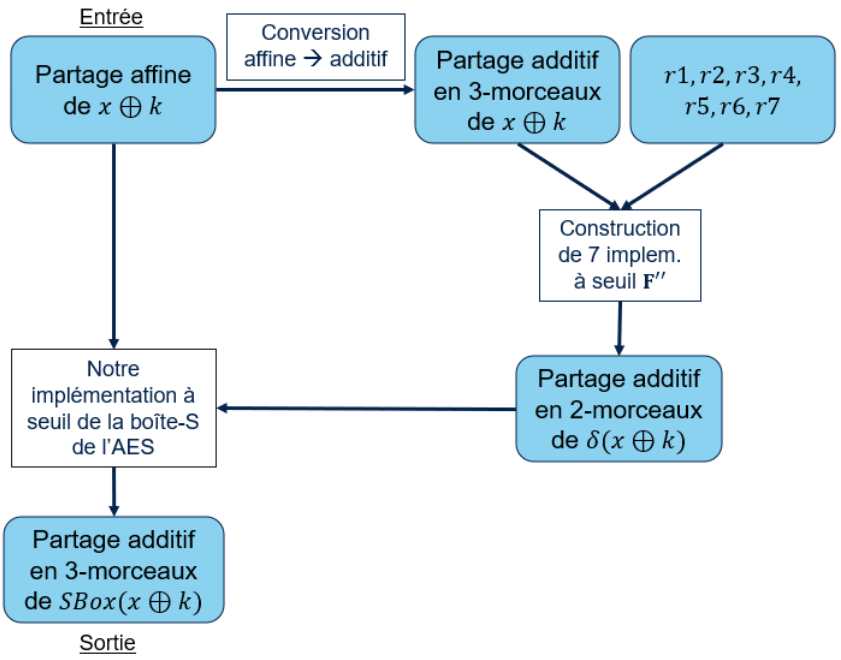


FIGURE 41 – Structure de notre code RTL

— la clé secrète $k = 2d$
— la valeur $x \oplus k = 20 \oplus 2d = 0d$
— le masque additif $bool\_mask = 0f$ (*i.e.* notre masque β)
— le masque multiplicatif $mult\_mask = f4$ (*i.e.* notre masque α)

En appliquant la méthode de la Figure 34, à partir du partage affine de $0d$ on obtient un partage additif de $0d$. Un partage additif de la fonction de Dirac de $0d$ est ensuite calculé à partir du partage additif de $0d$ et des 7 mots de deux bits $r1, r2, r3, r4, r5, r6$ et $r7$ respectivement égaux à $0, 1, 1, 1, 3, 1$ et 1.

Notre implémentation à seuil de la boîte-S de l'AES représentée en Figure 38 est appliquée pour le partage affine de $0d$ et le partage additif de $\delta(0d)$ en entrée. À la fin de la simulation, notre code RTL retourne un partage additif $(sb1, sb2, sb3)$ (*i.e.* soit notre partage additif $\mathbf{x}_3$ de la Figure 38). Nous obtenons le partage additif $(72, 68, a5)$. Il est facile de vérifier que ce partage correspond à celui de la boîte-S de $0d$. En effet, en appliquant la fonction de reconstruction de sortie sur ce partage, on a $72 \oplus a5 = d7 = \text{SB}ox(0d)$. La fonctionnalité de notre proposition est donc validée. En remarque, le morceau $sb2 = 68$ servira plus tard comme masque multiplicatif pour le calcul de la boîte-S au tour suivant.

Dans la suite nous présentons nos performances par rapport à celles des propositions de l'état de l'art.

4.6.3 Comparaison avec l'état de l'art

Pour ce calcul de boîte-S, après la synthèse et l'analyse de notre implémentation sur le FPGA, nous obtenons une surface en de 3,4k GE (portes équivalentes). De manière à pouvoir nous comparer aux implémentations à seuil proposées dans l'état de l'art, nous ajoutons nos performances au Tableau 4. Il en résulte le Tableau 11.

Lorsque la fonction de Dirac est implémentée par sa table de correspondance, notre proposition est meilleure ou équivalentes en termes de nombre de cycles que celles proposées dans

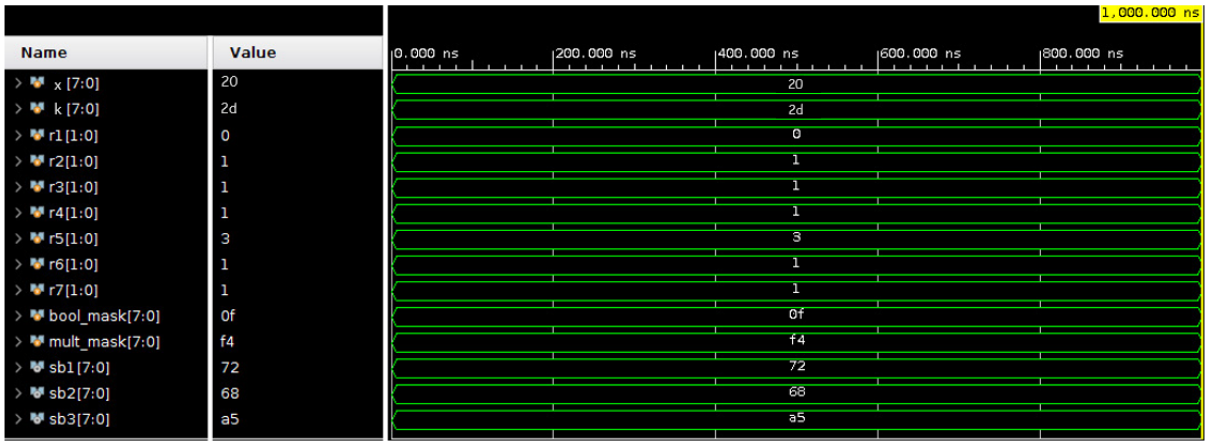


FIGURE 42 – Simulation sous Vivado de notre code RTL en Verilog de la boîte-S de l'AES

[64, 10, 22, 40] et présente l'avantage de ne pas générer de nouveaux bits aléatoires. Comparé à [82], notre construction prend un cycle de plus mais peut être étendue facilement pour évaluer un grand nombre de fonctions polynomiales, comme montré en Section 4.5. De plus, lorsque la fonction Dirac est sécurisée avec les implémentations à seuil des multiplications bit à bit, notre technique est meilleure que [82] en terme de surface. Enfin, nos deux implémentations à seuil sont applicables pour n'importe quelle boîte-S équivalente à un calcul de F_{pow}, ou une somme de fonctions F_{pow} sous certaines conditions sur les puissances, citées en Section 4.5.

Conception	**Surface [kGE/boîte-S]**	**Aléa [bits/boîte-S]**	**Nb cycles /boîte-S**	**applicable pour d'autres puissances?**
[64]	4.2	44	7	non
[10]	2.2	16	5	non
[22]	1.9	54	5	non
[40]	2.6	18	6	non
[82]	3.5	0	4	non
Notre travail [49] (Dirac en 7 implem. à seuil)	3.4	14	8	oui (voir Section 4.5)
Notre travail [49] (Dirac en table)	3.2 + registres (256 bits)	0	5	oui (voir Section 4.5)

TABLEAU 11 – Implémentations à seuil de la boîte-S de l'AES sécurisée à l'ordre 1 en présence de glitches

Le Tableau 11 compare les performances de différentes implémentations à seuil de la boîte-S de l'AES résistantes contre le modèle d'attaque par sondage à l'ordre 1 en présence de glitches. Dans la suite, nous proposons une nouvelle approche afin de générer une implémentation à seuil d'une fonction polynomiale, résistante contre un modèle d'attaque par sondage à l'ordre $t > 1$ en présence de glitches.

4.7 Notre évaluation polynomiale sécurisée à l'ordre supérieur en présence de glitches

4.7.1 État de l'art et nouvelles problématiques

Dans cette section, on considère le modèle d'attaque par sondage en présence de glitches mais à l'ordre $t > 1$ (voir Définition 17). Dans [58], Messerges introduit pour la première fois la notion

d'attaque par observations à l'ordre 2. Plus généralement, dans le contexte des implémentations à seuil d'une (n,m)-fonction, on parle d'*attaque par observation à l'ordre supérieur* lorsqu'un attaquant est capable d'observer n'importe quelle combinaison des entrées de $t > 1$ sous-fonctions de la réalisation de la fonction. La propriété de glitches-immunité à l'ordre t présentée en Définition 20 est nécessaire pour assurer une sécurité contre un tel adversaire.

Pour la communauté cryptographique, l'étude d'une implémentation à seuil d'une (n,m)-fonction sécurisée dans le modèle d'attaque par sondage à l'ordre supérieur en présence de glitches est un sujet de recherche encore ouvert et actif, notamment en ce qui concerne l'amélioration des performances. En effet, une telle construction devient coûteuse quand l'ordre de sécurité augmente ou que le degré algébrique (voir Définition 4) de la fonction augmente. Ces deux paramètres sont liés au nombre minimal de morceaux nécessaires pour satisfaire la propriété de glitches-immunité. Bilgin [9] propose le théorème suivant :

Théorème 2. *[9] Soit une variable x et $\mathrm{F}(x)$ une (n,m)-fonction de degré algébrique s. Le nombre d minimum de morceaux du partage de x requis pour implémenter une réalisation de* F *glitches-immune à l'ordre t, est donné par $d \geq s \times t + 1$.*

D'après le Théorème 2, plus l'ordre de sécurité ou le degré algébrique augmentent, plus le nombre de morceaux des données sensibles est important. En conséquence, comme ces morceaux doivent être stockés dans des registres, cela augmente la surface de l'implémentation et le nombre de cycles nécessaires pour sécuriser le calcul. Dans le Tableau 12, les performances des travaux de l'état de l'art proposant une implémentation à seuil de la boîte-S de l'AES sécurisée à l'ordre 2 en présence de glitches sont affichées.

Conception	Surface [kGE/boîte-S]	Aléa [bits/boîte-S]	Nb cycles/boîte-S
Cnudde *et. al.* [21]	11.1	126	6
Cnudde *et. al.* [22]	3.7	162	6
Gross *et. al.* [40]	10	84	6
Wei *et. al.* [88]	4.3	116	6

TABLEAU 12 – État de l'art - Implémentations à seuil de la boîte-S de l'AES sécurisée à l'ordre 2 en présence de glitches

Les implémentations à seuil du Tableau 12 sont basées sur la méthode de Nikova *et. al.* présentée en Section 3.4. Dans l'état de l'art, seuls Reparaz *et. al.* [72] ont proposé une alternative à cette méthode. Afin de proposer de nouvelles idées, nous nous sommes posés les questions suivantes :

(i) Comment créer une implémentation à seuil de n'importe quelle (n,m)-fonction? C'est à dire comment obtenir une méthode générique, applicable sur n'importe quelle boîte-S?

(ii) Comment assurer que notre construction soit sécurisée à n'importe quel ordre en présence de glitches?

Dans la suite, nous montrons comment satisfaire ces deux problématiques. Nos recherches sont basées sur la *méthode CPRR*, présentée par Carlet *et. al.* dans [17] et pourront par exemple s'appliquer facilement pour sécuriser le calcul d'une boîte-S de l'AES. Notre objectif est d'introduire une nouvelle approche permettant de générer une implémentation à seuil d'une (n,m)-fonction, autre que celle de Nikova *et. al.* présentée en Section 3.4.

4.7.2 Présentation de la méthode CPRR

Dans l'état de l'art, plusieurs travaux [16, 24, 25, 45, 79] se sont concentrés sur l'implémentation sécurisée d'un chiffrement par blocs de la forme SPN (e.g. AES) dans le modèle d'attaque

par sondage à l'ordre supérieur (sans présence de glitches). Ces travaux consistent en la représentation des fonctions non-linéaires de ces blocs chiffrés (*i.e.* les boîtes-S) par des polynômes sur un corps fini. L'évaluation de tels polynômes implique des opérations linéaires (e.g. Xor) qui sont sécurisées directement avec du masquage additif, et des opérations non-linéaires (e.g. la multiplication) qui sont par exemple sécurisées avec le schéma ISW [42] présenté en Section 3.2.4. Cette évaluation polynomiale a pour objectif de minimiser le nombre de multiplications non-linéaires (*i.e.* des multiplications qui ne peuvent pas s'écrire sous la forme d'une puissance de 2) afin de masquer efficacement l'implémentation concernée. Pour l'implémentation d'une fonction F donnée, ce nombre de multiplications non-linéaires est appelé *complexité de masquage* de F. Jusqu'en 2015, la méthode la plus performante pour évaluer la sécurité des fonctions non-linéaires dans le modèle d'attaque par sondage à l'ordre supérieur était celle proposée par Coron *et. al.* [25].

Dans ce contexte, Carlet *et. al.* ont proposé en 2016 [17] une alternative à la méthode de Coron *et. al.* , appelée ici la méthode CPRR. Afin de minimiser la complexité de masquage de chaque boîte-S, les auteurs introduisent une nouvelle décomposition algébrique d'une telle fonction polynomiale en plusieurs fonctions de plus petit degré algébrique (voir Définition 4). Ils évaluent leur décomposition dans le modèle d'attaque par sondage à l'ordre supérieur par le théorème suivant.

Théorème 3. *[17] Soit* F *une* (n,m)*-fonction de degré algébrique au plus égal à* s*. Pour tous entiers* $d > s$ *et* $j \leq s$*, nous avons :*

$$\mathrm{F}\left(\sum_{i=1}^{d} x_i\right) = \sum_{j=1}^{s} \binom{d-j-1}{s-j} mod\ 2 \times \sum_{\substack{\mathrm{I}\subseteq[\![1,d]\!] \\ |\mathrm{I}|=j}} \mathrm{F}\left(\sum_{i\in\mathrm{I}} x_i\right). \tag{4.3}$$

Le Théorème 3 stipule que l'évaluation de F sur la somme des d morceaux x_i peut être représentée par plusieurs évaluations de F sur la somme $\sum_{i\in\mathrm{I}} x_i$, avec $\mathrm{I} \subseteq [\![1,d]\!]$ et $|\mathrm{I}| \leq s$. En d'autres termes, cela réduit l'évaluation de F dans le modèle d'attaque par sondage à l'ordre $(d-1)$ en plusieurs évaluations de F dans le modèle d'attaque par sondage à l'ordre $(j-1)$, où $j = |\mathrm{I}|$ est borné supérieurement par le degré algébrique s de F. Ainsi, chaque évaluation prend j morceaux $(x_i)_{i\in\mathrm{I}}$ et calcule un partage en j-morceaux de $\mathrm{F}(\sum_{i\in\mathrm{I}} x_\mathrm{I})$. Par construction, chaque évaluation est fonctionnellement indépendante d'au moins $(d-s)$ morceaux du partage de x. En remarque, à la fin, il sera nécessaire de combiner de manière sécurisée les morceaux obtenus de tous les $\mathrm{F}(\sum_{i\in\mathrm{I}} x_\mathrm{I})$ afin d'obtenir un partage en d-morceaux de $\mathrm{F}(x)$.

Dans la suite, nous étudions l'application de la méthode CPRR dans le modèle d'attaque par sondage à l'ordre supérieur en présence de glitches. Le but est de construire une implémentation à seuil d'une (n,m)-fonction à partir de l'Équation 4.3, sécurisée dans ce modèle.

4.7.3 Notre approche : de la méthode CPRR à l'implémentation à seuil

Soit $\mathrm{F}(x)$ une (n,m)-fonction de degré algébrique s. Soit t l'ordre de sécurité souhaité dans le modèle d'attaque par sondage en présence de glitches. La donnée sensible x est découpée en d morceaux selon le Théorème 2. Soit F_{dec} la décomposition algébrique de $\mathrm{F}(x)$ donnée par Carlet *et. al.* dans l'Équation 4.3. À partir de F_{dec}, nous souhaitons construire une implémentation à seuil **F** de F. Notre méthode est décrite dans les étapes suivantes :

Étape 1 : Pour un élément v, on a $\binom{0}{0} = \binom{v}{0} = 1\ mod\ 2$. Avec les paramètres s, t et d connus, on développe F_{dec}. Soit γ le nombre de termes de F_{dec}. Pour $1 \leq j \leq s$, γ est défini par Carlet *et. al.* [17] tel que :

$$\gamma = \#\left\{\mathrm{I}\subseteq[\![1,d]\!] \text{ tel que } |\mathrm{I}| = j \text{ et } \binom{d-j-1}{s-j} = 1\ mod\ 2\right\}. \tag{4.4}$$

Étape 2 : Le but est maintenant de construire une réalisation **F** de F. Pour cela, nous attribuons chaque terme de F_{dec} à une sous-fonction de **F**. Nous obtenons une réalisation γ-masquée **F** de F et nous avons la proposition ci-dessous.

Proposition 8. *La réalisation γ-masquée* **F** *de la (n,m)-fonction* F *décrire dans l'étape 2 est correcte et glitches-immune à l'ordre t. Elle est sécurisée à l'ordre t en présence de glitches.*

Preuve. *Par construction (voir Equation 4.3), la réalisation* **F** *est correcte. De plus, chaque terme de F_{dec} prend en entrée maximum $j \leq s$ morceaux du partage de x. On a l' équivalence suivante : $j \leq s \Leftrightarrow j \times t \leq s \times t$ (pour t un entier strictement positif). D'après le Théorème 2, on a $s \times t + 1 \leq d$ c'est à dire $s \times t \leq d-1$, soit $j \times t \leq d-1$. En d'autres termes, la combinaison des entrées de t termes de F_{dec} est fonctionnellement indépendante d'au moins 1 morceau du partage de x. Comme un terme de F_{dec} est attribué à une sous-fonction de* **F**, *d'après la Définition 20, la réalisation* **F** *est glitches-immune à l'ordre t. Elle est en conséquence sécurisée à l'ordre t en présence de glitches.*

□

Étape 3 : Dans certains cas, il existe plusieurs valeurs de j pour lesquelles le coefficient binomial $\binom{d-j-1}{s-j}$ est non-nul. On note alors $jmax$ la valeur maximale possible de j telle que $\binom{d-j-1}{s-j} = 1 \ mod \ 2$. Dans ce contexte, nous présentons une technique d'optimisation afin de minimiser le nombre de sous-fonctions de **F**. Notre technique doit satisfaire les deux points suivants :

- Si **toutes les entrées** d'un terme de F_{dec} sont communes avec celles d'un ou plusieurs autre(s) terme(s), nous pouvons regrouper ces termes dans une seule sous-fonction.
- Cela implique que chaque sous-fonction **ne doit pas prendre plus que** $jmax$ morceaux du partage de x en entrée. Par construction, chaque sous-fonction doit être fonctionnellement indépendante d'au moins $(d - jmax)$ morceaux du partage de x.

Soit γ_{opt} le nombre optimisé de sous-fonctions et soit $\mathbf{F}'$ la nouvelle réalisation γ_{opt}-masquée de F. On définit γ_{opt} par la relation suivante :

$$\gamma_{opt} = \#\left\{\mathrm{I} \subseteq [\![1,d]\!] \text{ tel que } |\mathrm{I}| = jmax\right\}. \tag{4.5}$$

Nous avons la proposition suivante.

Proposition 9. *La réalisation γ_{opt}-masquée $\mathbf{F}'$ de la (n,m)-fonction* F *décrire dans l'étape 3 est correcte et glitches-immune à l'ordre t. Elle est sécurisée à l'ordre t en présence de glitches.*

Preuve. *Comme nous ne supprimons aucun termes de F_{dec}, d'après la Proposition 8, $\mathbf{F}'$ est correcte. De plus, si nous respectons les deux conditions citées ci-dessus, chaque sous-fonction de $\mathbf{F}'$ prend en entrée maximum $jmax \leq s$ morceaux du partage de x. D'après le Théorème 2, on a $jmax \times t \leq d-1$. La combinaison des entrées de t sous-fonctions de $\mathbf{F}'$ est donc toujours fonctionnellement indépendante d'au moins 1 morceau du partage de x. D'après la Définition 20, la réalisation γ_{opt}-masquée $\mathbf{F}'$ de* F *est glitches-immune à l'ordre t. Elle est en conséquence sécurisée à l'ordre t en présence de glitches.*

□

Étape 4 : Si la réalisation $\mathbf{F}'$ ne satisfait pas la propriété d'uniformité (voir Définition 21), nous utilisons la technique de remasquage présentée en Proposition 1 afin de transformer $\mathbf{F}'$ en une nouvelle réalisation γ_{opt}-masquée $\mathbf{F}''$ de F uniforme. À la fin, $\mathbf{F}''$ est une construction d'une implémentation à seuil de la fonction F.

Nous venons de présenter une approche permettant de construire une implémentation à seuil d'une (n,m)-fonction, sécurisée à l'ordre supérieur en présence de glitches. Dans la suite, nous appliquons notre proposition dans un exemple.

4.7.4 Application de notre approche à l'ordre 2 sur la fonction AND

Soit la fonction $\mathrm{F} : (x,y) \in (\mathrm{GF}(2))^2 \mapsto x \otimes y \in \mathrm{GF}(2)$. Dans cette partie nous montrons comment construire une implémentation à seuil de F, sécurisée à l'ordre 2 en présence de glitches, en suivant les étapes de notre approche décrite en Section 4.7.3. Nous remarquons que la décomposition algébrique de Carlet *et. al.* donnée dans le Théorème 3 est définie pour une unique variable en entrée. Nous l'adaptons ci-dessous pour deux variables x et y en entrée :

$$\mathrm{F}\left((\sum_{i=1}^{d} x_i),(\sum_{i=1}^{d} y_i)\right) = \sum_{j=1}^{s} \binom{d-j-1}{s-j} mod\ 2 \times \sum_{\substack{\mathrm{I}\subseteq[\![1,d]\!] \\ |\mathrm{I}|=j}} \mathrm{F}\left(\sum_{i\in\mathrm{I}} x_i, \sum_{i\in\mathrm{I}} y_i\right).$$

Nous avons alors :

Étape 1 : Le degré algébrique de F est $s = 2$. Pour une sécurité à l'ordre $t = 2$ en présence de glitches, le Théorème 2 stipule que le nombre minimal d de morceaux des partages des entrées x et y doit être supérieur ou égal à 5. Nous choisissons $d = 5$ et introduisons les partages additifs en 5-morceaux $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$ de x et $\mathbf{y} = (y_1, y_2, y_3, y_4, y_5)$ de y.

Pour $j \in \{1,2\}$, le coefficient binomial $\binom{5-j-1}{2-j}$ est non-nul pour $j = 1$ et $j = 2$. Par définition, $\mathrm{I} \subseteq [\![1,5]\!]$, on a alors les ensembles I possibles suivants :

— $\mathrm{I} \in \{1,2,3,4,5\}$ pour $|\mathrm{I}| = j = 1$,

— $\mathrm{I} \in \{(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(3,5),(4,5)\}$ pour $|\mathrm{I}| = j = 2$,

soient 15 ensembles I possibles. D'après l'Equation 4.4, la décomposition algébrique F_{dec} de Carlet *et. al.* de la fonction F comporte $\gamma = 15$ termes telle que :

$$\begin{aligned}
\mathrm{F}_{dec} = {} & \mathrm{F}(x_1,y_1) \oplus \mathrm{F}(x_2,y_2) \oplus \mathrm{F}(x_3,y_3) \oplus \mathrm{F}(x_4,y_4) \oplus \mathrm{F}(x_5,y_5) \\
& \oplus \mathrm{F}((x_1 \oplus x_2),(y_1 \oplus y_2)) \oplus \mathrm{F}((x_1 \oplus x_3),(y_1 \oplus y_3)) \oplus \mathrm{F}((x_1 \oplus x_4),(y_1 \oplus y_4)) \oplus \mathrm{F}((x_1 \oplus x_5),(y_1 \oplus y_5)) \\
& \oplus \mathrm{F}((x_2 \oplus x_3),(y_2 \oplus y_3)) \oplus \mathrm{F}((x_2 \oplus x_4),(y_2 \oplus y_4)) \oplus \mathrm{F}((x_2 \oplus x_5),(y_2 \oplus y_5)) \\
& \oplus \mathrm{F}((x_3 \oplus x_4),(y_3 \oplus y_4)) \oplus \mathrm{F}((x_3 \oplus x_5),(y_3 \oplus y_5)) \\
& \oplus \mathrm{F}((x_4 \oplus x_5),(y_4 \oplus y_5)).
\end{aligned} \tag{4.6}$$

À partir de F_{dec}, nous devons construire une réalisation 15-masquée $\mathbf{F}$ de F. L'étape suivante nous montre comment assurer que $\mathbf{F}$ soit correcte et glitches-immune à l'ordre 2.

Étape 2 : Nous attribuons un terme de F_{dec} (voir Equation 4.6) à chaque sous-fonction de notre réalisation 15-masquée $\mathbf{F} = (f_1, ..., f_{15})$ de F, définie ainsi par :

$$\begin{aligned}
f_1(x_1,y_1) &= \mathrm{F}(x_1,y_1) & f_2(x_2,y_2) &= \mathrm{F}(x_2,y_2) \\
f_3(x_3,y_3) &= \mathrm{F}(x_3,y_3) & f_3(x_4,y_4) &= \mathrm{F}(x_4,y_4) \\
f_5(x_5,y_5) &= \mathrm{F}(x_5,y_5) & f_6(x_1,x_2,y_1,y_2) &= \mathrm{F}((x_1 \oplus x_2),(y_1 \oplus y_2)) \\
f_7(x_1,x_3,y_1,y_3) &= \mathrm{F}((x_1 \oplus x_3),(y_1 \oplus y_3)) & f_8(x_1,x_4,y_1,y_4) &= \mathrm{F}((x_1 \oplus x_4),(y_1 \oplus y_4)) \\
f_9(x_1,x_5,y_1,y_5) &= \mathrm{F}((x_1 \oplus x_5),(y_1 \oplus y_5)) & f_{10}(x_2,x_3,y_2,y_3) &= \mathrm{F}((x_2 \oplus x_3),(y_2 \oplus y_3)) \\
f_{11}(x_2,x_4,y_2,y_4) &= \mathrm{F}((x_2 \oplus x_4),(y_2 \oplus y_4)) & f_{12}(x_2,x_5,y_2,y_5) &= \mathrm{F}((x_2 \oplus x_5),(y_2 \oplus y_5)) \\
f_{13}(x_3,x_4,y_3,y_4) &= \mathrm{F}((x_3 \oplus x_4),(y_3 \oplus y_4)) & f_{14}(x_3,x_5,y_3,y_5) &= \mathrm{F}((x_3 \oplus x_5),(y_3 \oplus y_5)) \\
f_{15}(x_4,x_5,y_4,y_5) &= \mathrm{F}((x_4 \oplus x_5),(y_4 \oplus y_5)).
\end{aligned}$$

Nous notons que ce n'est pas la seule réalisation de F possible. La fonction de reconstruction de sortie de $\mathbf{F}$ est définie par $\bigoplus_{i=1}^{15} f_i = x \otimes y = \mathrm{F}(x,y)$. D'après la Définition 19, notre réalisation 15-masquée $\mathbf{F}$ est donc correcte. Par ailleurs, la combinaison des entrées de 2 sous-fonctions de $\mathbf{F}$ permet d'obtenir de l'information sur maximum 4 morceaux (sur 5) des partages de x et de y. En

d'autres termes, la combinaison des entrées de 2 sous-fonctions de **F** est fonctionnellement indépendante d'un morceau du partage de x et d'un morceau du partage de y. D'après la Définition 20, notre réalisation **F** est donc glitches-immune à l'ordre 2. Nous venons d'illustrer la Proposition 8 dans le cas particulier de notre exemple. Dans l'étape suivante, nous montrons qu'il est possible d'optimiser le nombre de sous-fonctions de **F**.

Étape 3 : Comme le coefficient binomial $\binom{5-j-1}{2-j}$ est non-nul pour plusieurs valeurs de j (*i.e.* $j = 1$ et $j = 2 = jmax$), il est possible de réduire le nombre γ de sous-fonctions de **F**. Par exemple, comme toutes les entrées des deux premiers termes de F_{dec} (*i.e.* x_1, y_1 et x_2, y_2) sont communes avec celles du sixième terme de F_{dec} (voir Equation 4.6), nous pouvons regrouper ces termes dans une unique sous-fonction. Le raisonnement est similaire pour le troisième, quatrième et cinquième terme de F_{dec}. D'après l'Equation 4.5, le nombre optimisé γ_{opt} de sous-fonctions de **F** est défini par le nombre d'ensembles I tel que $|\mathrm{I}| = 2$, soit $\gamma_{opt} = 10$.

Remarque 15. *Notre nombre optimisé de sous-fonctions γ_{opt} correspond à celui obtenu par la construction de Reparaz* et. al. *[72]. Les auteurs se basent sur le Théorème 2 de Bilgin [9] qui stipule que le nombre de sous-fonctions d'une implémentation à seuil d'une fonction de degré algébrique s est toujours borné inférieurement par la valeur $\binom{d}{s}$ (où d représente le nombre de morceaux de l'entrée de la fonction). À l'ordre 2, nous avons bien $\gamma_{opt} = \binom{5}{2} = 10$. Par ailleurs, notre optimisation présente l'avantage d'être applicable sur toutes les fonctions de degré algébrique $s = 2$.*

Après avoir vérifié que chacune des nouvelles sous-fonctions ne prennent pas plus que 2 morceaux de x et 2 morceaux de y en entrée, nous obtenons la nouvelle réalisation 10-masquée $\mathbf{F}' = (f'_1, ..., f'_{10})$ de F suivante :

$$
\begin{aligned}
f'_1(x_1, x_2, y_1, y_2) &= \mathrm{F}(x_1, y_1) \oplus \mathrm{F}(x_2, y_2) \oplus \mathrm{F}((x_1 \oplus x_2), (y_1 \oplus y_2)) \\
f'_2(x_1, x_3, y_1, y_3) &= \mathrm{F}(x_3, y_3) \oplus \mathrm{F}((x_1 \oplus x_3), (y_1 \oplus y_3)) \\
f'_3(x_1, x_4, y_1, y_4) &= \mathrm{F}(x_4, y_4) \oplus \mathrm{F}((x_1 \oplus x_4), (y_1 \oplus y_4)) \\
f'_4(x_1, x_5, y_1, y_5) &= \mathrm{F}(x_5, y_5) \oplus \mathrm{F}((x_1 \oplus x_5), (y_1 \oplus y_5)) \\
f'_5(x_2, x_3, y_2, y_3) &= \mathrm{F}((x_2 \oplus x_3), (y_2 \oplus y_3)) \\
f'_6(x_2, x_4, y_2, y_4) &= \mathrm{F}((x_2 \oplus x_4), (y_2 \oplus y_4)) \\
f'_7(x_2, x_5, y_2, y_5) &= \mathrm{F}((x_2 \oplus x_5), (y_2 \oplus y_5)) \\
f'_8(x_3, x_4, y_3, y_4) &= \mathrm{F}((x_3 \oplus x_4), (y_3 \oplus y_4)) \\
f'_9(x_3, x_5, y_3, y_5) &= \mathrm{F}((x_3 \oplus x_5), (y_3 \oplus y_5)) \\
f'_{10}(x_4, x_5, y_4, y_5) &= \mathrm{F}((x_4 \oplus x_5), (y_4 \oplus y_5)).
\end{aligned}
$$

D'après la Proposition 9, la réalisation 10-masquée $\mathbf{F}'$ est correcte et glitches-immune à l'ordre 2. La preuve de cette proposition est la même que celle décrite en étape 2 ci-dessus. De la même manière que la réalisation 3-masquée $\mathbf{F}'$ de F décrite dans les Équations 3.4 en Section 3.2.4, la réalisation 10-masquée $\mathbf{F}'$ de F n'est pas uniforme. L'étape suivante montre comment pallier à ce problème.

Étape 4 : Nous appliquons la technique de remasquage présentée en Proposition 1 sur la réalisation 10-masquée $\mathbf{F}'$. Nous introduisons 9 variables aléatoires $(r_1, r_2, ..., r_9)$ statistiquement indépendantes et uniformément distribuées dans GF(2). Nous définissons une nouvelle réalisation

10-masquée $\mathbf{F}''' = (f_1''', ..., f_{10}''')$ de F telle que :

$$
\begin{aligned}
f_1'''(x_1, x_2, y_1, y_2) &= \mathrm{F}(x_1, y_1) \oplus \mathrm{F}(x_2, y_2) \oplus \mathrm{F}((x_1 \oplus x_2), (y_1 \oplus y_2)) \oplus r_1 \\
f_2'''(x_1, x_3, y_1, y_3) &= \mathrm{F}(x_3, y_3) \oplus \mathrm{F}((x_1 \oplus x_3), (y_1 \oplus y_3)) \oplus r_2 \\
f_3'''(x_1, x_4, y_1, y_4) &= \mathrm{F}(x_4, y_4) \oplus \mathrm{F}((x_1 \oplus x_4), (y_1 \oplus y_4)) \oplus r_3 \\
f_4'''(x_1, x_5, y_1, y_5) &= \mathrm{F}(x_5, y_5) \oplus \mathrm{F}((x_1 \oplus x_5), (y_1 \oplus y_5)) \oplus r_4 \\
f_5'''(x_2, x_3, y_2, y_3) &= \mathrm{F}((x_2 \oplus x_3), (y_2 \oplus y_3)) \oplus r_5 \\
f_6'''(x_2, x_4, y_2, y_4) &= \mathrm{F}((x_2 \oplus x_4), (y_2 \oplus y_4)) \oplus r_6 \\
f_7'''(x_2, x_5, y_2, y_5) &= \mathrm{F}((x_2 \oplus x_5), (y_2 \oplus y_5)) \oplus r_7 \\
f_8'''(x_3, x_4, y_3, y_4) &= \mathrm{F}((x_3 \oplus x_4), (y_3 \oplus y_4)) \oplus r_8 \\
f_9'''(x_3, x_5, y_3, y_5) &= \mathrm{F}((x_3 \oplus x_5), (y_3 \oplus y_5)) \oplus r_9 \\
f_{10}'''(x_4, x_5, y_4, y_5) &= \mathrm{F}((x_4 \oplus x_5), (y_4 \oplus y_5)) \oplus \bigoplus_{i=1}^{9} r_i.
\end{aligned}
$$

D'après la Proposition 1, la réalisation 10-masquée $\mathbf{F}'''$ est uniforme. Étant aussi correcte et glitches-immune à l'ordre 2, $\mathbf{F}'''$ est une implémentation à seuil possible de la fonction F. Notre construction est en conséquence sécurisée à l'ordre 2 en présence de glitches.

Dans la suite, nous concluons et comparons notre nouvelle approche à celles de l'état de l'art.

4.7.5 Conclusion et comparaison de notre approche à celle de Nikova *et. al.*

Nous avons présenté une construction possible d'une implémentation à seuil d'une (n, m)-fonction. D'après les Propositions 8 et 9, notre approche est prouvée comme étant sécurisée à l'ordre supérieur en présence de glitches. Contrairement à la méthode de Nikova *et. al.* [65], notre construction peut s'appliquer sur n'importe quelle (n, m)-fonction, autre que le boîte-S de l'AES, et propose une sécurité dans le modèle d'attaque par sondage en présence de glitches à n'importe quel ordre. Nous avons donc répondu aux problématiques (i) et (ii) posées en Section 4.7.1

De plus, notre approche permet d'assurer la propriété de glitches-immunité à l'ordre t de manière directe, contrairement à la technique de Nikova *et. al.* . Prenons la fonction $\mathrm{F}(x, y) = x \otimes y$, le but est de construire les sous-fonctions de la réalisation de F. Avec la méthode de Nikova *et. al.* , le développeur doit réfléchir à la manière de trier les termes de la décomposition de $\mathrm{F}(\bigoplus_{i=1}^{d} x_i \otimes \bigoplus_{i=1}^{d} y_i)$ dans des sous-fonctions, en respectant la glitches-immunité à l'ordre t. Plus le nombre de morceaux d des partages de x et y est élevé, plus ce travail est difficile. Avec notre approche, le développeur n'a pas besoin de réfléchir à la façon de regrouper les termes de la décomposition algébrique de $\mathrm{F}(x, y)$. Par construction, l'Equation 4.3 impose que la combinaison de t termes soit fonctionnellement indépendante d'au moins un morceau du partage de x et un morceau du partage de y. Notre méthode décrite en Section 4.7.3 est la même pour toute valeur de d et assure la glitches-immunité à n'importe quel l'ordre t. Contrairement à la technique de Nikova *et. al.* , en suivant notre approche, il n'est pas possible de se tromper lors de la construction d'une implémentation à seuil de F, sécurisée à l'ordre supérieur en présence de glitches.

4.8 Conclusion

Dans ce chapitre, nous avons introduit deux nouvelles implémentations à seuil pour évaluer le calcul d'un monôme. L'une où la fonction de Dirac est implémentée à l'aide d'une table de correspondance, et l'autre où elle est sécurisée grâce aux implémentations à seuil. Notre idée est basée sur un partage affine de la donnée sensible, nous permettant de manipuler le masque le plus approprié pour la protéger à travers une fonction linéaire ou une fonction non-linéaire. Nos constructions sont prouvées mathématiquement sûre dans le modèle d'attaque par sondage à

l'ordre 1 en présence de glitches et peuvent s'étendre sur un grand nombre de fonctions polynomiales. La surface de nos constructions augmentera alors linéairement en fonction du nombre de puissances exécutées par l'évaluation polynomiale. En illustration, nous avons appliqué notre proposition sur la boîte-S de l'AES et nous nous sommes comparés à l'état de l'art. D'une part, nous contribuons à ce dernier par l'aspect générique de notre méthode, applicable à l'évaluation de n'importe quelle structure de substitution-permutation. D'autre part, nous donnons une implémentation à seuil détaillée de tout l'algorithme de l'AES, ce qui n'est pas courant dans les articles sur ce sujet. Enfin, nous avons codé et simulé notre méthode sur un circuit logique programmable de manière à montrer la correction de notre construction.

En travaux futurs, il serait intéressant d'attaquer notre implémentation sur un circuit électronique (e.g. FPGA) dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches. De nos jours, savoir simuler des glitches volontairement au sein d'un circuit reste un problème difficile à mettre en oeuvre pour la communauté cryptographique.

De plus, afin de contribuer à l'état de l'art concernant la recherche d'une construction sécurisée à l'ordre supérieur en présence de glitches, ce chapitre présente une nouvelle approche afin de générer une implémentation à seuil d'une fonction polynomiale. Notre méthode se base sur le travail de Carlet *et. al.* [17]. Elle diffère de celle de Nikova *et. al.* proposée originellement dans [65] par son aspect générique, que ça soit pour son application sur un grand nombre de (n,m)-fonctions ou pour une sécurité assurée à n'importe quel ordre, et sa fiabilité, dans le sens où la propriété de glitches-immunité est assurée dans tous les cas. Notre approche est en effet prouvée mathématiquement sûre dans le modèle d'attaque par sondage à l'ordre supérieur en présence de glitches. La surface de notre construction augmentera alors linéairement en fonction de la valeur de l'ordre de sécurité souhaité ainsi que du degré algébrique de la (n,m)-fonction.

Dans le chapitre suivant, nous présentons une autre contribution à l'état de l'art dans le cadre de ma thèse, à savoir l'étude de l'état de l'art sur le SM4 face à la cryptanalyse moderne et la proposition d'une nouvelle contre-mesure.

Chapitre 5

Étude et comparaison des contre-mesures du SM4 sécurisées dans le modèle d'attaque par sondage à l'ordre 1 - Proposition d'un SM4 protégé par les implémentations à seuil

Sommaire

Aujourd'hui, la Chine représente plus de 40% du commerce électronique [1] mondial [57]. Si une entreprise technologique souhaite s'établir en Chine afin de participer à ce marché en pleine expansion, il est nécessaire de connaître et respecter les lois chinoises. La législation concernant l'application de la cryptographie est définie par l'administration du commerce international cryptographique, plus connue sous son nom anglais Office of State Commercial Cryptography Administration (OSCCA). Cette dernière a été créé en 1999 par l'académie des sciences de Chine (Chinese

1. aussi appelé le *e-commerce*, il représente les différentes transactions commerciales qui se font à distance sur internet au travers d'objets numériques et digitales.

Academy of Science (CAS)) afin de tester et certifier les algorithmes standards cryptographiques commercialisés dans le pays.

Afin de remplacer les algorithmes cryptographiques publiés par le NIST aux États-Unis, l'OSCCA propose en Chine 5 algorithmes [51] : le SM2, le SM3, le SM4, le SM9 et le ZUC. Dans [57], Martinkauppi *et. al.* proposent une comparaison des performances entre ces derniers et les algorithmes cryptographiques du NIST correspondants. Les initiales "SM" correspondent à "Shāngyè Mìmǎ" en phonétique chinoise, qui veut dire "commecial cipher" en anglais.

Dans ce chapitre, nous nous concentrons sur le SM4 dont les spécifications sont décrites en anglais dans [29]. Pendant ma thèse, j'ai eu l'opportunité d'encadrer pendant 6 mois un étudiant de $4^{ème}$ année de l'INSA de Lyon, spécialisé en génie électrique. Son sujet de stage portait sur l'implémentation en assembleur des contre-mesures proposées dans l'état de l'art pour le SM4, visant une sécurité vis-à-vis du sondage à l'ordre 1. Le but était de pouvoir comparer les performances des implémentations obtenues. Grâce au partage de nos connaissances en programmation logicielle et en cryptographie, les travaux réalisés ont permis de rédiger un état de l'art et de proposer de nouvelles idées. Un article est notamment en cours de soumission pour une conférence.

Dans la Section 5.1, nous présentons le contexte dans lequel s'inscrivent nos travaux et les différentes contributions que nous avons apportées à l'état de l'art. Puis, la Section 5.2 décrit les mécanismes de chiffrement et de déchiffrement de l'algorithme SM4. Nous expliquons ensuite les différentes contre-mesures du SM4 de l'état de l'art dans la Section 5.3. La Section 5.4 est dédiée à la comparaison des performances des implémentations logicielles de ces propositions. Enfin, nous proposons dans la Section 5.5 une implémentation à seuil du SM4, sécurisée dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches.

5.1 Contexte et contributions

Alors que la Chine ouvre la voie du commerce électronique à l'échelle mondiale, il devient important pour les entreprises technologiques du monde entier de connaître les mécanismes cryptographiques des algorithmes publiés par l'OSCCA. C'est le cas d'entreprises comme STMicroelectronics où j'ai effectué ma thèse. Afin de participer au marché chinois, certains produits proposés par STMicroelectronics doivent obligatoirement intégrer une implémentation matérielle de l'algorithme cryptographique SM4.

Dans ce contexte, nous avons étudié les articles de l'état de l'art portant sur le SM4, à savoir son fonctionnement cryptographique ainsi que les propositions d'attaque ou de défense de l'algorithme. Nous avons alors fait face à plusieurs contraintes : la quantité de documents traduits du chinois à l'anglais est faible, certains articles sont très courts et ne détaillent que très peu le contenu technique, d'autres articles sont mathématiquement incorrects. Ces barrières nous ont alors poussé à redéfinir et parfois corriger les principes cryptographiques étudiés dans ces articles.

En contribution, nous centralisons dans ce chapitre les informations sur l'état de l'art du SM4. Premièrement, nous décrivons précisément la manière dont la partie non-linéaire de l'algorithme est définie mathématiquement. Les formules données dans l'état de l'art [1, 53] ne sont en effet pas toujours correctes, ce qui crée une confusion pour le lecteur. Puis, nous détaillons le mode de fonctionnement des différentes implémentations logicielles sécurisées vis-à-vis du sondage à l'ordre 1 de l'état de l'art. Nous donnons également une visibilité sur les performances de chacune d'entre elles. Enfin, il n'existe qu'un seul article [52] proposant une version du SM4 sécurisée à l'ordre 1 en présence de glitches. Nos recherches ont montré que ce papier ne donne pas d'informations correctes sur la manière de protéger le SM4 contre les attaques par observations en présence de glitches. Une deuxième contribution consiste en la proposition d'une implémentation à seuil détaillée de chaque opération du SM4. Il s'agit de la seule construction de l'état de l'art [2], prouvée mathématiquement sûre à l'ordre 1 en présence de glitches.

5.2 Description de l'algorithme SM4

L'OSCCA publie en janvier 2006 les spécifications de l'algorithme cryptographique SM4 [29], inventé par Shu-Wang Lu, afin de remplacer l'algorithme AES-128 [27] proposé par le NIST en octobre 2000 (voir Section 2.1.3). Le SM4 est standardisé et intégré en 2012 dans le gouvernement chinois. Il est utilisé par la norme WAPI [3], un standard national chinois pour la sécurité des réseaux sans fil.

5.2.1 Structure et notations du SM4

Le SM4 est un algorithme symétrique de chiffrement par blocs. Il manipule des blocs de données de taille 128 bits et chiffre un message clair grâce à une clé cryptographique originale de taille 128 bits. Ces blocs sont représentés sous la forme d'une concaténation de 4 mots de 32 bits. Le chiffrement ou le déchiffrement d'un bloc de données s'effectue en 32 tours. Chaque tour met à jour un quart d'un bloc de données interne, soit un mot de 32 bits, grâce à une fonction de tour. La structure du SM4 est donc basée sur un schéma de Feistel non-équilibré (voir Remarque 1). Les seules opérations manipulées par la fonction de tour sont des Xors entre des mots de 32 bits, des décalages circulaires sur des mots de 32 bits et des applications de boîtes-S sur des octets. De plus, chaque fonction de tour est paramétrée pour une clé de tour. Cette dernière est générée par un algorithme de cadencement de clé à partir de la clé cryptographique originale. Comme pour l'AES, la

2. dans les documents disponibles en anglais
3. WAPI correspond à "WLAN Authentication Privacy Infrastructure" en anglais où WLAN correspond à "Wireless Local Area Network" en anglais.

procédure de déchiffrement diffère de celle du chiffrement par l'utilisation des clés de tours dans l'ordre inverse et par l'application inverse des calculs effectués dans la fonction de tour. Dans la suite, nous présentons la procédure de chiffrement du SM4.

On note respectivement $x \in \mathrm{GF}(2)^{128}$ et $k \in \mathrm{GF}(2)^{128}$ le message clair et la clé secrète cryptographique originale. Comme dit précédemment, chaque bloc est défini par 4 mots de 32-bits. Le message clair correspond à $x = (x_0 \| x_1 \| x_2 \| x_3)$. Plus généralement, on dit que la fonction de tour notée F manipule un bloc interne $(x_i \| x_{i+1} \| x_{i+2} \| x_{i+3})$ et la clé de tour rk_i correspondante, pour $i \in [0..31]$. Avec ces notations, nous décrivons cette fonction dans la suite.

5.2.2 Fonction de tour du SM4

La fonction de tour du SM4 suit un schéma de Feistel non-équilibré. À chaque tour, elle permet de mettre à jour un des 4 mots d'un bloc interne. Comme illustré dans la Figure 43, elle consiste en la composition de 4 étapes que nous détaillons ci-dessous.

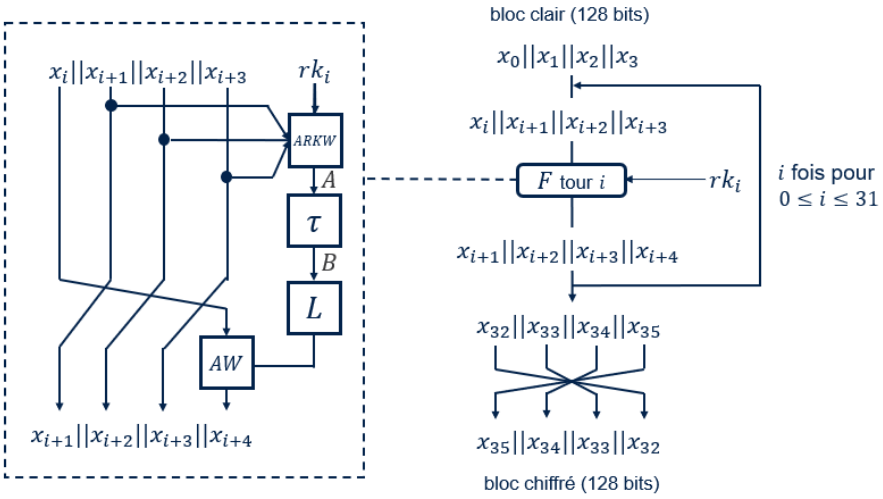


FIGURE 43 – Procédure de chiffrement du SM4

Premiers Xors. Dans un premier temps, la transformation linéaire ARKW additionne les 3 derniers mots du bloc interne manipulé au tour $i \in [0..31]$ ainsi que la clé de tour rk_i. Le résultat obtenu est noté $\mathrm{A} \in \mathrm{GF}(2)^{32}$ et défini par :

$$\mathrm{A} = \mathrm{ARKW}(x_{i+1}, x_{i+2}, x_{i+3}, rk_i) = x_{i+1} \oplus x_{i+2} \oplus x_{i+3} \oplus rk_i. \tag{5.1}$$

Transformation non-linéaire. Puis, la propriété de confusion (voir Section 2.1.2) est assurée par la fonction non-linéaire τ qui consiste en l'application de la même boîte-S sur chaque octet de A. Comme pour l'AES, la boîte-S du SM4 peut être pré-calculée avant l'exécution du SM4 et stockée dans une table de correspondance. Cette dernière est représentée dans la Figure 44 pour toutes les substitutions possibles d'un octet dans sa notation hexadécimale.

Dans les spécifications du SM4 [29], les mécanismes mathématiques de la boîte-S du SM4 ne sont pas donnés. L'état de l'art propose différentes représentations de cette boîte-S dans sa décomposition algébrique [1, 53, 89]. Une des contributions de cette thèse consiste en la recherche d'une implémentation correcte de la boîte-S. Nous avons ainsi implémenté et validé la décomposition que nous présentons dans la suite [4].

4. Ces résultats valident également la décomposition algébrique de la boîte-S du SM4 donnée par Wei *et. al.* [89].

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
5	68	6b	81	b2	71	64	da	8b	f8	eb	0f	4b	70	56	9d	35
6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	d7	cb	39	48

FIGURE 44 – Table de correspondance de la boîte-S du SM4 pour un octet en notation hexadécimale, en commençant par la ligne, puis la colonne. Par exemple, l'octet $b3$ est substitué par l'octet 45

En plus de la table de correspondance, la substitution d'un octet a de A par la boîte-S du SM4, notée SBox, peut aussi être définie par la relation algébrique suivante :

$$\mathrm{SB}ox(a) = \mathrm{AF}_{\mathrm{SM4}}(\mathrm{INV}_{\mathrm{SM4}}(\mathrm{AF}_{\mathrm{SM4}}(a))), \tag{5.2}$$

telle que :

— $\mathrm{AF}_{\mathrm{SM4}}(a)$ est une transformation affine définie par $\mathrm{AF}_{\mathrm{SM4}}(a) = \mathscr{M}_{\mathrm{SM4}}(a) \oplus \mathrm{CS}w_{\mathrm{SM4}}$,

où la matrice $\mathscr{M}_{\mathrm{SM4}}$ est définie sur $\mathrm{GF}(2)^8 \times \mathrm{GF}(2)^8$ par $\mathscr{M}_{\mathrm{SM4}} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$

et la constante $\mathrm{CST}_{\mathrm{SM4}}$ est définie par le vecteur colonne : $\mathrm{CST}_{\mathrm{SM4}} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$.

— L'opération $\mathrm{INV}_{\mathrm{SM4}}$ est définie par $\mathrm{INV}_{\mathrm{SM4}} : a \in \mathrm{GF}(2)^8 \mapsto a^{254} \in \mathrm{GF}(2^8)$, soit l'inverse multiplicatif de a dans le corps $\mathrm{GF}(2^8) \simeq \mathrm{GF}(2)[a]/(a^8 + a^7 + a^6 + a^5 + a^4 + a^2 + 1)$.

Transformation linéaire. Soit B la sortie de τ(A). La propriété de diffusion (voir Section 2.1.2) est ensuite assurée par la fonction linéaire L qui consiste en la modification de B telle que :

$$\mathrm{L}(\mathrm{B}) = \mathrm{B} \oplus (\mathrm{B} \lll 2) \oplus (\mathrm{B} \lll 10) \oplus (\mathrm{B} \lll 18) \oplus (\mathrm{B} \lll 24), \tag{5.3}$$

où $w \lll j$ représente un décalage circulaire de j positions vers la gauche sur un mot w de 32 bits.

Xor final. Enfin, la transformation linéaire AW additionne le premier mot du bloc interne (non utilisé jusque là) et le résultat obtenu au calcul précédent. Cette étape conduit à un nouveau mot de 32 bits défini par :

$$x_{i+4} = \mathrm{AW}(x_i, \mathrm{L(B)}) = x_i \oplus \mathrm{L(B)}.$$

Pour résumer, à chaque tour $i \in [0..31]$, la fonction de tour F produit le mot x_{i+4} à partir de 3 mots du bloc interne et de la clé de tour telle que :

$$x_{i+4} = \mathrm{F}(x_i, x_{i+1}, x_{i+2}, x_{i+3}, rk_i) = \mathrm{AW}(x_i, \mathrm{L}(\tau(\mathrm{ARKW}(x_{i+1}, x_{i+2}, x_{i+3}, rk_i)))).$$

À la fin du tour i, le bloc interne correspondant à l'entrée du tour $i+1$ est défini par :

$$(x_{i+1} \| x_{i+2} \| x_{i+3} \| x_{i+4}).$$

À la fin de la procédure de chiffrement (*i.e.* des 32 tours), nous obtenons le bloc $(x_{32} \| x_{33} \| x_{34} \| x_{35}) \in \mathrm{GF}(2)^{128}$. Le message chiffré de 128 bits est alors défini par ces derniers mots dans le sens inverse, soit le bloc $(x_{35} \| x_{34} \| x_{33} \| x_{32}) \in \mathrm{GF}(2)^{128}$.

Il reste à discuter de la diversification de la clé secrète dans le SM4. On décrit dans la suite la procédure permettant de générer une clé de tour différente pour chaque tour de l'algorithme.

5.2.3 Algorithme de cadencement de clé du SM4

Pour chaque tour $i \in [0..31]$, l'algorithme de cadencement de clé du SM4 prend en entrée la clé cryptographique originale $k \in \mathrm{GF}(2)^{128}$ ainsi que deux autres paramètres publics, le bloc $\mathrm{FK} \in \mathrm{GF}(2)^{128}$ et les mots $\mathrm{CK}_i \in \mathrm{GF}(2)^{32}$, disponibles dans les spécifications du SM4 [29]. Il retourne 32 clés de tour de 32 bits en suivant une procédure similaire à celle du chiffrement, comme illustré en Figure 45 et détaillé ci-dessous.

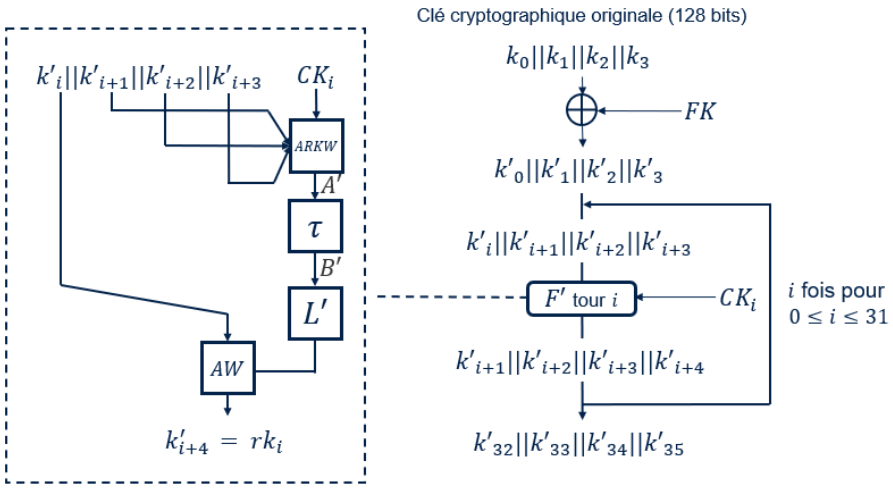


FIGURE 45 – Procédure de l'algorithme de cadencement de clé du SM4

Initialisation. La clé cryptographique originale k et le paramètre public FK sont additionnés. Le résultat obtenu est noté $k' \in \mathrm{GF}(2)^{128}$ et défini par : $k' = k \oplus \mathrm{FK}$. Comme pour la procédure de chiffrement, un bloc de 128 bits est représenté sous la forme d'une concaténation de 4 mots de 32 bits. Nous avons :

$$k' = k'_0 \| k'_1 \| k'_2 \| k'_3 = k_0 \oplus \mathrm{FK}_0 \| k_1 \oplus \mathrm{FK}_1 \| k_2 \oplus \mathrm{FK}_2 \| k_3 \oplus \mathrm{FK}_3. \tag{5.4}$$

À partir de $(k'_0 \| k'_1 \| k'_2 \| k'_3) \in \mathrm{GF}(2)^{128}$, il est possible de construire les 32 clés de tour grâce à la fonction de tour suivante.

Premiers Xors. La transformation linéaire ARKW additionne les 3 derniers mots du bloc interne manipulé au tour $i \in [0..31]$ au paramètre public $\mathrm{CK}_i \in \mathrm{GF}(2)^{32}$. Le résultat obtenu est noté $\mathrm{A}' \in \mathrm{GF}(2)^{32}$ et est défini par :

$$\mathrm{A}' = \mathrm{ARKW}(k'_{i+1}, k'_{i+2}, k'_{i+3}, \mathrm{CK}_i) = k'_{i+1} \oplus k'_{i+2} \oplus k'_{i+3} \oplus \mathrm{CK}_i.$$

Transformation non-linéaire. Puis, la propriété de confusion (voir Section 2.1.2) est assurée par la même fonction non-linéaire τ définie pour la procédure de chiffrement du SM4.

Transformation linéaire. Soit B' la sortie de $\tau(\mathrm{A}')$. La propriété de diffusion (voir Section 2.1.2) est ensuite assurée par la fonction linéaire L' qui consiste en la modification de B' telle que :

$$\mathrm{L}'(\mathrm{B}') = \mathrm{B}' \oplus (\mathrm{B}' \lll 13) \oplus (\mathrm{B}' \lll 23), \tag{5.5}$$

où $\lll j$ représente un décalage circulaire de j positions vers la gauche sur un mot de 32 bits.

Xor final. Enfin, la transformation linéaire AW additionne le premier mot du bloc de clé interne (non utilisé jusque là) et le résultat obtenu au calcul précédent pour tout $i \in [0..31]$. Cette étape conduit à un nouveau mot de 32 bits défini par la relation suivante :

$$k'_{i+4} = \mathrm{AW}(k'_i, \mathrm{L}'(\mathrm{B}')) = k'_i \oplus \mathrm{L}'(\mathrm{B}').$$

Pour résumer, à chaque tour $i \in [0..31]$, la fonction de tour F' produit la clé de tour rk_i à partir de 3 mots du bloc de clé interne et du paramètre de sécurité constant CK_i telle que :

$$rk_i = k'_{i+4} = \mathrm{F}'(k'_i, k'_{i+1}, k'_{i+2}, k'_{i+3}, \mathrm{CK}_i) = \mathrm{AW}(k'_i, \mathrm{L}'(\tau(\mathrm{ARKW}(k'_{i+1}, k'_{i+2}, k'_{i+3}, \mathrm{CK}_i)))). \tag{5.6}$$

À la fin du tour i, le bloc interne correspondant à l'entrée du tour $i+1$ est défini par :

$$(k'_{i+1} \| k'_{i+2} \| k'_{i+3} \| k'_{i+4}).$$

Depuis sa publication, le SM4 tel que nous l'avons décrit fait face à de nombreuses attaques par observations à l'ordre 1 [5, 53, 4, 86]. Ainsi, plusieurs contre-mesures ont été proposées dans l'état de l'art afin de sécuriser le calcul d'une donnée sensible au sein de l'algorithme SM4. Dans la section suivante, nous détaillons ces schémas de défense.

5.3 Contre-mesures proposées dans l'état de l'art

Dans cette section, nous décrivons les contre-mesures proposées par Duan *et. al.* [30], Chen *et. al.* [20] et Pu *et. al.* [70] afin de sécuriser le chiffrement d'une donnée sensible dans le SM4. Ces schémas sont basés sur du masquage additif et visent une sécurité à l'ordre 1 vis-à-vis du sondage. Nos recherches ont permis de corriger certaines erreurs faites par les auteurs et de donner plus de détails sur les mécanismes de défense étudiés.

Nous décrivons les différents types de masquage des 3 travaux de notre étude [30, 20, 70] [5] dans la suite. Nous détaillons également une technique de dissimulation (voir Section 2.4.1) proposée dans [20]. Comme le masquage suffit à se protéger d'un adversaire dans le modèle d'attaque par sondage à l'ordre 1, la méthode de dissimulation que nous décrivons peut être combinée au masquage additif de n'importe quelle contre-mesure afin de rendre plus difficile les attaques par sondage à l'ordre $t > 1$.

5. Ces articles sont les seuls traduits du chinois à l'anglais de l'état de l'art.

5.3.1 Masquage additif du SM4

Masquage additif des parties linéaires du SM4

Les parties linéaires du chiffrement du SM4 sont les fonctions ARKW, L et AW (voir Section 5.2.2). Comme ces fonctions opèrent sur des mots de 32 bits, Duan *et. al.* [30] proposent de masquer la sortie de chacune d'entre elles par un masque additif fixé noté M, généré aléatoirement dans $GF(2)^{32}$. Chen *et. al.* [20] proposent quant à eux de générer plusieurs masques de 32 bits afin de masquer additivement les sorties des parties linéaires du SM4. Nous expliquons l'idée de Duan *et. al.* ci-dessous, le raisonnement est le même pour le schéma de masquage de Chen *et. al.* .

Soit $(B \oplus M)$ la sortie masquée de la fonction τ (*i.e.* l'entrée de la fonction L). Afin de masquer la sortie de L, Duan *et. al.* proposent d'ajouter au résultat de $L(B \oplus M)$ la valeur $L(M) \oplus M = (M \lll 2) \oplus (M \lll 10) \oplus (M \lll 18) \oplus (M \lll 24)$. Par linéarité de L, il en résulte alors la donnée sensible masquée additivement, *i.e.* $L(B) \oplus M$. Pour la valeur de M connue, cette solution permet de calculer une seule fois la valeur de $L(M) \oplus M$ et de l'ajouter à $L(B \oplus M)$ quand cela est nécessaire lors du chiffrement. Nous privilégions ce schéma de masquage de la fonction linéaire L dans la suite. En remarque, le raisonnement est le même pour le masquage de la fonction linéaire L′.

Comme pour l'AES, la partie la plus délicate à masquer dans le SM4 est le calcul de la partie non-linéaire. Dans la suite, nous étudions les techniques de masquage additif appliquées à la boîte-S du SM4. Nous avons vu en Section 5.2.2 que cette dernière peut être représentée sous la forme d'une LUT ou définie par sa décomposition algébrique. Les travaux [30] et [20] proposent un masquage de la LUT tandis que les auteurs de [70] se concentrent sur la protection des calculs non-linéaires de la boîte-S. Nous détaillons leurs méthodes et illustrons les propositions de l'état de l'art dans la suite.

Masquage additif de la partie non-linéaire du SM4 : boîte-S en tant que LUT

Duan *et. al.* emploient la solution proposée dans [3] présentée en Équation 2.6 afin de masquer additivement la LUT de la boîte-S du SM4. Soient a un octet de la valeur sensible de 32 bits et $M[b]$ (pour $b \in [0..3]$) un octet d'un masque $M \in GF(2)^{32}$, généré pour protéger les opérations linéaires précédentes (e.g. les Xors). Les auteurs de [30] proposent de pré-calculer les 4 nouvelles LUT notées $SBox'_b$ telles que :

$$SBox'_b(a) = SBox(a \oplus M[b]) \oplus M[b]. \tag{5.7}$$

Chen *et. al.* [20] proposent quant à eux de masquer l'entrée et la sortie de ces nouvelles LUT par deux mots de 32 bits différents M_{in} et M_{out}, respectivement, tels que :

$$SBox'_b(a) = SBox(a \oplus M_{in}[b]) \oplus M_{out}[b]. \tag{5.8}$$

La fonction non-linéaire τ du SM4 est alors remplacée par une autre notée τ_M, qui appelle les 4 LUTs $SBox'_b$ décrites ci-dessus, en fonction de la méthodes de Duan *et. al.* ou celle de Chen *et. al.* . Dans la suite, nous illustrons leurs propositions.

Illustration des propositions de Duan *et. al.* [30] et Chen *et. al.* [20]

Duan et. al. *[30].* L'algorithme original proposé par Duan *et. al.* conduit à des questions sur la correction de leur schéma de masquage. En effet, pour un message clair aléatoire dans $GF(2)^{128}$, leur algorithme ne permet pas de retrouver le message chiffré correspondant pour un chiffrement de SM4. Nous pensons que cela est notamment dû au fait que les auteurs suppriment la clé de tour au milieu du calcul de la fonction de tour (voir ligne 10 de la Figure 4 dans [30]). Par ailleurs, nous pointons une vulnérabilité due à l'utilisation d'un unique masque M tout au long de l'algorithme. Par exemple, le bloc interne à l'entrée du deuxième tour correspond à $(x_2, x_3, x_4 \oplus M, x_5 \oplus M)$. Duan

et. al. proposent de masquer l'opération ARKW par M de la manière suivante : $x_3 \oplus x_4 \oplus M \oplus x_5 \oplus M \oplus rk_2$. Comme x_3 est connu (*i.e.* mot du message clair), un attaquant pourrait être capable d'avoir de l'information sur $x_4 \oplus x_5$ et de monter une attaque par observations afin de retrouver la clé de tour rk_2 [86]. En effet, il est important que chaque mot x_i du bloc interne soit indépendant de plus de 128 bits de clé afin de ne pas la retrouver. Cette condition est atteinte à partir du cinquième tour (*i.e.* pour $i \geq 4$).

Inspiré par leur travail, nous proposons dans l'Algorithme 5 une version correcte et sécurisée dans le modèle d'attaque par sondage à l'ordre 1 de leur schéma de masquage pour un chiffrement SM4, utilisant deux masques $M, M' \in GF(2)^{32}$ fixes pour tous les calculs. On se réfère à la méthode présentée en Section 5.3.1 pour le masquage de la fonction linéaire L et à l'Equation 5.7 pour le calcul de la fonction non-linéaire τ_M.

Algorithme 5 Notre version de l'algorithme de chiffrement masqué du SM4 de Duan *et. al.* [30]

Entrée(s) : $x = (x_0, x_1, x_2, x_3) \in (GF(2)^{32})^4, rk_{i_{0 \leq i \leq 31}} \in GF(2)^{32}, M, M' \in GF(2)^{32}$
Sortie(s) : $(x_{35} \oplus M, x_{34} \oplus M, x_{33} \oplus M, x_{32} \oplus M) \in (GF(2)^{32})^4$

pour $(i = 0; i < 32; i++)$ **faire**
  si $i == 0$ **alors**
    $x_{i+4} = ARKW(x_{i+1}, x_{i+2}, x_{i+3}, M) \oplus rk_i$
  si $i == 1$ **alors**
    $x_{i+4} = ARKW(x_{i+1}, x_{i+2}, x_{i+3}, rk_i)$
  si $(i == 2 \,||\, i == 3)$ **alors**
    $x_{i+4} = ARKW(x_{i+1}, M', x_{i+2}, x_{i+3}) \oplus rk_i$
  sinon
    $x_{i+4} = ARKW(x_{i+1}, x_{i+2}, x_{i+3}, rk_i)$
  $x_{i+4} = \tau_M(x_{i+4})$
  si $i < 4$ **alors**
    si $i == 0 \,||\, i == 1$ **alors**
      $m = L(M) \oplus M$
    sinon
      $m = L(M') \oplus M$
    $x_{i+4} = L(x_{i+4}) \oplus m$
    $x_{i+4} = AW(x_{i+4}, x_i)$
  sinon
    $x_{i+4} = AW(L(x_{i+4}), x_i)$
    $x_{i+4} = x_{i+4} \oplus L(M)$

retourner $(x_{35} \oplus M, x_{34} \oplus M, x_{33} \oplus M, x_{32} \oplus M)$

Les blocs "si-alors" de l'Algorithme 5 permettent d'éviter un démasquage involontaire de la donnée sensible. Pour illustrer les différentes situations, nous représentons les 4 premiers tours du SM4 pour $i = 0, 1, 2$ et 3 dans les Figures 46,47,48, et 49, respectivement. La Figure 50 correspond au mécanisme de chiffrement d'un bloc interne pour les tours suivants (*i.e.* à partir de $i = 4$). Les données non-encadrées représentent des mots de 32 bits tandis que les cadres correspondent à des opérations opérant sur des mots de 32 bits.

Remarque 16. *Comme Duan* et. al. *ne donnent aucune indication sur le masquage de l'algorithme de cadencement de clé, nous avons décidé dans notre étude de le protéger en suivant une approche similaire à celle présentée dans l'Algorithme 5.*

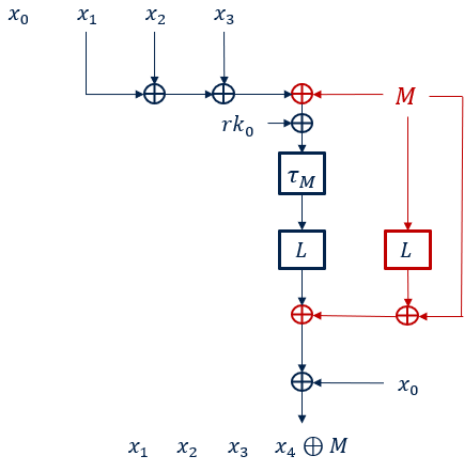


FIGURE 46 – Notre schéma de masquage additif de l'algorithme de chiffrement du SM4 pour le premier tour (*i.e.* $i = 0$), basé sur Duan *et. al.* [30]

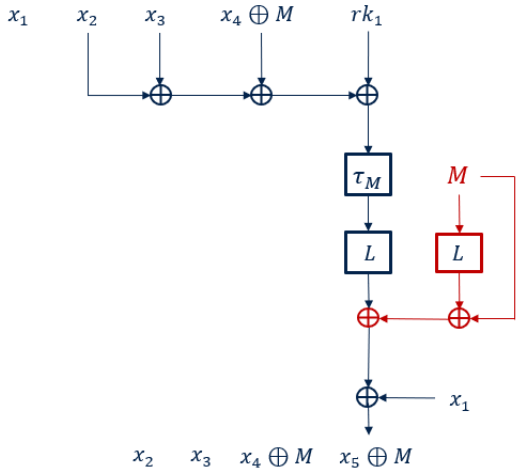


FIGURE 47 – Notre schéma de masquage additif de l'algorithme de chiffrement du SM4 pour le deuxième tour (*i.e.* $i = 1$), basé sur Duan *et. al.* [30]

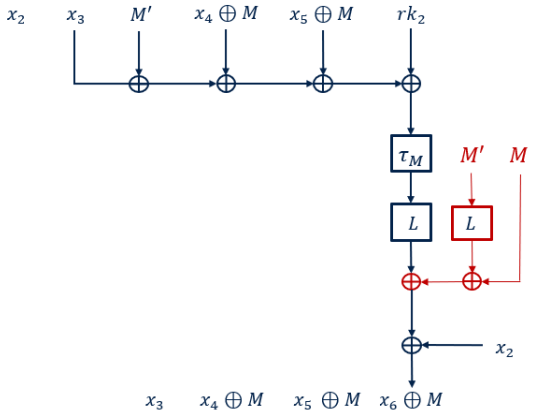


FIGURE 48 – Notre schéma de masquage additif de l'algorithme de chiffrement du SM4 pour les troisième tour (*i.e.* $i = 2$), basé sur Duan *et. al.* [30]

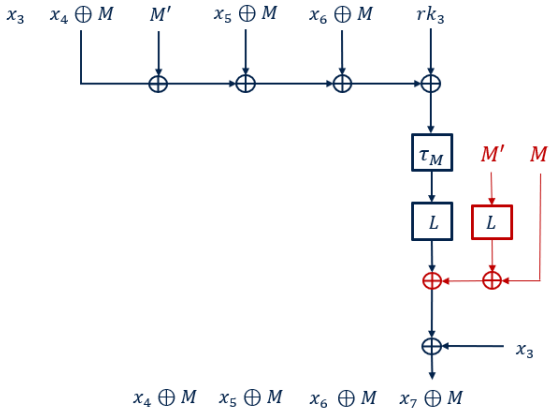


FIGURE 49 – Notre schéma de masquage additif de l'algorithme de chiffrement du SM4 pour le quatrième tour (*i.e.* $i = 3$), basé sur Duan *et. al.* [30]

Chen et. al. *[20]*. Le masquage additif de l'algorithme de chiffrement du SM4 de Chen *et. al.* diffère de celui de Duan *et. al.* par la génération de plusieurs masques de 32 bits. En effet, la procédure de masquage débute par la génération d'un masque noté $\mathrm{M}_0 = (\mathrm{M}_{0,j})_{0 \le j \le 4} \in (\mathrm{GF}(2)^{32})^4$. Puis, à chaque tour $i \in [0..31]$, les auteurs génèrent un masque aléatoire de 128 bits, noté M_{i+1}, et en déduisent 4 masques de 32 bits, notés $\mathrm{M}_{i+1,j}$ pour $0 \le j \le 4$, permettant de sécuriser les fonctions linéaires (*i.e.* ARKW, AW et L). On se réfère à la méthode présentée en Section 5.3.1 pour les détails de masquage de L. Par ailleurs, en correspondance avec l'Equation 5.8, deux masques $\mathrm{M}_{in}, \mathrm{M}_{out} \in \mathrm{GF}(2)^{32}$ sont calculés et fixés au début de l'exécution du SM4 dans le but de protéger l'entrée et la sortie de la LUT de la boîte-S. Nous illustrons le premier tour (*i.e.* $i = 0$) du schéma de masquage additif de Chen *et. al.* dans la Figure 51. Le raisonnement pour les tours suivants est similaire au schéma de masquage de Duan *et. al.* (voir Figures 47 à 50). Comme précédemment, dans la Figure 51, les données non-encadrées représentent des mots de 32 bits tandis que les cadres correspondent à des opérations opérant sur des mots de 32 bits. Les protections des parties linéaires sont représentées par des flèches pleines, celles des parties non-linéaires par des flèches en pointillés.

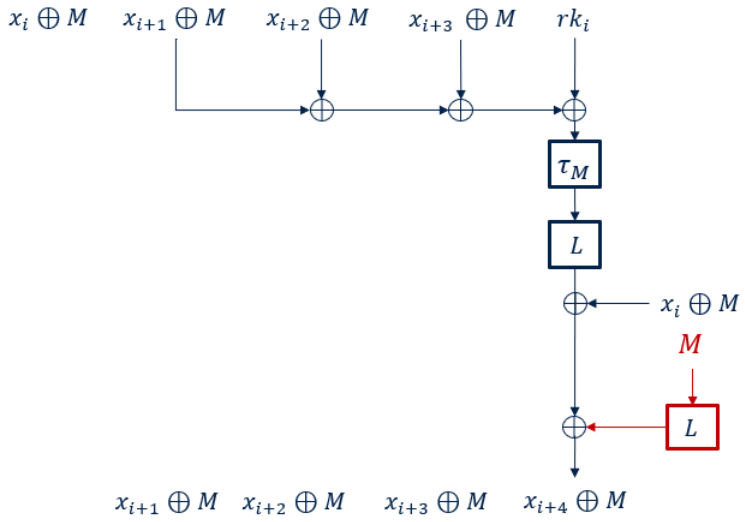


FIGURE 50 – Notre schéma de masquage additif de l'algorithme de chiffrement du SM4 pour les tours $4 \leq i \leq 31$, basé sur Duan *et. al.* [30]

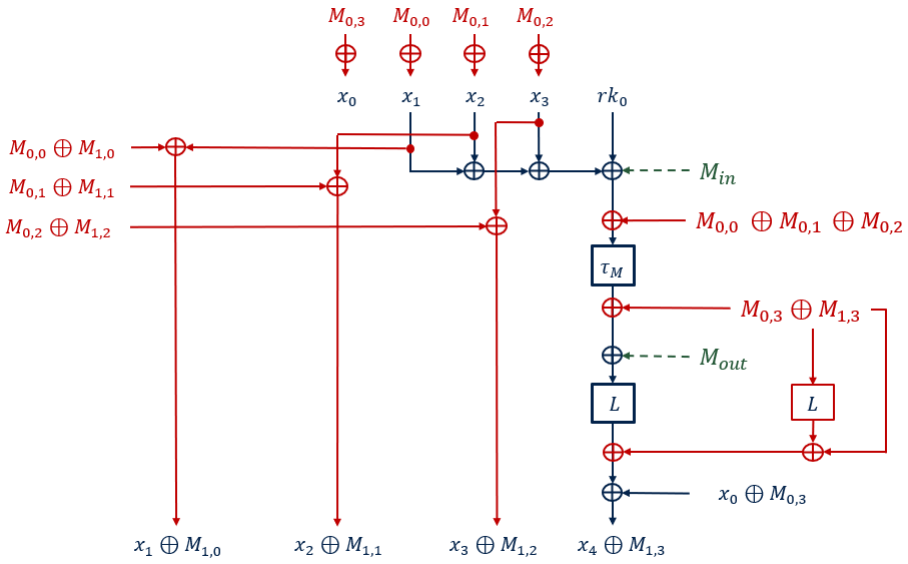


FIGURE 51 – Schéma de masquage additif de l'algorithme de chiffrement du SM4 pour le premier tour (*i.e.* $i = 0$) - Chen *et. al.* [20]

Dans la suite, nous expliquons une deuxième manière d'appliquer du masquage additif sur la partie non-linéaire du SM4.

Masquage additif de la partie non-linéaire du SM4 : boîte-S dans sa décomposition algébrique

Dans l'Équation 5.2, la boîte-S du SM4 est définie par sa décomposition algébrique. Elle peut s'exprimer par des opérations affines (*i.e.* AF_{SM4}), linéaires (*i.e.* des Xors) et une fonction non-linéaire (*i.e.* INV_{SM4}). Nous avons vu précédemment comment appliquer du masquage additif sur les fonctions linéaires [6] du SM4. Afin de masquer efficacement la partie non-linéaire de l'algorithme, Pu *et. al.* [70] procèdent de la manière suivante :

— Tout d'abord, il est important de minimiser la complexité de masquage des opérations non-linéaires (voir Section 4.7.2). Cela revient à minimiser le nombre de multiplications dans le

6. Les fonctions affines sont masquées de la même façon.

calcul de $\text{INV}_{\text{SM4}} : a \in \text{GF}(2^8) \mapsto a^{254} \in \text{GF}(2^8)$. Dans [74] Rivain et Prouff proposent une manière efficace de décomposer la fonction INV_{AES} en 4 multiplications et 3 exponentiations linéaires. Leur technique peut s'appliquer à la fonction INV_{SM4}.

— Puis, chaque opération du calcul de INV_{SM4} doit être masquée. Les exponentiations linéaires sont accessibles via une table de 2^8 mots de 8 bits additivement masquée par un masque fixe définis sur $\text{GF}(2)^8$. Les multiplications sont quant à elles protégées par la proposition de Pu *et. al.* [70] qui consiste en un *masquage matriciel additif* (défini par Wang *et. al.* [87] et expliqué ci-dessous) de la multiplication entre deux octets dans le corps d'inversion du SM4.

Remarque 17. *Le masquage des multiplications de la fonction* INV_{SM4} *peut aussi être réalisé en adaptant le schéma ISW [42] (décrit en Section 3.2.4) dans* $\text{GF}(2)$ *sur ces multiplications entre deux octets. Une sécurité vis-à-vis du sondage à l'ordre* 1 *est assurée en prenant le paramètre* $d \geq 2$*, soit en générant un partage additif en* 2*-morceaux des deux octets d'entrée à multiplier.*

Dans la suite, nous présentons le masquage matriciel additif de Wang *et. al.* [87] ainsi que son application au SM4 (*i.e.* la contre-mesure de Pu *et. al.* [70]).

Masquage matriciel additif [87]

Dans l'état de l'art, de nombreux auteurs se sont concentrés sur l'amélioration de la résistance des implémentations cryptographiques dans le cas où les mesures de courant d'un attaquant contiendraient peu de bruit. En effet, dans un tel scénario, un masquage additif classique d'une donnée sensible possède une structure algébrique simple (voir Equation 2.3) ce qui peut être vu comme une faiblesse. Par exemple, si l'adversaire connaît un bit de chaque morceau de la donnée sensible, il sera capable de calculer le bit correspondant de cette donnée. Des solutions ont alors été proposées [6, 33, 38, 78] afin d'augmenter la complexité algébrique des opérations dans les schémas de masquage. Ces solutions sont plus lourdes à implémenter (surtout pour une sécurité vis-à-vis du sondage aux ordres supérieurs), mais assurent une sécurité plus importante. En effet, cette fois, si l'adversaire connaît un bit d'information de chaque morceau de la donnée sensible, il ne sera pas capable de construire le bit d'information correspondant de cette donnée.

Dans ce contexte, Wang *et. al.* [87] proposent alors un masquage hybride en combinant un masquage par produit matriciel (pour améliorer la complexité algébrique du schéma) et un masquage additif classique (pour masquer efficacement aux ordres supérieurs [78]). Leur proposition est inspirée du travail de Von Willich [85] et est appelée *masquage matriciel additif*. Elle convient bien aux blocs chiffrés où les calculs peuvent s'effectuer par un *découpage de bits* (*bitslicing* en anglais [8]) des données (e.g. les calculs du SM4). Le découpage de bits est une méthode qui découpe des données de n bits en n éléments de 1 bit et réalise des opérations bit-à-bit sur ces éléments en parallèle. Cette technique accélère la vitesse d'exécution d'un chiffrement par bloc. Par exemple, si on prend deux vecteurs $a, b \in \text{GF}(2)^n$ tels que $a[i]$ (resp. $b[i]$) représente le $i^{ème}$ bit de a (resp. b), la multiplication dans $\text{GF}(2)$ notée ci-dessous $\otimes$ entre a et b s'effectue telle que :

$$a \otimes b = \begin{pmatrix} a[0] \otimes b[0] \\ \vdots \\ a[n-1] \otimes b[n-1] \end{pmatrix}.$$

Dans la suite, nous expliquons le fonctionnement du masquage matriciel additif du produit de deux vecteurs définis dans $\text{GF}(2)^n$. Ce masquage assure une sécurité vis-à-vis du sondage à l'ordre $t \geq 1$.

Soient x et y deux éléments définis dans le corps $\text{GF}(2^n)$. Nous notons $\cdot$ la multiplication dans le corps $\text{GF}(2^n)$ et $\times$ le produit matriciel sur $\text{GF}(2)$. Quand cela est nécessaire, un élément du corps

$\mathrm{GF}(2^n)$ sera représenté sous la forme d'un vecteur de taille n sur $\mathrm{GF}(2)$. Réciproquement, un vecteur de $\mathrm{GF}(2)^n$ sera vu comme un élément du corps $\mathrm{GF}(2^n)$ quand l'opération $\cdot$ doit lui être appliquée. Wang *et. al.* introduisent une matrice inversible A de taille $n \cdot n$, constante et fixée à chaque exécution d'un chiffrement par bloc. Les auteurs proposent de découper x en $d \geq 2t+1$ (application du Théorème 2) morceaux x_i ($i \in [1..d]$) tels que les morceaux $\{x_2, ..., x_d\}$ sont générés uniformément aléatoirement dans $\mathrm{GF}(2^n)$ et $x_1 = \mathrm{A}^{-1} \times (x \oplus x_2 \oplus ... \oplus x_d)$. Le raisonnement est le même pour le découpage de y. Nous obtenons les partages additifs en d-morceaux $\mathbf{x} = ((\mathrm{A} \times x_1), x_2, ..., x_d)$ de x et $\mathbf{y} = ((\mathrm{A} \times y_1), y_2, ..., y_d)$ de y tels que les fonctions de reconstruction des entrées sont définies par les équations suivantes :

$$rec_{add}(\mathbf{x}) = (\mathrm{A} \times x_1) \oplus x_2 \oplus ... \oplus x_d = x$$
$$rec_{add}(\mathbf{y}) = (\mathrm{A} \times y_1) \oplus y_2 \oplus ... \oplus y_d = y.$$

Nous nous concentrons sur la multiplication de $x \cdot y$. Nous avons alors la décomposition suivante :

$$\begin{aligned} x \cdot y &= ((\mathrm{A} \times x_1) \oplus x_2 \oplus ... \oplus x_d) \cdot ((\mathrm{A} \times y_1) \oplus y_2 \oplus ... \oplus y_d) \\ &= ((\mathrm{A} \times x_1) \cdot (\mathrm{A} \times y_1)) \oplus ((\mathrm{A} \times x_1) \cdot y_2) \oplus ... \oplus ((\mathrm{A} \times x_1) \cdot y_d) \\ &\quad \oplus (x_2 \cdot (\mathrm{A} \times y_1)) \qquad \oplus (x_2 \cdot y_2) \qquad \oplus ... \oplus (x_2 \cdot y_d) \\ &\quad ... \\ &\quad \oplus (x_d \cdot (\mathrm{A} \times y_1)) \qquad \oplus (x_d \cdot y_2) \qquad \oplus ... \oplus (x_d \cdot y_d). \end{aligned} \tag{5.9}$$

Le but est de construire une réalisation d-masquée (e.g. $\mathbf{z}$) du calcul $x \cdot y$ présenté en Equation 5.9. Cette implémentation doit être correcte et sécurisée vis-à-vis du sondage à l'ordre souhaité. Comme le masquage matriciel (ici par la matrice A) n'existe que dans $\mathrm{GF}(2)$, Wang *et. al.* décomposent chaque termes de $x \cdot y$ sous la forme de *produit de Kronecker*, noté $\odot$ dans la suite. On note également $x[i]$ le $i^{ème}$ élément de x vu comme un vecteur dans $\mathrm{GF}(2)^n$ ($0 \leq i \leq n-1$). Le produit de Kronecker entre deux vecteurs $x, y \in \mathrm{GF}(2)^n$ forme un vecteur de taille n^2 tel que :

$$x \odot y = \begin{pmatrix} x[0] \cdot y \\ \vdots \\ x[n-1] \cdot y \end{pmatrix},$$

où $\cdot$ désigne ici le produit scalaire sur $\mathrm{GF}(2)^n$.

Soient deux matrices P et Q de taille $p \cdot p'$ et $q \cdot q'$, respectivement. On note $\mathrm{P}(i, j)$ l'élément de P correspondant au croisement entre la ligne i et la colonne j. Le produit de Kronecker entre P et Q forme une matrice de taille $pq \cdot p'q'$ telle que :

$$\mathrm{P} \odot \mathrm{Q} = \begin{pmatrix} \mathrm{P}(1,1) \times \mathrm{Q} & \dots & \mathrm{P}(1,p) \times \mathrm{Q} \\ \dots & \dots & \dots \\ \mathrm{P}(p',1) \times \mathrm{Q} & \dots & \mathrm{P}(p',p) \times \mathrm{Q} \end{pmatrix}.$$

Le produit de Kronecker est également associatif. Pour quatre matrices P, Q, R et S de tailles quelconques nous avons :

$$(\mathrm{P} \odot \mathrm{Q}) \times (\mathrm{R} \odot \mathrm{S}) = (\mathrm{P} \odot \mathrm{R}) \times (\mathrm{Q} \odot \mathrm{S}). \tag{5.10}$$

Wang *et. al.* représentent alors la multiplication $x \cdot y$ en une décomposition de produits de Kronecker telle que :

$$x \cdot y = \mathrm{J} \times (x \odot y), \tag{5.11}$$

où J est une matrice de taille $n \cdot n^2$ indépendante de x et y et générée à partir du polynôme irréductible de l'algorithme manipulé (e.g. le polynôme $a^8 + a^7 + a^6 + a^5 + a^4 + a^2 + 1$ pour le SM4).

Les termes de l'Equation 5.9 se partitionnent en 4 groupes tels que :

— 1^{er} groupe : $(\mathrm{A} \times x_i) \cdot (\mathrm{A} \times y_j)$ si $i = j = 1$
— 2^e groupe : $(\mathrm{A} \times x_i) \cdot y_j$ si $i = 1, j \in [2..d]$
— 3^e groupe : $x_i \cdot (\mathrm{A} \times y_j)$ si $j = 1, i \in [2..d]$
— 4^e groupe : $x_i \cdot y_j$ sinon.

Soit E la matrice identité de taille $n \cdot n$. D'après les Équations 5.10 et 5.11, nous avons la correspondance suivante pour les trois premiers groupes [7] :

— 1^{er} groupe : $\mathrm{J} \times (\mathrm{A} \odot \mathrm{A}) \times (x_i \odot y_j)$ si $i = j = 1$
— 2^e groupe : $\mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_i \odot y_j)$ si $i = 1, j \in [2..d]$
— 3^e groupe : $\mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_i \odot y_j)$ si $j = 1, i \in [2..d]$

Wang *et. al.* décomposent ainsi le produit $x \cdot y \in \mathrm{GF}(2^n)$ tel que :

$$\begin{aligned} x \cdot y &= ((\mathrm{A} \times x_1) \oplus x_2 \oplus ... \oplus x_d) \times ((\mathrm{A} \times y_1) \oplus y_2 \oplus ... \oplus y_d) \\ &= \mathrm{J} \times (\mathrm{A} \odot \mathrm{A}) \times (x_1 \odot y_1) \oplus \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_2) \oplus ... \oplus \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_d) \\ &\oplus \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_2 \odot y_1) \qquad \oplus (x_2 \cdot y_2) \qquad \oplus ... \oplus (x_2 \cdot y_d) \\ &... \\ &\oplus \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_d \odot y_1) \qquad \oplus (x_d \cdot y_2) \qquad \oplus ... \oplus (x_d \cdot y_d). \end{aligned} \tag{5.12}$$

Afin d'être correcte (voir étape 2 ci-dessous), la réalisation d-masquée $\mathbf{z}$ de $x \cdot y$ réalisée par Wang *et. al.* implique de masquer par A^{-1} les termes de l'Equation 5.12 tels que $i = 1$ ou $j = 1$ (*i.e.* les termes des trois premiers groupes). Wang *et. al.* introduisent alors une matrice B contenant des vecteurs $b_{i,j}$ de taille n définis par :

$$b_{i,j} = \begin{cases} \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{A}) \times (x_i \odot y_j) & \text{si } i = j = 1 \\ \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_i \odot y_j) & \text{si } i = 1, j \in [2..d] \\ \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_i \odot y_j) & \text{si } j = 1, i \in [2..d] \\ x_i \cdot y_j & \text{sinon.} \end{cases} \tag{5.13}$$

Remarque 18. *La représentation d'une multiplication dans* $\mathrm{GF}(2^n)$ *sous la forme de produits de Kronecker permet de pouvoir pré-calculer les valeurs de* $\mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{A}), \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E})$ *et* $\mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A})$ *avant l'exécution de l'algorithme. Wang* et. al. *mettent ainsi en valeur la rapidité d'exécution de leur contre-mesure.*

Puis, inspiré du schéma ISW présenté en Section 3.2.4, les auteurs construisent un partage additif $\mathbf{z}$ en d-morceaux de $x \cdot y$ à partir des partages $\mathbf{x}$ et $\mathbf{y}$. Leur construction est sécurisée vis-à-vis du sondage à l'ordre souhaité. Les éléments $(z_1, ..., z_d)$ de $\mathbf{z}$ sont formés à partir des vecteurs $b_{i,j}$ de B en suivant les étapes suivantes :

— **Étape 1 :** Pour tout $1 \leq i < j \leq d$, on pose $r_{i,i} = b_{i,i}$ et le vecteur $r_{i,j}$ de taille n est tiré aléatoirement et uniformément dans $\mathrm{GF}(2)^n$. Le vecteur $r_{j,i}$ est ensuite calculé tel que $r_{j,i} = b_{j,i} \oplus (r_{i,j} \oplus b_{i,j})$. L'utilisation des parenthèses indique l'ordre dans lequel les opérations sont exécutées, ce qui est primordial pour la sécurité du schéma de masquage. De plus, après avoir calculé $r_{j,i}$, si $i = 1$, alors le vecteur $r_{i,j}$ est multiplicativement masqué par la matrice A tel que $r_{i,j} = \mathrm{A} \times r_{i,j}$.
— **Étape 2 :** Les vecteurs de sorties $(z_1, ..., z_d)$ de la réalisation d-masquée $\mathbf{z}$ sont calculés tels que pour $1 \leq i \leq d$, $z_i = \bigoplus_j r_{j,i}$. Afin de vérifier la correction, nous avons la fonction de reconstruction de sortie suivante : $rec_{add}(\mathbf{z}) = (\mathrm{A} \times z_1) \oplus z_2 \oplus ... \oplus z_d = x \cdot y$.

7. Le 4^e groupe reste inchangé.

Pour un ordre t de sécurité et un nombre de morceaux d choisis tel que $d \geq 2t+1$, l'observation de t éléments de $\mathbf{z}$ ne donne pas d'information sur x ou y, et donc sur la donnée sensible $x \cdot y$. La construction de $\mathbf{z}$ est donc sécurisée vis-à-vis du sondage à l'ordre t.

Dans la suite, nous utilisons la représentation de Rivain *et. al.* [74] afin d'illustrer la construction des éléments de la réalisation 3-masquée $\mathbf{z}$ de la multiplication entre deux octets dans le corps d'inversion du SM4, soit la contre-mesure présentée dans le papier de Pu *et. al.* [70].

Article de Pu et. al. *[70] : application du masquage matriciel additif de Wang* et. al. *[87] sur le SM4*

Nous nous référons à la méthode présentée dans l'exemple 2 en Section 3.2.4. Nous nous plaçons dans le corps d'inversion du SM4, *i.e.* $n = 8$, et nous visons une sécurité vis-à-vis du sondage à l'ordre 1, *i.e.* $d = 3$ (d'après le Théorème 2). Pour deux octets x et y, la matrice B contient les termes de la multiplication de $x \cdot y \in \mathrm{GF}(2^8)$ sous la forme de produits de Kronecker comme décrits par Wang *et. al.* (voir Equation 5.13). Nous obtenons :

$$\begin{aligned}\mathrm{B} &= \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \\ &= \begin{pmatrix} \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{A}) \times (x_1 \odot y_1) & \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_2) & \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_3) \\ \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_2 \odot y_1) & x_2 \cdot y_2 & x_2 \cdot y_3 \\ \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_3 \odot y_1) & x_3 \cdot y_2 & x_3 \cdot y_3 \end{pmatrix}\end{aligned}$$

La matrice B peut se ré-écrire par la relation suivante :

$$\mathrm{B} = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & b_{2,2} & b_{2,3} \\ 0 & 0 & b_{3,3} \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ b_{2,1} & 0 & 0 \\ b_{3,1} & b_{3,2} & 0 \end{pmatrix}$$

soit :

$$\begin{aligned}\mathrm{B} &= \begin{pmatrix} \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{A}) \times (x_1 \odot y_1) & \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_2) & \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_3) \\ 0 & x_2 \cdot y_2 & x_2 \cdot y_3 \\ 0 & 0 & x_3 \cdot y_3 \end{pmatrix} \\ &\oplus \begin{pmatrix} 0 & 0 & 0 \\ \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_2 \odot y_1) & 0 & 0 \\ \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_3 \odot y_1) & x_3 \cdot y_2 & 0 \end{pmatrix}.\end{aligned}$$

Comme pour le schéma ISW, Wang *et. al.* introduisent 3 vecteurs $r_{1,2}, r_{1,3}$ et $r_{2,3}$ uniformément distribués dans $\mathrm{GF}(2)^8$ et indépendants de x et de y. Les auteurs construisent les éléments (z_1, z_2, z_3) de $\mathbf{z}$ tels que z_i $(i = \{1,2,3\})$ correspond à l'addition des termes de la $i^{ème}$ ligne[8] de la matrice B′, définie à partir de B et l'aléa par :

$$\mathrm{B}' = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & b_{2,2} & b_{2,3} \\ 0 & 0 & b_{3,3} \end{pmatrix} \oplus \begin{pmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & b_{2,1} & b_{3,1} \\ 0 & 0 & b_{3,2} \\ 0 & 0 & 0 \end{pmatrix}$$

8. Comme présenté en en Section 3.2.4, le schéma ISW définit les éléments de $\mathbf{z}$ comme l'addition des termes de la $i^{ème}$ colonne, mais il est possible de construire ces éléments en considérant la $i^{ème}$ ligne, sans altérer l'ordre de sécurité ciblé.

Pu *et. al.* obtiennent ainsi le partage additif en 3-morceaux $\mathbf{z} = (z_1, z_2, z_3)$ de $x \cdot y$ tel que :

$$
\begin{aligned}
z_1 &= \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{A}) \times (x_1 \odot y_1) \oplus \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_2) \oplus r_{1,2} \oplus \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_2 \odot y_1) \\
&\quad \oplus \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{A} \odot \mathrm{E}) \times (x_1 \odot y_3) \oplus r_{1,3} \oplus \mathrm{A}^{-1} \times \mathrm{J} \times (\mathrm{E} \odot \mathrm{A}) \times (x_3 \odot y_1) \\
z_2 &= \mathrm{A} \times r_{1,2} \oplus x_2 \cdot y_2 \oplus x_2 \cdot y_3 \oplus r_{2,3} \oplus x_3 \cdot y_2 \\
z_3 &= \mathrm{A} \times r_{1,3} \oplus r_{2,3} \oplus x_3 \cdot y_3.
\end{aligned}
$$

La propriété de correction est vérifiée par $rec_{add}(\mathbf{z}) = \mathrm{A} \times z_1 \oplus z_2 \oplus z_3$. En effet, en multipliant z_1 par A nous obtenons l'addition des produits $(\mathrm{A} \times x_1) \cdot (\mathrm{A} \times y_1), (\mathrm{A} \times x_1) \cdot y_2, (\mathrm{A} \times x_1) \cdot y_3, x_2 \cdot (\mathrm{A} \times y_1)$ et $x_3 \cdot (\mathrm{A} \times y_1)$ décomposés sous la forme de produits de Kronecker et masqués par les vecteurs $\mathrm{A} \times r_{1,2}$ et $\mathrm{A} \times r_{1,3}$. Puis, additionner les termes de $\mathrm{A} \times z_1, z_2$ et de z_3 implique de supprimer les masques $r_{1,2}, r_{1,3}$ et $r_{2,3}$ et retourne bien la multiplication de x par y telle que décrite en Equation 5.9 (pour $d = 3$). De plus, comme chaque vecteur z_i ($i = \{1,2,3\}$) est masqué par un vecteur de la forme $r_{i,j}$ il est impossible pour un attaquant d'avoir de l'information sur la valeur de x ou de y. La réalisation $\mathbf{z}$ est donc sécurisée vis-à-vis du sondage à l'ordre 1.

Pour conclure, basé sur le travail de Wang *et. al.* , Pu *et. al.* [70] montrent comment sécuriser le calcul des 4 multiplications de la fonction $\mathrm{INV}_{\mathrm{SM4}}$. Les 3 exponentiations linéaires de $\mathrm{INV}_{\mathrm{SM4}}$ ainsi que la fonction $\mathrm{AF}_{\mathrm{SM4}}$ sont accessibles via une table additivement masquée par un masque fixe définis sur $\mathrm{GF}(2)^8$. La fonction non-linéaire τ du SM4 est alors remplacée par une nouvelle transformation non-linéaire, constituée de 4 boîtes-S dont les multiplications sont protégées par le masquage matriciel additif de Pu *et. al.* . En remplaçant τ_{M} par cette nouvelle transformation, la contre-mesure de Pu *et. al.* de tout l'algorithme SM4 est similaire à celle de Duan *et. al.* , représentée dans les Figures 46 à 50.

Dans la suite, nous décrivons une technique de dissimulation qui peut être combinée aux différents schémas de masquage présentés ci-dessus.

5.3.2 Dissimulation du SM4

En 2013, Wang *et. al.* [86] publient une attaque par observations à l'ordre 1 contre le SM4 en utilisant la technique de l'attaque *à clairs choisis*. Cette approche permet non seulement à l'adversaire de connaître les messages chiffrés et les messages clairs correspondants, mais il peut en plus choisir des messages clairs spécifiques qui donneront de l'information sur la clé secrète. Par exemple, il peut choisir de chiffrer un message clair dont tous les octets sont nuls sauf le premier octet du quatrième mot de 32 bits. En exploitant la sortie de la première fonction de tour il est possible de trouver la valeur de la première clé de tour. En continuant ainsi sur les tours suivants, l'attaque proposée par Wang *et. al.* permet de retrouver la clé cryptographique originale.

Chen *et. al.* [20] proposent une méthode de dissimulation afin de sécuriser le chiffrement d'une donnée sensible à travers les quatre premiers tours (*i.e.* contre les attaques par observations à clairs choisis, par exemple). Les auteurs augmentent le bruit dans le circuit implémentant le SM4 en utilisant la technique d'ajout d'opérations factices présentée en Section 2.4.1. Nous présentons leur schéma de dissimulation ci-dessous.

Avant chaque chiffrement d'un message clair, quatre entiers positifs $\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2$ et R_3 sont générés aléatoirement tels que $\mathrm{R}_0 + \mathrm{R}_1 + \mathrm{R}_2 + \mathrm{R}_3 = 32$. Comme illustré dans la Figure 52 chacun de ces entiers déterminent le nombre de tours factices (aussi appelés *faux tours*) à insérer entre les quatre premiers vrais tours du SM4. Afin de rendre difficile la distinction entre un faux tour d'un vrai, la structure d'un faux tour est similaire à celle d'un vrai, à l'exception de l'entrée du tour et de la clé, générées aléatoirement dans $\mathrm{GF}(2)^{128}$. Les sorties des faux tours sont ensuite stockées dans des registres.

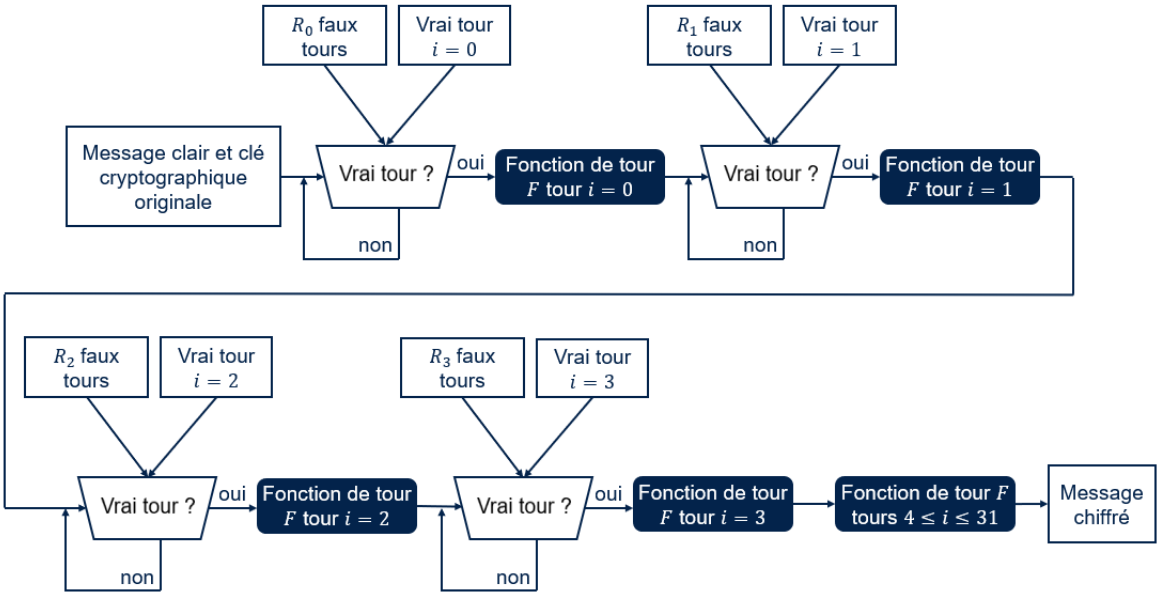


FIGURE 52 – Proposition de Chen *et. al.* [20] - Schéma de dissimulation de l'algorithme de chiffrement du SM4

Remarque 19. *Pour contrer les attaques par observations à l'ordre* 1 *visant à retrouver la valeur des dernières clés de tours du SM4 afin de remonter à la clé originale [5], le schéma de dissimulation décrit en Figure 52 peut aussi s'appliquer aux quatre derniers tours de l'implémentation du SM4.*

Dans la suite, nous comparons les performances des différentes contre-mesures [30, 20, 70] présentées précédemment.

5.4 Comparaison des performances des propositions de l'état de l'art

Dans cette section, nous avons implémenté les contre-mesures de l'état de l'art [30, 20, 70] présentées dans la section précédente sur un microcontrôleur afin de comparer leur performances. Pour une sécurité à l'ordre 1 dans le modèle d'attaque par sondage, nous avons mesuré pour chaque implémentation le temps d'exécution et la taille du code implémenté.

5.4.1 Choix d'implémentation

Nos résultats expérimentaux sont évalués sur le microcontrôleur de STMicroelectronics appelé *STM32F411RET6 Nucleo-64*. Ce microcontrôleur est basé sur un processeur ARM Cortex M4 cadencé à 100MHz disposant de 512KB de mémoire flash et de 128KB de mémoire vive (SRAM). La carte de développement Nucleo-F411RE permet à l'utilisateur de construire ses propres prototypes pour les applications embarquées.

Dans notre cas, nous avons implémenté en langage assembleur sur STM32 les contre-mesures du SM4 de l'état de l'art [30, 20, 70] telles que nous les avons décrite dans la Section 5.3. Nous avons privilégié ce langage de programmation car son fonctionnement est très proche du *langage machine*, c'est à dire du langage binaire qu'utilise un ordinateur. Pour faire exécuter une suite d'instructions au processeur du STM32, il faut lui fournir des données binaires, souvent représentées en notations hexadécimales. Ces instructions hexadécimales étant difficiles à comprendre pour un humain, le langage assembleur a été mis au point afin de les noter avec des noms explicites suivis de paramètres. Par exemple, l'instruction "A1 01 10" permet de copier le contenu de la mémoire à l'adresse $0110h$ dans le registre AX [9] du processeur. Comme cette instruction ne dit rien à

9. Ce registre sert à effectuer des calculs arithmétiques.

un humain, le langage assembleur utilise une notation symbolique sous forme textuelle, appelée *code source*, afin d'obtenir une instruction plus facile à comprendre. L'instruction "A1 01 10" est donc remplacée par "MOV AX, [0110]" en langage assembleur. Le code source est ensuite compilé par l'assembleur et traduit en langage machine.

De plus, le langage assembleur présente l'avantage de toujours savoir quel code est exécuté par le microprocesseur à un instant donné. Cela nous a d'une part permis de mesurer correctement le temps d'exécution de chaque contre-mesure, à l'opération près, et d'autre part d'être sûr que l'implémentation s'exécute comme on le souhaite.

Nous avons implémenté deux versions du SM4 non-protégé. La première, appelée *SM4_LUT* implémente la fonction non-linéaire de la boîte-S sous la forme de sa table de correspondance donnée dans la Figure 44. La deuxième, appelée *SM4_Algébrique*, traite la boîte-S sous sa décomposition algébrique donnée dans l'Equation 5.2. Après avoir comparé les performances de [30, 20] entre elles, nous les avons confronté à celles de *SM4_LUT*. Puis, nous avons comparé les performances de [70] avec celles de *SM4_Algébrique*. En supplément, nous avons également implémenté un SM4 sécurisé en utilisant le masquage additif du schéma ISW [42] sur les multiplications de la fonction INV_{SM4} de la boîte-S. Cette implémentation a ensuite été comparé à la contre-mesure de Pu *et. al.* [70] et à l'implémentation *SM4_Algébrique*.

Dans la suite, nous décrivons nos résultats.

5.4.2 Performances

Nos résultats expérimentaux sont affichés dans le Tableau 13. Notre implémentation en assembleur de *SM4_LUT* occupe 24KB d'espace mémoire et s'exécute en 0.4203ms. Après application du masquage additif sur les opérations du SM4, la contre-mesure de Duan *et. al.* (voir Algorithme 5) chiffre un message clair en 0.8373ms, soit quasiment deux fois moins vite. La génération d'un masque additif aléatoire et les conséquences sur les opérations du SM4 augmentent également la taille du code à 35KB. La contre-mesure de Chen *et. al.* présentée en Figure 51 diffère de celle de Duan *et. al.* par la génération de différents masques additifs au cours du chiffrement. Comme la manière de construire des nombres aléatoires est propre à chaque développeur, nous n'avons pas pris en compte les performances liées à la génération d'aléa dans nos résultats. Ainsi, pour une taille de code de 39KB, et une vitesse d'exécution de 0.8782ms, nous concluons que la technique de Chen *et. al.* est une bonne alternative à celle de Duan *et. al.* car elle présente des résultats similaires. Les deux contre-mesures décrites dans les articles [30] et [20] sont sécurisées vis-à-vis du sondage à l'ordre 1.

Notre implémentation en assembleur non-sécurisée *SM4_Algébrique* implémente la boîte-S du SM4 avec sa décomposition algébrique. Sa taille de code est de 350KB et son temps d'exécution est de 3.1616ms, soit 7,5 fois moins vite que l'exécution de *SM4_LUT*. Cependant, il peut être intéressant de travailler sur une description algébrique de la boîte-S du SM4 lorsque l'on souhaite réaliser une implémentation matérielle sécurisée du SM4. Plus encore, Pu *et. al.* [70] proposent une contre-mesure générique sécurisée vis-à-vis du sondage à n'importe quel ordre (*i.e.* égal ou supérieur à 1). Comparé à *SM4_Algébrique*, pour une taille de code de 443KB, la contre-mesure de [70] protège le chiffrement d'un message clair en 8.8107ms contre les attaques par observations à l'ordre 1, soit un facteur de 2,79 fois plus lent que la version non-sécurisée du SM4. Afin de réduire ce facteur, nous avons implémenté un masquage additif du SM4 en utilisant le schéma ISW [42] à l'ordre 1 sur les multiplications non-linéaires de la boîte-S (voir Remarque 17). Nous obtenons un chiffrement sécurisé en 5.4571ms, soit 1,73 fois plus lent qu'un chiffrement non-sécurisé avec *SM4_Algébrique* et 1,62 fois plus rapide que le schéma de masquage de Pu *et. al.* . De plus, le masquage de données sensibles est assuré au moyen d'octets aléatoires dans le schéma ISW au lieu de matrices dans [70], ce qui réduit également la taille de code à 402KB.

Conception	Vitesse d'exécution (ms)	Mémoire (KB)	Implémentation de la boîte-S	Ordre de sécurité vis-à-vis du sondage
SM4_LUT [29]	0.4203 (x1.00)	24 (x1.00)	LUT	non-sécurisé
Duan *et. al.* [30]	0.8373 (x1.99)	35 (x1.46)	LUT	ordre 1
Chen *et. al.* [20]	0.8782 (x2.09)	39 (x1.63)	LUT	ordre 1
SM4_Algébrique (voir Équation 5.2)	3.1616 (x1.00)	350 (x1.00)	Décomposition algébrique	non-sécurisé
Pu *et. al.* [70]	8.8107 (x2.79)	443 (x1.27)	Décomposition algébrique	ordre 1 *Rq. : ordre* ≥ 1 *possible*
Schéma ISW [42]	5.4571 (x1.73)	402 (x1.15)	Décomposition algébrique	ordre 1 *Rq. : ordre* ≥ 1 *possible*

TABLEAU 13 – Notre comparaison des contre-mesures de l'état de l'art pour le SM4, sécurisées vis-à-vis du sondage à l'ordre 1

Nous pouvons remarquer que les contre-mesures étudiées dans notre comparaison (voir Tableau 13) ne proposent pas de sécurité dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches. En s'inspirant de la proposition de Bilgin *et. al.* [10] sur l'AES, seulement Li *et. al.* [51] proposent une implémentation à seuil de la boîte-S du SM4. Nos travaux sur cet article ont conclu à de nombreuses erreurs présentes dans leur construction. Par exemple, leur réalisation ne satisfait pas la propriété de correction. Par ailleurs, les auteurs de [51] ne donnent aucun détail sur la manière de protéger les autres opérations du SM4 dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches. Dans la suite, nous proposons une implémentation à seuil du SM4 afin de sécuriser le chiffrement d'une donnée sensible face aux attaques par observations à l'ordre 1 en présence de glitches.

5.5 Notre proposition du SM4 sécurisée à l'ordre 1 en présence de glitches

5.5.1 Structure de notre construction

Nous nous référons à la Section 5.2 pour une description complète du SM4. Notre stratégie de masquage est similaire à celle de l'AES présentée en Section 2.3.6. La donnée sensible est découpée en un partage additif en 3-morceaux lorsqu'il s'agit de masquer une opération linéaire. Pour le calcul sécurisé d'une boîte-S, nous utilisons un partage affine (voir Définition 22) de l'octet manipulé afin de masquer multiplicativement la partie non-linéaire du SM4.

Nous représentons dans la Figure 53 la structure de notre implémentation à seuil l'algorithme de chiffrement d'un tour de SM4. Les variables **x** et **rk** correspondent aux partages additifs du bloc interne et de la clé de tour, respectivement, en entrée du tour. L'élément $\mathbf{x}_i$ (resp. $\mathbf{x}'_j$) pour $i \in [1..6]$ (resp. $j \in [1..5]$) représente un partage intermédiaire d'un mot de 32 bits (resp. d'un octet) entre les opérations du SM4. Les fonctions $\mathrm{AF}_{\mathrm{SM4}}$ et $\mathrm{INV}_{\mathrm{SM4}}$ impliquées dans le calcul d'une boîte-S de la fonction non-linéaire τ sont expliquées dans la Section 5.2.2. L'implémentation à seuil de la fonction $\mathrm{INV}_{\mathrm{SM4}}$ est une application de notre évaluation monomiale décrite dans le Chapitre 4 en Section 4.3.

Dans la suite nous décrivons la Figure 53 telle que les opérations soient sécurisées à l'ordre 1 en présence de glitches.

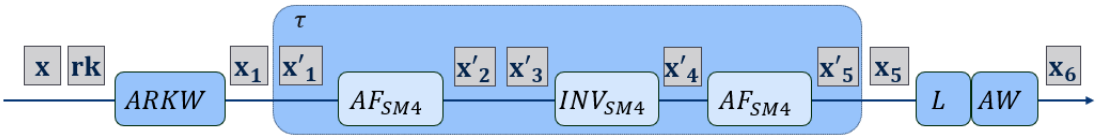


FIGURE 53 – Structure de notre implémentation à seuil de l'algorithme de chiffrement d'un tour du SM4

5.5.2 Implémentation d'un chiffrement par bloc du SM4

Dans notre proposition, comme pour l'AES, tous les partages en 3-morceaux manipulent un même masque multiplicatif au cours d'un tour. Ce n'est pas le cas du masque additif que nous faisons évoluer à chaque masquage d'une opération du SM4. Sans perte de généralités, nous détaillons ci-dessous les implémentations à seuil des opérations de l'algorithme de chiffrement du SM4 pour le premier tour. Le message d'entrée est représenté par le bloc interne $(x_0 \| x_1 \| x_2 \| x_3) \in \mathrm{GF}(2)^{128}$ et la clé de tour correspondante est notée rk_0. Notre construction est la même pour tout bloc interne $(x_i \| x_{i+1} \| x_{i+2} \| x_{i+3}) \in \mathrm{GF}(2)^{128}$ et sa clé de tour rk_i ($i \in [0..31]$) en entrée (voir Figure 43).

Initialisation. On note $\beta_0, \beta_1, \beta_2, \beta_3$ et β_{rk_0} 5 masques additifs de x_0, x_1, x_2, x_3 et rk_0, respectivement, uniformément distribués dans $\mathrm{GF}(2)^{32}$. Soit également $\alpha \in \mathrm{GF}(2)^{32\star}$ un masque multiplicatif. On a :

$$\mathbf{x} = ((x_0 \oplus \beta_0, \alpha, \beta_0), (x_1 \oplus \beta_1, \alpha, \beta_1), (x_2 \oplus \beta_2, \alpha, \beta_2), (x_3 \oplus \beta_3, \alpha, \beta_3)).$$

La fonction de reconstruction de sortie des 4 partages additifs de $\mathbf{x}$ correspond à l'addition entre leur premier et leur troisième morceau. De la même manière, le partage additif de la clé de tour est défini par :

$$\mathbf{rk} = (rk_0 \oplus \beta_{rk_0}, \alpha, \beta_{rk_0}).$$

Étape 1 : Premiers Xors (fonction ARKW**).** Le calcul de ARKW peut être sécurisé en ajoutant les premiers (respectivement les troisièmes) morceaux des 3 derniers partages de $\mathbf{x}$ et celui de $\mathbf{rk}$ entre eux. Nous illustrons en Figure 54 notre construction en 1 cycle du partage additif en 3-morceaux $\mathbf{x}_1 = (x_1 \oplus x_2 \oplus x_3 \oplus rk_0 \oplus \beta, \alpha, \beta)$, où $\beta = \beta_1 \oplus \beta_2 \oplus \beta_3 \oplus \beta_{rk_0}$. On définit le résultat de l'opération ARKW par le mot $\mathrm{A} \in \mathrm{GF}(2)^{32}$ (voir Equation 5.1). Pour plus de lisibilité dans la suite, le partage additif de A est représenté par $\mathbf{x}_1 = (\mathrm{A} \oplus \beta_\mathrm{A}, \alpha, \beta_\mathrm{A})$.

Proposition 10. *La réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{ARKW}} : (\mathbf{x}, \mathbf{rk}) \mapsto \mathbf{x_1}$ *de la fonction* $\mathrm{F} : (x, rk) \mapsto x \oplus rk$ *est correcte, glitches-immune à l'ordre* 1 *et uniforme. Elle est sécurisée à l'ordre* 1 *en présence de glitches.*

Preuve. *La preuve de la Proposition 10 est similaire à la Preuve de la Proposition 4*

□

Étape 2 : Transformation non-linéaire. La fonction τ consiste à l'application de la boîte-S du SM4 sur chacun des octets du mot A obtenu à l'étape précédente (*i.e.* après l'opération ARKW). Cette boîte-S, notée SBox, est définie en Équation 5.2 par la composition d'une transformation affine $\mathrm{AF}_{\mathrm{SM4}}$ suivi d'une fonction inverse $\mathrm{INV}_{\mathrm{SM4}}(x) = x^{254} \in \mathrm{GF}(2^8)$ et de $\mathrm{AF}_{\mathrm{SM4}}$ une deuxième fois. Notre construction se divise en 2 étapes :

— Soient a un octet de A et $\beta_a \in \mathrm{GF}(2)^8$ le même octet de β_A, soit le masque additif associé à a. Une première phase est dédiée à l'application de la transformation $\mathrm{AF}_{\mathrm{SM4}}$ sur a. On considère donc le partage additif $\mathbf{x}'_1 = (a \oplus \beta_a, \alpha, \beta_a)$ de a. Afin de préparer le masquage multiplicatif de la fonction non-linéaire suivante (*i.e.* $\mathrm{INV}_{\mathrm{SM4}}$), nous transformons en 1 cycle le partage additif $\mathbf{x}'_1$ en un partage affine $\mathbf{x}'_2$ de $\mathrm{AF}_{\mathrm{SM4}}(a)$. Cette conversion est illustrée en

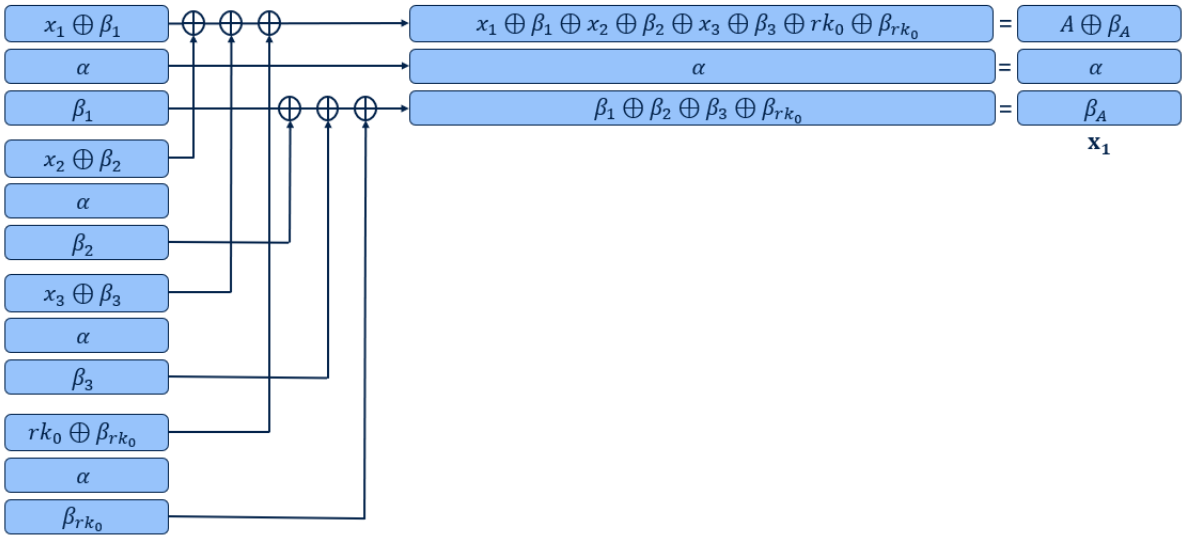


FIGURE 54 – Calcul sécurisé de ARKW en 1 cycle

Figure 55. Nous obtenons alors $\mathbf{x}'_2 = ((\mathrm{AF}_{\mathrm{SM4}}(a) \oplus \beta_{a'}) \otimes \alpha, \alpha, \beta_{a'})$, où $\beta_{a'} = \mathscr{M}_{\mathrm{SM4}}(\beta_a)$ (voir le détail de $\mathscr{M}_{\mathrm{SM4}}$ en Section 5.2.2). Pour plus de lisibilité dans la suite, nous notons $\mathbf{x}'_2 = ((a' \oplus \beta_{a'}) \otimes \alpha, \alpha, \beta_{a'})$, où $a' = \mathrm{AF}_{\mathrm{SM4}}(a)$. La fonction de reconstruction de sortie de $\mathbf{x}'_2$ est donnée dans la Définition 22.

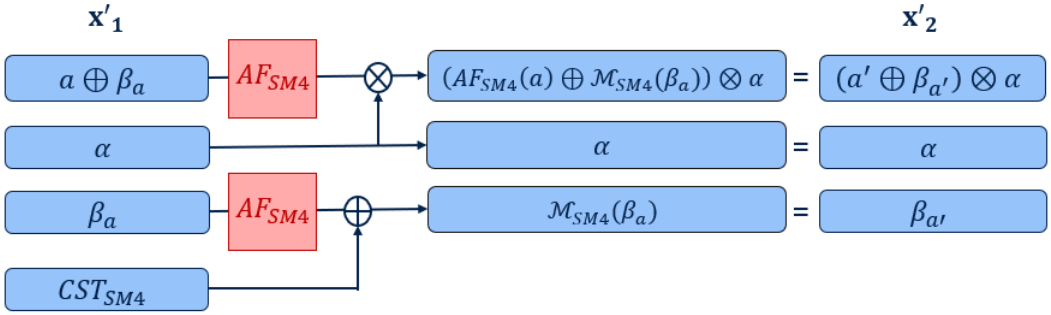


FIGURE 55 – Conversion du partage additif $\mathbf{x}'_1$ de a en un partage affine $\mathbf{x}'_2$ de $\mathrm{AF}_{\mathrm{SM4}}(a)$ pour un octet a de A - $1^{ère}$ partie de l'illustration de notre implémentation à seuil de la boîte-S du SM4 en 1 cycle

— Une deuxième phase consiste alors à appliquer notre implémentation à seuil de la fonction $\mathrm{F}_{pow}(x) = x^q$ présentée en Section 4.3 sur la fonction $\mathrm{INV}_{\mathrm{SM4}}(x) = x^{254}$. En effet, comme la puissance de x^{254} est première avec $2^8 - 1$ (i.e. l'ordre du sous-groupe multiplicatif de $\mathrm{GF}(2^8)$), nous pouvons utiliser la Proposition 3. Pour plus de lisibilité, nous notons $q = -1$ dans la suite. Ainsi, de la même manière que pour la boîte-S de l'AES, notre réalisation 3-masquée de la fonction $\mathrm{INV}_{\mathrm{SM4}}(a')$ est construite par l'application de la Figure 36 sur la fonction $\mathrm{INV}_{\mathrm{SM4}}$ dans laquelle nous intégrons le calcul sécurisé de la fonction $\mathrm{AF}_{\mathrm{SM4}}$. Un partage additif en 2-morceaux (δ_1, δ_2) de la fonction de Dirac de a' est généré comme présenté en Section 4.2.2. Notre réalisation de $\mathrm{INV}_{\mathrm{SM4}}(a')$ prend alors en entrée un partage en 5-morceaux $\mathbf{x}'_3 = ((a' \oplus \beta_{a'}) \otimes \alpha, \alpha, \beta_{a'}, \delta_1, \delta_2)$ constitué du partage affine $\mathbf{x}'_2$ de a' et du partage additif de $\delta(a')$. Notre construction est illustrée en Figure 56 et détaillée ci-dessous.

Dans un premier temps, en adaptant la description de la partie supérieure de la Figure 36, nous obtenons le partage additif en 3-morceaux $(\mathrm{INV}_{\mathrm{SM4}}(a') \oplus \beta_{a'} \oplus \delta(a'), \alpha^{-1}, \beta_{a'} \oplus \delta(a'))$ de $\mathrm{INV}_{\mathrm{SM4}}(a')$. La fonction de reconstruction associée à ce partage consiste en la somme entre le premier et le troisième morceau. Nous parlons d'adaptation de notre proposition car le masque multiplicatif α^{-1} a été retiré du premier morceau. En effet, un masquage additif de la valeur sensible est suf-

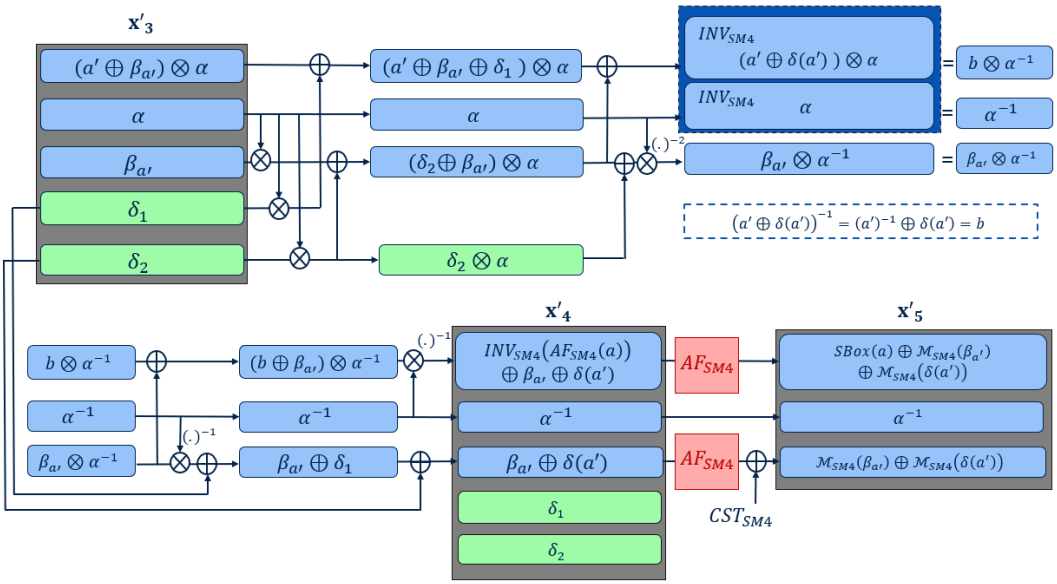


FIGURE 56 – Adaptation de notre évaluation monomiale sur la fonction $\mathrm{INV_{SM4}}(x) = x^{254}$ suivie du calcul sécurisé de la fonction $\mathrm{AF_{SM4}}$ - $2^{ème}$ partie de l'illustration de notre implémentation à seuil de la boîte-S du SM4 en 5 cycles

fisant pour garantir la glitches-immunité et protéger la deuxième fonction affine $\mathrm{AF_{SM4}}$ qui suit. Nous évitons ainsi une conversion de notre partage affine en un partage additif plus tard. Nous obtenons le partage additif en 5-morceaux $\mathbf{x}'_4 = (\mathrm{INV_{SM4}}(a') \oplus \beta_{a'} \oplus \delta(a'), \alpha^{-1}, \beta_{a'} \oplus \delta(a'), \delta_1, \delta_2)$ de $\mathrm{INV_{SM4}}(a')$. Puis, le calcul sécurisé de $\mathrm{AF_{SM4}}$ nous conduit au partage additif en 3-morceaux $\mathbf{x}'_5 = (\mathrm{SB}ox(a) \oplus \mathcal{M}_{\mathrm{SM4}}(\beta_{a'}) \oplus \mathcal{M}_{\mathrm{SM4}}(\delta(a')), \alpha^{-1}, \mathcal{M}_{\mathrm{SM4}}(\beta_{a'}) \oplus \mathcal{M}_{\mathrm{SM4}}(\delta(a')))$ de $\mathrm{SB}ox(a)$. Ce calcul peut être fusionné avec le $4^{ème}$ cycle de la construction de $\mathbf{x}'_4$ sans introduire de vulnérabilité dans l'implémentation car $\mathrm{AF_{SM4}}$ est une bijection appliquée séparément sur le premier et troisième morceau de $\mathbf{x}'_4$. La fonction de reconstruction de sortie associée à $\mathbf{x}'_4$ (respectivement $\mathbf{x}'_5$) consiste en la somme entre le premier et le troisième morceau de $\mathbf{x}'_4$ (respectivement $\mathbf{x}'_5$). Au final, la réalisation 3-masquée de la boîte-S du SM4 s'effectue en 6 cycles (1 pour le calcul sécurisé du premier appel de $\mathrm{AF_{SM4}}$, 4 pour le calcul sécurisé de $\mathrm{INV_{SM4}}(\mathrm{AF_{SM4}}(a))$ et 1 pour la fonction de Dirac sous forme de table de correspondance).

Proposition 11. *La réalisation* 3*-masquée* $\mathbf{F}_\tau : \mathbf{x}'_1 \mapsto \mathbf{x}'_5$ *de la fonction* $\mathrm{F} : a \mapsto \mathrm{SB}ox(a)$ *est correcte, glitches-immune à l'ordre* 1 *et uniforme. Elle est sécurisée à l'ordre* 1 *en présence de glitches.*

Preuve. *Soit la réalisation* 3*-masquée* $\mathbf{F}_{\mathrm{AF}} : \mathbf{x}'_1 \mapsto \mathbf{x}'_2$ *de la fonction* $\mathrm{F} : x \mapsto \mathrm{AF_{SM4}}(x)$ *(voir Figure 55). La fonction de reconstruction de sortie de* $\mathbf{x}_2$ *est définie dans la Définition 22. Par ailleurs, la fonction* $\mathrm{AF_{SM4}}$ *est appliquée au premier et au troisième morceau de* $\mathbf{x}'_1$*, séparément. Chaque image de* $\mathrm{AF_{SM4}}$ *a donc un seul antécédent, ce qui garantit la glitches-immunité à l'ordre* 1 *de la réalisation* $\mathbf{F}_{\mathrm{AF}}$*. Enfin, comme* $\mathrm{AF_{SM4}}$ *est une bijection, la réalisation* $\mathbf{F}_{\mathrm{AF}}$ *est uniforme.*

Soit la réalisation 3*-masquée* $\mathbf{F}_{\mathrm{INV-AF}} : \mathbf{x}'_3 \mapsto \mathbf{x}'_5$ *de la fonction* $\mathrm{F} : x \mapsto \mathrm{INV_{SM4}}(\mathrm{AF_{SM4}}(x))$ *(voir Figure 56). D'après la Proposition 5,* $\mathbf{F}_{\mathrm{INV-AF}}$ *est correcte, glitches-immune à l'ordre* 1 *et uniforme.*

Comme la réalisation 3*-masquée* $\mathbf{F}_\tau$ *est définie par* $\mathbf{F}_\tau = \mathbf{F}_{\mathrm{AF}} \circ \mathbf{F}_{\mathrm{INV-AF}}$*, d'après la Propriété 3,* $\mathbf{F}_\tau$ *est une implémentation à seuil de la boîte-S du SM4, sécurisée à l'ordre* 1 *en présence de glitches.*

□

Après application de la boîte-S sur les octets de A, on définit le résultat de l'opération τ(A) par le mot $\mathrm{B} \in \mathrm{GF}(2)^{32}$. Pour plus de lisibilité dans la suite, le partage additif de B est représenté par

$\mathbf{x}_5 = (B \oplus \beta_B, \alpha, \beta_B)$, où β_B est égal à la concaténation des masques additifs de chaque boîte-S utilisée par la fonction τ.

Étape 3 : Transformation linéaire. Le mot B est ensuite modifié par l'opération linéaire L (voir Equation 5.3). La transformation L implique simplement un ré-indexage du partage additif $\mathbf{x}_5$ de B. Elle est appliquée sur les morceaux de $\mathbf{x}_5$ sans nécessiter un cycle supplémentaire. Cette étape peut donc être fusionnée avec l'étape suivante sans introduire de vulnérabilité.

Étape 4 : Xor final (AW)**.** L'opération AW fonctionne de la même manière que ARKW. Elle peut être sécurisée en ajoutant les premiers (respectivement les troisièmes) morceaux du partage additif $\mathbf{x}_5$ (après application de L sur chacun des morceaux) et du premier partage additif de $\mathbf{x}$ (*i.e.* le partage du mot x_0). Nous obtenons en 1 cycle le partage additif $\mathbf{x}_6 = (L(B) \oplus x_0 \oplus L(\beta_B) \oplus \beta_0, \alpha, L(\beta_B) \oplus \beta_0)$, soit le partage additif du nouveau mot x_4 produit à l'issu du premier tour du chiffrement SM4. L'entrée du tour suivant est donc défini par les partages additifs des mots x_1, x_2, x_3, x_4 ainsi que le partage additif de la clé de tour rk_1.

Proposition 12. *La réalisation* 3*-masquée* $\mathbf{F}_{lin} : (\mathbf{x}, \mathbf{x_5}) \mapsto \mathbf{x_6}$ *de la fonction* $F : (x, y) \mapsto x \oplus L(y)$ *est correcte, glitches-immune à l'ordre* 1 *et uniforme. Elle est sécurisée à l'ordre* 1 *en présence de glitches.*

Preuve. *La propriété de correction et d'uniformité de la réalisation* 3*-masquée* $\mathbf{F}_{lin}$ *est une conséquence directe de la linéarité dans* $GF(2)^{32}$ *des transformations* L *et* AW*. De plus, la construction du premier (respectivement du troisième) morceau de* $\mathbf{x_6}$ *en sortie est indépendant fonctionnellement du troisième (respectivement du premier) morceau de* $\mathbf{x_5}$ *en entrée. Le deuxième morceau de* $\mathbf{x_6}$ *est une copie directe du deuxième morceau de* $\mathbf{x_5}$*. La réalisation* 3*-masquée* $\mathbf{F}_{lin}$ *est donc glitches-immune à l'ordre* 1*. Pour conclure, la réalisation* 3*-masquée* $\mathbf{F}_{lin}$ *est une implémentation à seuil de la fonction* $F : (x, y) \mapsto x \oplus L(y)$*, sécurisée à l'ordre* 1 *en présence de glitches.*

□

Remarque 20. *Les partages additifs* $\mathbf{x}$ *et* $\mathbf{rk}$ *en entrée de tour doivent contenir le même masque multiplicatif. Ainsi un tour* $i = 0 \ mod(2)$ *du SM4 doit manipuler le masque multiplicatif* α *alors qu'un tour* $i = 1 \ mod(2)$ *doit utiliser la valeur de* α^{-1} *pour masque la partie non-linéaire de l'algorithme (pour* $i \in [0..31]$*).*

Pour conclure cette partie, d'après les Propositions 10, 11 et 12, notre proposition d'implémentation à seuil de l'algorithme de chiffrement du SM4 est sécurisée à l'ordre 1 en présence de glitches. Notre construction se fait en 8 cycles comme détaillé dans le Tableau 14.

Fonctions	Nombre de cycles
ARKW	1
τ	4×6 (voir Remarque 21)
L et AW	1

TABLEAU 14 – Nombre de cycle pour chaque opération d'un tour de chiffrement SM4

Remarque 21. *Les* 6 *cycles de l'opération* $\tau(A)$ *s'exécutent pour chacun des* 4 *octets de* A *en parallèle.*

Notre implémentation à seuil de la génération de chaque clé de tour est détaillée dans la suite.

5.5.3 Implémentation de l'algorithme de cadencement de clé du SM4

Nous nous référons à la Section 5.2.3 pour une description complète de l'algorithme de cadencement de clé du SM4. Nous rappelons que cet algorithme prend en entrée la clé cryptographique originale $k \in GF(2)^{128}$, et deux paramètres publics $FK \in GF(2)^{128}$ et $CK_i \in GF(2)^{32}$, et retourne à chaque tour $i \in [0..31]$ un nouveau mot de 32 bits correspondant à la clé de tour rk_i.

Nous détaillons notre proposition d'implémentation à seuil de la génération de ces clés de de tour ci-dessous.

Initialisation. Soit $l \in [0..3]$, on note k_l un mot de 32 bits de k. Soient β_{k_l} le masque additif uniformément distribué dans $GF(2)^{32}$ de k_l et α un masque multiplicatif définis dans $GF(2)^{32\star}$. On définit par $(k_l \oplus \beta_{k_l}, \alpha, \beta_{k_l})$ le partage additif de k_l tel que la fonction de reconstruction de sortie correspond à l'addition entre le premier et le troisième morceau. Comme les paramètres FK et CK_i sont publics, il n'est pas nécessaire de les protéger.

Avant de rentrer dans la fonction de tour de l'algorithme de cadencement de clé, nous avons vu en Equation 5.4 qu'il est nécessaire d'additionner k et FK entre eux. Nous illustrons en Figure 57 notre calcul sécurisé de l'addition entre les mots de k et FK. En 1 cycle, le mot FK_l est ajouté au premier morceau du partage additif $(k_l \oplus \beta_{k_l}, \alpha, \beta_{k_l})$ de k_l. Il en résulte le mot $k'_l \in GF(2)^{32}$ dont le partage additif est défini par $(k'_l \oplus \beta_{k'_l}, \alpha, \beta_{k'_l})$. La fonction de reconstruction de sortie est définie par l'addition entre le premier et le troisième morceau de ce partage.

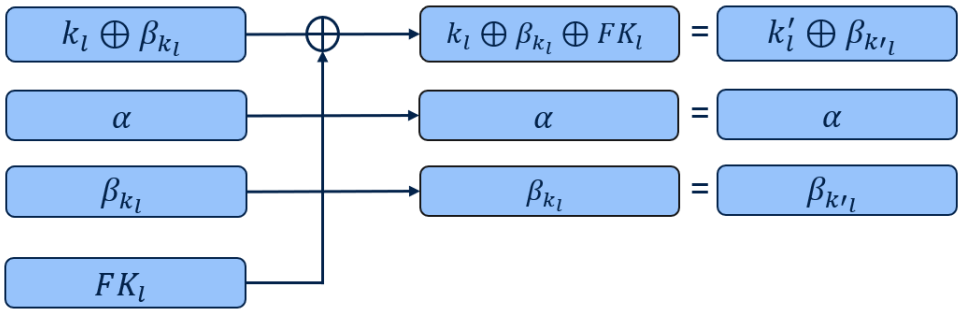


FIGURE 57 – Calcul sécurisé de $k_l \oplus FK_l$ pour $l \in [0..3]$

Puis, à partir de $(k'_0 \| k'_1 \| k'_2 \| k'_3) \in GF(2)^{128}$, pour $i \in [0..31]$, il est possible de construire les 32 clés de tour $rk_i = k'_{i+4}$ par la fonction de tour décrite en Equation 5.6. Comme cette dernière suit les mêmes étapes que celle de l'algorithme de chiffrement pour une transformation linéaire différente (voir Equation 5.5), son implémentation à seuil est similaire à celle que nous avons décrite dans les étapes [10] 1 à 4 de la section précédente.

Pour conclure, pour les mêmes raisons invoquées en Section 5.5.2 (*i.e.* les Propositions 10, 11 et 12 et la conservation de la sécurité en présence de glitches à travers la transformation linéaire de la Figure 57), nous pouvons argumenter que notre implémentation à seuil de l'algorithme de cadencement de clés du SM4 en 9 cycles (voir Tableau 15) est sécurisée à l'ordre 1 en présence de glitches.

Fonctions	Nombre de cycles
$FK \oplus k$	1
ARKW	1
τ	4×6 (voir Remarque 22)
L′ et AW	1

TABLEAU 15 – Nombre de cycle pour chaque opération d'un tour de cadencement de clé du SM4

Remarque 22. *Les* 6 *cycles de l'opération* $\tau(A')$ *s'exécutent pour chacun des* 4 *octets de* A' *en parallèle.*

10. Pour la première étape, comme la variable CK_i est publique, il n'est pas nécessaire de la protéger. Comme pour FK en Figure 57, elle peut être ajoutée au premier morceau du partage additif de A' sans altérer la sécurité à l'ordre 1 en présence de glitches.

5.6 Conclusion

Dans ce chapitre, nous avons étudié la résistance face aux attaques par observations à l'ordre 1 d'un algorithme symétrique peu connu dans la littérature cryptographique, le SM4. Très utilisé en Chine, les mécanismes cryptographiques de cet algorithme intéressent les entreprises technologiques du monde entier.

Nous avons ainsi contribué à l'état de l'art en apportant des éléments de correction et une explication détaillée de chaque contre-mesures du SM4 dans le modèle d'attaque par sondage à l'ordre 1. Une implémentation logicielle en langage assembleur sur le microcontrôleur STM32F4-11RET6 Nucleo-64 de ces schémas de masquage nous a permis de comprendre les techniques de défense proposées. Une comparaison de ces dernières en termes de performances (vitesse d'exécution et mémoire) a également mis en valeur un manque dans l'état de l'art : aucune implémentation sécurisée (correcte) n'est proposée afin de satisfaire une sécurité à l'ordre 1 en présence de glitches. Afin de pallier à ce manque, nous décrivons notre propre contre-mesure du SM4. Notre construction est basée sur les implémentations à seuil et est sécurisée à l'ordre 1 en présence de glitches. En particulier, la partie non-linéaire du SM4 est une application directe de notre évaluation monomiale décrite dans le Chapitre 4.

Notre implémentation à seuil du SM4 est mathématiquement prouvée comme étant sécurisée à l'ordre 1 en présence de glitches. En travaux futurs, il serait intéressant de tester la sécurité de notre construction en pratique. Une implémentation sur FPGA permettrait par exemple de pouvoir simuler des attaques par observations à l'ordre 1 sur des chiffrements SM4.

Chapitre 6

Conclusion et perspectives

Sommaire

Dans ce chapitre nous résumons les recherches et les travaux présentés dans cette thèse. Nous concluons sur notre travail et nous étudions l'impact de nos résultats sur l'état de l'art. Plus encore, en rapport avec le sujet de thèse, nous discutons des potentielles pistes de recherches intéressantes à explorer pour des futurs projets. Enfin, nous mettons en valeur le parallèle entre les monde académiques et industriels à travers lequel se sont déroulés ces trois dernières années et nous discutons des suites possibles de cette thèse.

6.1 Conclusion générale

Le but de cette thèse était de proposer de nouveaux moyens de défense en cryptographie symétrique pour se protéger des attaques physiques par observations sur les composants électroniques. Nous nous sommes concentrés sur une contre-mesure présentée en 2006 par Nikova *et. al.* [65], les implémentations à seuil. Ces dernières permettent de contrer un adversaire dans un modèle d'attaque plus performant, capable d'exploiter la présence de glitches au sein d'un circuit électronique afin de mettre en défaut la sécurité d'un algorithme cryptographique.

Durant notre état de l'art, nous avons mis en évidence la difficulté de réaliser une implémentation à seuil d'un algorithme symétrique, sécurisée à l'ordre 1 en présence de glitches, tout en optimisant les performances en termes de vitesse d'exécution (*i.e.* nombre de cycles) et de stockage (*i.e.* nombre de bits contenus dans des registres). Les méthodes existantes ne sont pas génériques et ne sont souvent pas applicable à un ordre de sécurité supérieur à 1. Nous avons donc proposé deux implémentations à seuil du calcul d'un monôme élevé à une puissance [1]. Nos propositions sont prouvées mathématiquement sécurisées à l'ordre 1 dans le modèle d'attaque par sondage en présence de glitches. En termes de performances, pour un nombre de cycles similaires, nous innovons en proposant un espace de stockage plus faible que l'état de l'art : notre première construction nécessite 14 bits aléatoires supplémentaires tandis que la deuxième n'en requière aucun (voir Tableau 4 pour l'état de l'art). De plus, grâce au partage en morceaux de la donnée sensible, nos constructions présentent l'avantage de ne pas avoir besoin de sécuriser la fonction puissance en

1. Cette puissance doit être première avec l'ordre du sous-groupe multiplicatif impliqué dans l'algorithme.

elle-même. Nous avons illustré notre proposition sur le monôme de l'AES, très étudié dans l'état de l'art. Nous présentons ainsi les premières constructions d'implémentations à seuil génériques sur un grand nombre de fonctions monomiales et polynomiales. Cette avantage n'existait jusque-là pas dans la littérature cryptographique. Ces recherches nous ont également permis de contribuer à l'état de l'art par la proposition d'une implémentation à seuil de toutes les opérations du chiffrement de l'AES, et pas seulement de la partie non-linéaire de l'algorithme impliquant le calcul d'un monôme, comme souvent présenté dans les articles. Notre construction est entièrement prouvée mathématiquement sécurisée à l'ordre 1 dans le modèle d'attaque par sondage en présence de glitches. Enfin, un des avantages de cette thèse a été la possibilité de travailler au sein d'une équipe d'ingénieurs chez STMicroelectronics. Ce pied dans le monde industriel m'a permis de me familiariser avec d'autres domaines de compétences comme la conception d'algorithmes sur des circuits électroniques. J'ai ainsi eu l'opportunité d'implémenter et de simuler matériellement nos contre-mesures sur un circuit logique programmable (FPGA). Nos résultats expérimentaux ont montré la correction de nos propositions.

Les implémentations à seuil publiées dans l'état de l'art étant quasiment toutes issues de la méthode originale de Nikova *et. al.* [65], nous avons réfléchi à une approche différente afin de générer une construction d'une fonction polynomiale, sécurisée en présence de glitches (*i.e.* satisfaisant la propriété de glitches-immunité). Nous avons donc proposé une implémentation à seuil basée sur la méthode de Carlet *et. al.* [17] permettant de satisfaire plusieurs problématiques. Notre proposition présente en effet l'avantage d'être prouvée mathématiquement sécurisée à n'importe quel ordre en présence de glitches, et ce de manière générique, pour n'importe quelle fonction polynomiale (et pas seulement la partie non-linéaire de l'AES). De plus, nous présentons une nouvelle technique permettant de minimiser le nombre de sous-fonctions de notre implémentation à seuil.

Dans le but de participer aux enjeux industriels mondiaux, l'entreprise STMicroelectronics s'intéresse au marché chinois, en pleine expansion économique. Il devient alors important d'intégrer des implémentations matérielles d'algorithmes cryptographiques chinois dans les produits de STMicroelectronics. C'est le cas de l'algorithme symétrique SM4, publié en 2006 par l'OSCCA et très peu documenté dans la communauté cryptographique. Dans cette thèse, nous avons centralisé les informations de l'état de l'art portant sur les techniques de défense publiées pour le SM4 contre les attaques par observations dans le modèle d'attaque par sondage à l'ordre 1. Nous avons décrit et parfois corrigé/amélioré les différents schémas de protection étudiés. Après avoir exécuté une implémentation logicielle de ces derniers sur un microcontrôleur, nous les avons comparé en termes de performances. Nous avons ensuite soulevé le manque dans l'état de l'art d'une construction sécurisée dans le modèle d'attaque par sondage à l'ordre 1 en présence de glitches. Nous avons donc proposé une implémentation à seuil de toutes les opérations du chiffrement du SM4. La partie non-linéaire de l'algorithme est une application de notre implémentation à seuil d'un monôme. Nous avons également montré la correction de notre proposition en la simulant matériellement sur un FPGA.

Enfin, durant cette thèse, nous avons déposé deux brevets au sein de STMicroelectronics. Nous avons présenté nos innovations auprès du comité de brevet de l'entreprise. Elles ont toutes les deux été validées, l'une est publiée et l'autre est encore confidentielle. Pour le premier brevet, nous nous sommes intéressés à l'application des implémentations à seuil permettant de contrer un autre type d'attaque physique en cryptanalyse moderne : les attaques par fautes. Notre schéma de défense se base sur les propriétés des implémentations à seuil afin de détecter la présence d'un adversaire lors d'une attaque par fautes et de l'empêcher d'avoir de l'information sur la donnée sensible manipulée par l'algorithme cryptographique.

La section suivante présentent d'autres pistes intéressantes à étudier pour de nouvelles pers-

pectives de recherches.

6.2 Nouvelles perspectives

Durant cette thèse, nous avons démontré mathématiquement la sécurité de nos propositions dans le modèle d'attaque par sondage en présence de glitches. Nous avons également validé la correction de nos constructions grâce à des implémentations matérielles sur FPGA. Il serait intéressant de savoir provoquer volontairement des glitches dans un circuit logique afin de tester nos contre-mesures dans un environnement propice aux attaques par observations. Il n'existe en effet aucun article dans la littérature cryptographique décrivant comment forcer des glitches à apparaître au sein d'un circuit électronique pendant l'exécution d'un algorithme cryptographique.

Notre proposition d'implémentation à seuil du calcul d'un monôme est mathématiquement prouvée sécurisée à l'ordre 1 en présence de glitches. Il serait intéressant d'étudier sa résistance dans un modèle d'attaque plus performant. Il est par exemple possible d'augmenter le nombre de masques et donc de morceaux du partage de la donnée sensible, tout en respectant les propriétés des implémentations à seuil. Dans la même optique, simuler matériellement sur un FPGA notre nouvelle approche quant à la manière de générer une construction sécurisée à l'ordre supérieur en présence de glitches constitue également une piste intéressante. En troisième piste, nous pourrions également étudier la résistance de ces implémentations face à une vraie attaque par observation afin de valider en pratique leur sécurité.

Un des aspects de cette thèse a été l'encadrement de deux stagiaires de l'INSA de Rennes et Lyon, pendant 3 et 6 mois. Ces collaborations m'ont notamment permis de vulgariser mon travail de recherches afin d'évoluer vers un objectif commun entre l'étudiant en stage et moi. J'ai ainsi utilisé ma formation musicale en violon afin de transmettre les propriétés mathématiques sur lesquels se basent les implémentations à seuil à travers des concepts de musique. Il serait intéressant d'approfondir ce travail de vulgarisation dans le but de partager de nouvelles connaissance à la communauté cryptographique académique et industrielle. Ces échanges pourraient permettre à un plus grand nombre de chercheurs et d'ingénieurs de comprendre les récents mécanismes de sécurité publiés dans l'état de l'art. Cela donnerait par exemple naissance à de nouvelles idées de contre-mesures matérielles contre les attaques par observations en présence de glitches.

Durant ces trois années, nous nous sommes concentrés sur l'impact des implémentations à seuil sur les algorithmes symétriques. Il serait motivant de regarder si cette contre-mesure peut s'appliquer dans d'autres systèmes de chiffrement. Une piste très intéressante concerne la cryptographie post-quantique. Cette branche de la cryptographie est actuellement très active et très étudiée dans la recherche publique cryptographique et dans le monde industriel. Le NIST a notamment annoncé en 2016 un concours permettant de sélectionner de nouveaux algorithmes standards en cryptographie post-quantique. Le but de ce concours est de proposer de nouveaux algorithmes résistants à l'avancée en cryptanalyse due au développement des ordinateurs quantiques par de grandes entreprises comme IBM ou Google. Quatre pistes de constructions mathématiques ont été retenues par la communauté de chercheurs comme les plus sérieuses pour développer des algorithmes cryptographiques inviolables par des ordinateurs quantiques : les problèmes de vecteurs courts dans les réseaux euclidiens, la cryptographie à partir de codes correcteurs d'erreurs, celle qui s'appuie sur l'inversion de systèmes de polynômes multivariés et, enfin, une cryptographie basée sur les isogénies, soit les relations entre des courbes elliptiques. Parmi les 69 soumissions valides reçues par le NIST en 2017, 7 finalistes ont été retenus en juillet 2020, 5 ont choisi l'approche par réseaux euclidiens, 1 celle des polynômes multivariés et 1 celle des codes correcteurs d'erreurs. Nous pouvons notamment nous intéresser au candidat proposant une implémentation du cryptosystème de McEliece, un schéma de chiffrement asymétrique basé sur les codes correcteurs d'erreurs, dont la sécurité croît beaucoup plus rapidement avec la taille des clés que

pour l'algorithme RSA. Plus particulièrement nous avons remarqué la possibilité d'utiliser les implémentations à seuil pour sécuriser les opérations impliquées dans le chiffrement de McEliece, comme présenté par Chen *et. al.* [19] en 2015. Il serait très intéressant d'étudier plus en détail l'apport des implémentations à seuil sur la sécurité de ce cryptosystème ainsi que des autres algorithmes post-quantiques.

La section suivante met en valeur le parallèle entre les mondes académiques et industriels dans lesquels j'ai évolué durant cette thèse ainsi que mes choix futurs de vie professionnelle.

6.3 Discussion et suite après la thèse

La thèse CIFRE présente l'avantage de prendre en considération les enjeux industriels liés aux questions que le monde académique se posent. Pendant trois ans, j'ai pu participer à cette étroite collaboration entre chercheurs académiques et industriels afin de mener à bien mon sujet de thèse. Proposer de nouvelles contre-mesures résistantes contre les attaques par observations en présence de glitches est en effet un sujet ouvert et très étudié par les chercheurs et les ingénieurs.

D'un côté, la recherche académique vise à publier de nouveaux articles scientifiques. Les performances en temps et en stockage sont optimisées le plus possible afin d'apporter une contribution à l'état de l'art. La sécurité des constructions proposées est démontrée théoriquement (mathématiquement). C'est le cas de nos propositions où nous avons cherché à optimiser le plus possible le nombre de cycles et de bits stockés dans des registres. De l'autre côté, afin de répondre au marché industriel mondial, les entreprises technologiques privilégient la fiabilité et la sécurité de leur produits, quitte à alourdir un peu les performances du composant électronique, dans la limite du raisonnable. Durant mes recherches, j'ai ainsi appris à jongler entre ces différents objectifs afin de développer de nouvelles idées dans le monde académique (publications conférence et journal) et le domaine industriel (brevet chez STMicroelectronics).

De ma propre expérience, les interactions entre chercheurs académiques font des universités un acteur important dans le transfert de connaissances et de technologies. À mon sens, la théorie scientifique et les innovations technologiques sont étroitement liées. En contrepartie, la recherche universitaire a souvent des restrictions budgétaires qui l'empêche d'évoluer. La collaboration avec les entreprises permet de combler ce manque et d'appréhender l'innovation avec une approche plus concrète. Par exemple, dans le cadre de ma thèse, j'ai pu avoir facilement accès à des composants électroniques tels qu'un FPGA et un STM32. Cela m'a notamment permis de valider en pratique la propriété de correction des implémentations à seuil que nous proposions dans la littérature cryptographique.

Le dispositif CIFRE dont j'ai pu bénéficier chez STMicroelectronics durant ces trois dernières années m'a permis d'adapter mes connaissances théoriques à un nouvel environnement professionnel dans lequel je souhaite poursuivre par la suite. Dans l'optique de contribuer au processus d'innovation de l'entreprise, l'impact sur la société du métier d'ingénieur en cryptographie représente en ce sens une motivation intéressante vers laquelle je souhaite évoluer.

Bibliographie

[1] Imran Abbasi and Mehreen Afzal. A compact s-box design for SMS4 block cipher. *IACR Cryptol. ePrint Arch.*, 2011 :522, 2011. 87, 88

[2] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002. 24

[3] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and aes, secure against some attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001. 5, 36, 38, 92

[4] Guoqiang Bai, Hailiang Fu, Wei Li, and Xingjun Wu. Differential power attack on SM4 block cipher. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*, pages 1494–1497. IEEE, 2018. 91

[5] Xuefei Bai, Li Guo, and Tie Li. Differential power analysis attack on sms4 block cipher. In *2008 4th IEEE International Conference on Circuits and Systems for Communications*, pages 613–617, 2008. 91, 101

[6] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and practice of a leakage resilient masking scheme. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 758–775. Springer, 2012. 96

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988. 44

[8] Eli Biham. A fast new DES implementation in software. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997. 96

[9] Begül Bilgin. *Threshold implementations : as countermeasure against higher-order differential power analysis*. PhD thesis, University of Twente, Enschede, Netherlands, 2015. 49, 62, 78, 82

[10] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-offs for threshold implementations illustrated on AES. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 34(7) :1188–1200, 2015. 52, 53, 77, 103

[11] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably secure masking of AES. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004. 5, 36, 38

[12] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997. 24

[13] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004 : 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004. 26, 31, 32

[14] Cécile Canovas and Jessy Clédière. What do s-boxes say in differential side channel attacks? *IACR Cryptol. ePrint Arch.*, 2005 :311, 2005. 32

[15] Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021. 11, 47

[16] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012. 78

[17] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. *IACR Cryptol. ePrint Arch.*, 2016 :321, 2016. 40, 78, 79, 84, 111

[18] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. 5, 34, 35, 36, 38, 45

[19] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Masking large keys in hardware : A masked implementation of mceliece. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2015. 113

[20] Jiachao Chen, Qin Wang, Zheng Guo, Junrong Liu, and Haihua Gu. A circuit design of SMS4 against chosen plaintext attack. In *11th International Conference on Computational Intelligence and Security, CIS 2015, Shenzhen, China, December 19-20, 2015*, pages 371–374. IEEE Computer Society, 2015. xii, 91, 92, 94, 95, 100, 101, 102, 103

[21] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the AES s-box. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015. 7, 78

[22] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 43. ACM, 2016. 7, 52, 53, 77, 78

[23] Jean-Sébastien Coron and Louis Goubin. On boolean and arithmetic masking against differential power analysis. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware*

and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings, volume 1965 of *Lecture Notes in Computer Science*, pages 231–237. Springer, 2000. 36

[24] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013. 78

[25] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. *J. Cryptogr. Eng.*, 5(2) :73–83, 2015. 78, 79

[26] Joan Daemen. Changing of the guards : A simple and efficient method for achieving uniformity in threshold sharing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017. xii, 53, 61

[27] Joan Daemen and Vincent Rijmen. *The Design of Rijndael : AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. 10, 15, 19, 87

[28] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6) :644–654, 1976. 4

[29] Whitfield Diffie and George Ledin. SMS4 encryption algorithm for wireless networks. *IACR Cryptol. ePrint Arch.*, 2008 :329, 2008. 5, 8, 13, 86, 87, 88, 90, 103

[30] Xiaoyi Duan, Ronglei Hu, and Xiu Ying Li. Research and implementation of dpa-resistant SMS4 block cipher. In Yuping Wang, Yiu-ming Cheung, Ping Guo, and Yingbin Wei, editors, *Seventh International Conference on Computational Intelligence and Security, CIS 2011, Sanya, Hainan, China, December 3-4, 2011*, pages 1033–1036. IEEE Computer Society, 2011. xii, xiv, 91, 92, 93, 94, 95, 101, 102, 103

[31] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015. 40

[32] Horst Feistel. Cryptography and Computer Privacy. *Scientific American*, 228(5) :15–23, May 1973. 12, 13

[33] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010. 96

[34] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis : Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001. 24

[35] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Secure multiplicative masking of power functions. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, volume 6123 of *Lecture Notes in Computer Science*, pages 200–217, 2010. 5, 59

[36] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In Bart Preneel and Tsuyoshi

Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011. 35, 58

[37] Jovan Dj. Golic and Christophe Tymen. Multiplicative masking and power analysis of AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002. 5, 36, 58

[38] Louis Goubin and Ange Martinelli. Protecting AES with shamir's secret sharing scheme. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2011. 96

[39] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999. 34, 38, 44

[40] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017. 7, 52, 53, 77, 78

[41] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2011. 33

[42] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits : Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003. 40, 50, 55, 79, 96, 102, 103

[43] Kouichi Itoh, Masahiko Takenaka, and Naoya Torii. DPA countermeasure based on the "masking method". In Kwangjo Kim, editor, *Information Security and Cryptology - ICISC 2001, 4th International Conference Seoul, Korea, December 6-7, 2001, Proceedings*, volume 2288 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2001. 5

[44] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, pages 161–191, 1883. 4

[45] HeeSeok Kim, Seokhie Hong, and Jongin Lim. A fast and provably secure higher-order masking of AES s-box. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2011. 78

[46] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. 5, 24, 33

[47] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. xi, 24, 27, 30, 33

[48] Simon Landry, Yanis Linge, and Emmanuel Prouff. Monomial evaluation of polynomial functions protected by threshold implementations – illustration on the – extended version. In Yvette Bascon, editor, *Cryptography Communications Journal (2021)*. 7, 57

[49] Simon Landry, Yanis Linge, and Emmanuel Prouff. Monomial evaluation of polynomial functions protected by threshold implementations – illustration on the aes –. In Maryline Laurent and Thanassis Giannetsos, editors, *Information Security Theory and Practice - 13th IFIP WG 11.2 International Conference, WISTP 2019, Paris, France, December 11-12, 2019, Proceedings*, volume 12024 of *Lecture Notes in Computer Science*, pages 66–84. Springer, 2019. 7, 57, 77

[50] Jean Leclant. Champollion, la pierre de rosette et le déchiffrement des hiéroglyphes. 1972. 3

[51] D. Li and Y. Liu. Introduction to the commercial cryptography scheme in china. 05 2016. 86, 103

[52] Xinchao Li and Shuangpeng Ma. Design of a s-box for SMS4 based on threshold implementation. In Fatos Xhafa, Santi Caballé, and Leonard Barolli, editors, *Advances on P2P, Parallel, Grid, Cloud and Internet Computing, Proceedings of the 12th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC-2017, Barcelona, Spain, 8-10 November 2017*, volume 13 of *Lecture Notes on Data Engineering and Communications Technologies*, pages 206–214. Springer, 2017. 87

[53] Fen Liu, Wen Ji, Lei Hu, Jintai Ding, Shuwang Lv, Andrei Pyshkin, and Ralf-Philipp Weinmann. Analysis of the SMS4 block cipher. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Information Security and Privacy, 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007, Proceedings*, volume 4586 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2007. 87, 88, 91

[54] François Macé, François-Xavier Standaert, and Jean-Jacques Quisquater. Information theoretic evaluation of side-channel resistant logic styles. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2007. 33

[55] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007. 10, 20, 36

[56] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005. 5, 6, 37, 38, 40, 43, 55

[57] Louise Martinkauppi, Qiuping He, and Dragos Ilie. On the design and performance of chinese oscca-approved cryptographic algorithms. pages 119–124, 06 2020. 85, 86

[58] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000. 77

[59] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999. 26

[60] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Trans. Computers*, 51(5) :541–552, 2002. 33

[61] Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. Consolidating security notions in hardware masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3) :119–147, 2019. 47

[62] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004. 26

[63] Simon W. Moore, Robert D. Mullins, Paul A. Cunningham, Ross J. Anderson, and George S. Taylor. Improving smart card security using self-timed circuits. In *8th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2002), 9-11 April 2002, Manchester, UK*, pages 211–218. IEEE Computer Society, 2002. 33

[64] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits : A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011. 49, 52, 53, 77

[65] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006. xi, xiii, 5, 6, 7, 37, 38, 39, 43, 44, 45, 46, 47, 53, 55, 57, 83, 84, 110, 111

[66] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptol.*, 24(2) :292–321, 2011. 52

[67] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic : Dpa-resistance without routing constraints. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2005. 33

[68] Thomas Popp, Stefan Mangard, and Elisabeth Oswald. Power analysis attacks and countermeasures. *IEEE Des. Test Comput.*, 24(6) :535–543, 2007. xi, 31, 32

[69] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks : A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013. 40

[70] Sihang Pu, Zheng Guo, Junrong Liu, Dawu Gu, Yingxuan Yang, Xiaoke Tang, and Jie Gan. Boolean matrix masking for SM4 block cipher algorithm. In *13th International Conference on Computational Intelligence and Security, CIS 2017, Hong Kong, China, December 15-18, 2017*, pages 238–242. IEEE Computer Society, 2017. 91, 92, 95, 96, 99, 100, 101, 102, 103

[71] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA) : measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001. 24

[72] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015. 57, 78, 82

[73] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block ciphers implementations provably secure against second order side channel analysis. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February*

10-13, 2008, Revised Selected Papers, volume 5086 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2008. 40

[74] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010. 40, 50, 96, 99

[75] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009. 34

[76] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978. 4, 27

[77] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001. 44

[78] Thomas Roche and Emmanuel Prouff. Higher-order glitch free implementation of the AES using secure multi-party computation protocols - extended version. *J. Cryptogr. Eng.*, 2(2) :111–127, 2012. 40, 41, 96

[79] Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 417–434. Springer, 2013. 78

[80] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3) :379–423, 1948. 22

[81] Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4) :656–715, 1949. 12

[82] Takeshi Sugawara. 3-share threshold implementation of AES s-box without fresh randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1) :123–145, 2019. 52, 53, 77

[83] Kris Tiri and Ingrid Verbauwhede. A dynamic and differential CMOS logic style to resist power and timing attacks on security ic's. *IACR Cryptol. ePrint Arch.*, 2004 :66, 2004. 33

[84] Elena Trichina, Domenico De Seta, and Lucia Germani. Simplified adaptive multiplicative masking for AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2002. 36, 58

[85] Manfred von Willich. A technique with an information-theoretic basis for protecting secret data from differential power attacks. In Bahram Honary, editor, *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*, volume 2260 of *Lecture Notes in Computer Science*, pages 44–62. Springer, 2001. 96

[86] Shutong Wang, Dawu Gu, Junrong Liu, Zheng Guo, Weijia Wang, and Sigang Bao. A power analysis on SMS4 using the chosen plaintext method. In *Ninth International Conference on Computational Intelligence and Security, CIS 2013, Emei Mountain, Sichan Province, China, December 14-15, 2013*, pages 748–752. IEEE Computer Society, 2013. 91, 93, 100

[87] Weijia Wang, François-Xavier Standaert, Yu Yu, Sihang Pu, Junrong Liu, Zheng Guo, and Dawu Gu. Inner product masking for bitslice ciphers and security order amplification for linear leakages. In Kerstin Lemke-Rust and Michael Tunstall, editors, *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7-9, 2016, Revised Selected Papers*, volume 10146 of *Lecture Notes in Computer Science*, pages 174–191. Springer, 2016. 96, 99

[88] Yongzhuang Wei, Fu Yao, Enes Pasalic, and An Wang. New second-order threshold implementation of AES. *IET Inf. Secur.*, 13(2) :117–124, 2019. 7, 78

[89] Zihao Wei, Siwei Sun, Lei Hu, Man Wei, Joan Boyar, and René Peralta. Scrutinizing the tower field implementation of the 54589_{28} inverter - with applications to aes, camellia, and SM4. *IACR Cryptol. ePrint Arch.*, 2019 :738, 2019. 88

Résumé

De l'explosion du marché mondial des cartes à puce et de l'évolution des techniques en cryptanalyse naissent le besoin grandissant de sécuriser les algorithmes cryptographiques au sein d'un circuit électronique afin d'assurer la confidentialité des données privées de tout un chacun. L'objectif de cette thèse CIFRE est de contribuer à cet effort en proposant de nouvelles pistes de recherche dans l'état de l'art académique et industriel.
Notre axe d'étude principal s'articule autour de la contre-mesure des *implémentations à seuil*, résistante contre les attaques physiques par observations en présence de glitches. Ces derniers phénomènes apparaissent dans un circuit électronique de manière aléatoire et sont à l'origine de nombreuses attaques en cryptanalyse. Nous étudions l'application des implémentations à seuil sur la cryptographie symétrique.
Dans un premier temps, nous contribuons à la littérature cryptographique par la conception de nouvelles implémentations à seuil applicables sur un grand nombre d'algorithmes symétriques. Les travaux réalisés sont appuyés par des preuves mathématiques formelles et permettent de contrer les attaques physiques par observations en présence de glitches. En comparaison avec les récentes publications de l'état de l'art, nous répondons à de nouvelles problématiques tout en assurant des performances équivalentes ou meilleures. Par ailleurs, nos recherches ont également débouché à la proposition de deux brevets (acceptés et en cours de publication) au sein de l'entreprise STMicroelectronics, participant ainsi au processus d'innovation industriel.
Dans un second temps, nous nous intéressons à l'étude de l'algorithme symétrique `SM4` et sa résistance face à la cryptanalyse actuelle. Cet algorithme a été publié par l'OSCCA en 2006 dans le but de sécuriser les connexions sans fil. Réaliser une thèse au sein d'une entreprise technologique comme STMicroelectronics permet en effet de prendre part au marché mondial industriel en pleine expansion et de prendre conscience des enjeux qui en découlent. Les travaux obtenus à l'issu d'un encadrement d'un étudiant en stage de 6 mois permettent de centraliser les différentes implémentations du `SM4` sécurisées contre les attaques par observations proposées dans l'état de l'art et d'offrir une visibilité sur les performances de ces constructions. Nos recherches ont également permis d'exhiber un manque dans la littérature cryptographique concernant l'existence d'une contre-mesure résistante au modèle d'attaque par sondage en présence de glitches. Nous contribuons ainsi à la recherche académique par la première proposition d'une implémentation à seuil du `SM4`, prouvée mathématiquement résistante contre les attaques par observations en présence de glitches.

Mots clés

Algorithmes symétriques ⋆ Cryptographie ⋆ Contre-mesure ⋆ Masquage ⋆ Implémentations à seuil ⋆ Évaluation polynomiale ⋆ Cryptanalyse ⋆ Glitches ⋆ Attaques par canaux auxiliaires cachés ⋆ Attaques par analyse de consommation de courant ⋆ Circuits électroniques ⋆ `AES` ⋆ `SM4`

Abstract

Due to the market global explosion of embbeded devices and the expansion of cryptanalysis techniques, important efforts have been done to design countermeasures with provable security in order to assure confidentiality of data's customers. The goal of this thesis is to contribute to fulfilling such a need by proposing new areas of research in the academic and industrial state-of-the-art.
Our main focal axis is organized around the countermeasure called *threshold implementations* which is known to be resistant against side-channel analysis attacks in the presence of glitches. These latter phenomenon occur randomly within an electronic circuit and lead to numerous attacks in cryptanalysis. We study the application of threshold implementations on symmetric-key cryptography.
In a first phase, we participate to the cryptographic litterature by designing new threshold implementations easily applicable on a large variety of symmetric-key algorithms. Our countermeasures are provable mathematically secured against side-channel analysis attacks in the presence of glitches. In comparison with the recent publications of the state-of-the-art, we adress new issues and we assure similar or better performances. Therefore, our research has resulted in two (accepted and under publication) patents within STMicroelectronics, thereby contributing to the industrial innovation process.
In a second phase, we are interested in the study of the symmetric-key algorithm `SM4` and its resistance against side-channel analysis attacks. This algorithm has been published by the OSCCA in 2006 in order to secure wireless connections. Achieve a thesis within a technologic company such that STMicroelectronics allows to take part into the growing industrial global market and be aware of the issues involved. The works obtained from a supervision of a student for a six-months internship allow to centralize the proposed `SM4` countermeasures against side-channel analysis attacks of the state-of-the-art and offer a visibility on the software performances of these constructions. In the cryptographic litterature, we also highlight the lack of secure `SM4` implementation against side-channel analysis attacks in the presence of glitches. Hence, we finally introduce the first threshold implementation of the `SM4` algorithm. Our construction is provably mathematically resistant against side-channel analysis attacks in the presence of glitches.

Keywords

Symmetric-key algorithm ⋆ Cryptography ⋆ Countermeasure ⋆ Masking ⋆ Threshold implementations ⋆ Polynomial evaluation ⋆ Cryptanalysis ⋆ Glitches ⋆ Side-channel analysis attacks ⋆ Power analysis attacks ⋆ Electronic devices ⋆ `AES` ⋆ `SM4`