# Protection des algorithmes cryptographiques embarqués

Soline Renner

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

**DOCTEUR DE**
**L'UNIVERSITÉ DE BORDEAUX**

SPÉCIALITÉ : MATHÉMATIQUES PURES

ÉCOLE DOCTORALE
DE MATHÉMATIQUES ET INFORMATIQUES

par Soline RENNER

# Protection des algorithmes cryptographiques embarqués

Soutenue le 23 juin 2014 à l'Institut de Mathématiques de Bordeaux devant la commission d'examen composée de :

| | | |
|---|---|---|
| François-Xavier Standaert, | UCL Louvain la Neuve, Belgique, | Président |
| Philippe Gaborit, | Université de Limoges, | Rapporteur |
| Louis Goubin, | Université de Versailles, | Rapporteur |
| Gilles Zémor, | Université de Bordeaux, IMB Talence, | Directeur |
| Guilhem Castagnos, | Université de Bordeaux, IMB Talence, | Co-directeur |
| Christophe Giraud, | Oberthur Technologies, Pessac, | Encadrant |

Thèse réalisée d'octobre 2010 à juin 2014 dans le cadre d'une Convention Industrielle de Formation par la REcherche (CIFRE) à l'Institut de Mathématiques de Bordeaux et à Oberthur Technologies.

**Institut de Mathématiques de Bordeaux - UMR 5251**
Université Bordeaux 1
351, cours de la Libération
33405 TALENCE cedex
**Oberthur Technologies**
Parc scientifique Unitec 1
4, allée du Doyen Georges Brus
33600 Pessac

# Protection des algorithmes cryptographiques embarqués


**Résumé**

Depuis la fin des années 90, les cryptosystèmes implantés sur carte à puce doivent faire face à deux grandes catégories d'attaques : les attaques par canaux cachés et les attaques par injection de fautes. Pour s'en prémunir, des contre-mesures sont élaborées, puis validées en considérant un modèle d'attaquant bien défini. Les travaux réalisés dans cette thèse se concentrent sur la protection des cryptosystèmes symétriques contre les attaques par canaux cachés. Plus précisément, on s'intéresse aux contre-mesures de masquage permettant de se prémunir des attaques statistiques d'ordre supérieur pour lesquelles un attaquant est capable de cibler t valeurs intermédiaires. Après avoir rappelé l'analogie entre les contre-mesures de masquage et les schémas de partage de secret, on présente la construction des schémas de partage de secret à partir de codes linéaires, introduite par James L. Massey en 1993. En adaptant cette construction et des outils issus du calcul multi-parties, on propose une méthode générique de contre-mesure de masquage résistante aux attaques statistiques d'ordre supérieur. De plus, en fonction des cryptosystèmes à protéger et donc des opérations à effectuer, cette solution permet d'optimiser le coût induit par les contre-mesures en sélectionnant les codes les plus adéquats. Dans cette optique, on propose deux contre-mesures de masquage pour implanter le cryptosystème AES. La première est basée sur une famille de code d'évaluation proche de celle utilisée pour le schéma de partage de secret de Shamir, tandis que la seconde considère la famille des codes auto-duaux et faiblement auto-duaux ayant leur matrice génératrice à coefficient sur $\mathbb{F}_2$ ou $\mathbb{F}_4$. Ces deux alternatives se révèlent plus efficaces que les contre-mesures de masquage publiées en 2011 et basées sur le schéma de partage de secret de Shamir. De plus la seconde s'avère compétitive pour t=1 comparée aux solutions usuelles.



**Mots clés :** attaques par canaux cachés, schémas de partage de secret, codes linéaires.

# Cryptographic Protection in Embedded Systems


## Abstract

Since the late 90s, the implementation of cryptosystems on smart card faces two kinds of attacks : side-channel attacks and fault injection attacks. Countermeasures are then developed and validated by considering a well-defined attacker model. This thesis focuses on the protection of symmetric cryptosystems against side-channel attacks. Specifically, we are interested in masking countermeasures in order to tackle high-order attacks for which an attacker is capable of targeting t intermediate values. After recalling the analogy between masking countermeasures and secret sharing schemes, the construction of secret sharing schemes from linear codes introduced by James L. Massey in 1993 is presented. By adapting this construction together with tools from the field of Multi-Party Computation, we propose a generic masking countermeasure resistant to high-order attacks. Furthermore, depending on the cryptosystem to protect, this solution optimizes the cost of the countermeasure by selecting the most appropriate code. In this context, we propose two countermeasures to implement the AES cryptosystem. The first is based on a family of evaluation codes similar to the Reed Solomon code used in the secret sharing scheme of Shamir. The second considers the family of self-dual and self-orthogonal codes generated by a matrix defined over GF(2) or GF(4). These two alternatives are more effective than masking countermeasures from 2011 based on Shamir's secret sharing scheme. Moreover, for t=1, the second solution is competitive with usual solutions.



**Keywords :** side-channel attacks, secret sharing schemes, linear codes.

*Nil volentibus arduum,*

# Remerciements

C'est avec plaisir que je traite enfin ces traditionnels remerciements qui témoignent de la fin de ma vie de thésarde...

Je remercie en tout premier lieu mes encadrants de thèse qui m'ont permis de mener à bien ces travaux. Gilles Zémor, mon directeur de thèse, Guilhem Castagnos, mon co-directeur de thèse, et Christophe Giraud, mon responsable scientifique à Oberthur Technologies, m'ont été d'un grand secours tout au long de ces années. Cette thèse doit beaucoup à leurs conseils ainsi qu'à leur rigueur.

J'adresse également ma gratitude à Philippe Gaborit et à Louis Goubin qui m'ont fait le plaisir d'être les rapporteurs de cette thèse. Je suis également très reconnaissante de François-Xavier Standaert d'avoir accepté d'être membre de mon jury.

Je ne voudrais pas terminer sans remercier l'ensemble de mes co-auteurs, autant de personnes avec qui il m'a été agréable de réfléchir et d'écrire : Guillaume Barbu, Alberto Battistello, Guilhem Castagnos, Jean-Sébastien Coron, Guillaume Dabosville, Christophe Giraud, Emmanuel Prouff, Guénaël Renault, Matthieu Rivain, Praveen Kumar Vadnala, Rina Zeitoun et Gilles Zémor.

J'en profite pour remercier mes collègues côté université, notamment Olga Balkanova et Diomba Sambou, ainsi que l'ensemble de mes collègues Oberthuriens pour les trois années passées auprès d'eux. En particulier, je remercie chaleureusement mes anciens co-équipiers : Philippe Andouard, Guillaume Barbu, Alberto Battistello et Nicolas Morin.

Mes derniers remerciements vont évidemment à ma famille et mes amis pour leur soutien et leur compréhension. Je remercie tout particulièrement Jérôme de m'avoir accompagnée et soutenue durant ces années de thèse qui nous ont demandé quelques sacrifices.

# Table des matières

# Introduction

En cryptographie moderne, la sécurité d'un algorithme cryptographique est étudiée dans un modèle d'attaquant bien défini. Par exemple pour établir la sécurité d'un cryptosystème symétrique, on peut considérer le modèle dit classique qui suppose un attaquant capable d'obtenir un certain nombre de couples (message,chiffré). Son but consiste alors à déterminer la clé de chiffrement (ou encore à chiffrer/déchiffrer un message autre que ceux de sa connaissance). Si ce dernier ne parvient pas à ses fins par une méthode plus efficace qu'une recherche exhaustive sur la clé de chiffrement entière, la sécurité du cryptosystème est alors établie.

Toutefois, ce modèle n'est pas suffisant pour garantir la sécurité des cryptosystèmes implantés sur des systèmes embarqués. Un exemple concret est l'implantation des algorithmes cryptographiques sur carte à puce. Par leurs usages quotidiens dans des contextes très variés, tels que le secteur bancaire, la santé, le transport ou encore la téléphonie, on oublierait presque que ce micro-processeur répond à des fonctions nécessitant un haut niveau de sécurité réalisées à l'aide de cryptosystèmes. Notamment, ces derniers permettent de stocker des données confidentielles telles que des clés de chiffrement, de s'authentifier, d'assurer l'intégrité des données échangées...

Cependant, du fait de leurs usages, les cartes à puce s'exposent à diverses menaces et sont des cibles privilégiées des attaques. En effet, dès la fin des années 90, les cryptosystèmes supposés sûrs dans le modèle classique se sont révélés vulnérables face à deux grandes catégories d'attaques lorsque ces derniers sont implantés sur carte à puce. La première catégorie regroupe les attaques dites par injection de fautes qui consistent à engendrer une fuite d'information sensible en perturbant l'exécution d'un algorithme cryptographique. Ce type d'attaques fut initialement introduit sur carte à puce en 1996 à l'encontre de l'implantation CRT de la signature RSA, par Dan Boneh, Richard A. DeMillo et Richard J. Lipton, trois chercheurs du laboratoire *Bellcore*. Puis, peu de temps plus tard, Eli Biham et Adi Shamir proposèrent d'utiliser des perturbations pour cibler les algorithmes symétriques. La seconde catégorie d'attaques réunit quant à elle les attaques dites par canaux cachés qui furent introduites par les premières publications de Paul Kocher en 1996. Ce type d'attaques considère un adversaire capable d'obtenir des couples (message, chiffré), mais également d'observer des fuites physiques émises durant l'exécution d'un cryptosystème. Par exemple, l'observation du temps de calcul de l'algorithme ou encore de la consommation d'énergie émise lors de son exécution, sont des fuites phy-

siques pouvant dépendre des valeurs manipulées, et donc elles apportent de l'information sur les clés secrètes utilisées. Depuis leur introduction, ces deux catégories d'attaques ont un impact très important sur l'industrie de la carte à puce qui doit garantir la sécurité de ses produits tout en restant efficace en termes de temps d'exécution et de consommation mémoire. C'est pourquoi la communauté industrielle, mais aussi la communauté académique s'investissent pour élaborer des solutions pour se prémunir de ces attaques qui ne cessent d'évoluer et de se complexifier.

Dans cette thèse, on s'intéresse aux attaques par canaux cachés, et en particulier aux attaques statistiques par analyse de courant (cf. chapitre 1). Ce type d'attaque consiste à effectuer une recherche exhaustive sur une petite partie d'une clé secrète fixe en validant la bonne hypothèse à l'aide d'un traitement statistique. Ce dernier permet de corréler les consommations d'un point de fuite mesurées lors de plusieurs exécutions avec les valeurs manipulées associées. Afin de rendre pertinent ce traitement statistique, le microprocesseur est généralement modélisé par le modèle de fuite lié au poids de Hamming ou bien par celui lié à la distance de Hamming.
Comme introduit par Paul Kocher, Joshua Jaffe et Benjamin Jun à la fin des années 90, puis formalisé par Thomas S. Messerges en 2000, les attaques statistiques se généralisent en un groupe d'attaques qui suscite encore beaucoup d'intérêt de nos jours, à savoir les attaques statistiques d'ordre supérieur, aussi appelées les attaques d'ordre $t$ (cf. chapitre 2). Ce groupe d'attaques suppose un attaquant capable d'observer $t$ points de fuite durant chaque exécution. Pour se prémunir de telles attaques, des contre-mesures sont alors élaborées requérant des preuves de sécurité pour garantir leur résistance. Dans la littérature, ces dernières sont généralement menées dans le modèle de fuite lié au poids de Hamming. Cependant, on peut noter qu'en pratique un développeur devant implanter un cryptosystème de manière sécurisée sur un composant dont le modèle de fuite est lié à la distance de Hamming, peut se demander si une contre-mesure supposée sûre dans le modèle de fuite lié au poids de Hamming reste valide ou non. Pour répondre à cette question, on présente, à la fin du chapitre 2, des exemples pour lesquels porter une contre-mesure d'un modèle à un autre altère la preuve de sécurité. En particulier, ces résultats ont fait l'objet d'une publication à Cosade 2012 [CGP+12b].

De nos jours, la principale solution pour assurer la résistance des cryptosystèmes face aux attaques d'ordre $t$ est la contre-mesure de masquage à l'ordre $t+1$. Cette dernière consiste à partager chaque donnée sensible en au moins $t+1$ parts, de sorte qu'aucune information sensible ne puisse être déduite lors de l'observation de $t$ données manipulées durant l'exécution d'un algorithme. Depuis l'introduction des attaques par canaux cachés, la contre-mesure de masquage à l'ordre $t+1$ la plus couramment étudiée dans la littérature est le masquage booléen. Par exemple, on peut citer la publication de 2013 de Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain et Thomas Roche. De plus, une autre alternative fut récemment proposée indépendamment par Louis Goubin et Ange Martinelli, et Emmanuel Prouff et Thomas Roche en 2011, il s'agit du masquage dit de Shamir (basé sur l'utilisation du schéma de partage de secret de Shamir). L'avantage de

telles contre-mesures est que les opérations dites linéaires, comme l'addition entre deux données sensibles sur les parts associées ou encore la multiplication d'une donnée sensible par un scalaire sur les parts associées s'effectuent indépendamment sur chacune des parts. De telles opérations peuvent donc être facilement implantées de manière résistante aux attaques d'ordre $t$. Cependant, pour les opérations dites non linéaires telles que les multiplications entre deux données sensibles sur les parts associées, des méthodes plus complexes doivent être élaborées afin d'assurer le niveau de sécurité attendu. C'est notamment une des problématiques qui se pose pour l'implantation de la transformation *SubBytes* du chiffrement AES lorsque cette dernière est effectuée comme une exponentiation modulaire requérant des procédures d'élévations au carré et de multiplications. Pour répondre à cette problématique, de nombreuses publications sur ce sujet ont été publiées. De plus, on peut remarquer que la plupart des solutions proposées s'inspirent des méthodes déjà bien connues dans le contexte du calcul multi-parties. C'est d'ailleurs le cas de la solution proposée pour implanter une multiplication entre deux données sensibles utilisant le masquage de Shamir. Cette dernière est une adaptation de la procédure décrite pour le schéma de partage de secret de Shamir proposée indépendamment par M. Ben-Or *et al.* et D. Chaum *et al.* en 1988. Pour tirer profit des solutions proposées dans ce contexte, le chapitre 2 présente l'analogie souvent implicite entre la notion de contre-mesure de masquage et la notion de schéma de partage de secret décrite dans le contexte du calcul multi-parties. Partant de cette mise au point, il devient alors évident que les résultats bien connus dans ce contexte peuvent être adaptés pour mettre en place de nouvelles contre-mesures de masquage et établir des méthodes de calculs sécurisées dans le contexte des attaques par canaux cachés.

Dans cette continuité, l'objectif poursuivi dans cette thèse est la construction de nouvelles contre-mesures de masquage. Dans ce but, le chapitre 3 rappelle une construction de schéma de partage de secret à partir d'un code linéaire, très peu connue dans le contexte des attaques par canaux cachés. Cette construction, introduite par Massey en 1993, relie les propriétés d'un schéma à ceux d'un code linéaire. Ainsi les procédures étudiées pour les schémas de partage de secret peuvent être décrites en termes de codes linéaires. De plus les propriétés inhérentes à certains codes permettent de construire des schémas de partage de secret adéquates en fonction des opérations à effectuer. En particulier, les travaux de Ronald Cramer *et al.*, dans les années 2000, généralisent en termes de codes linéaires la procédure de multiplication, décrite pour le schéma de partage secret de Shamir en 1988. De plus, ils ont déterminé les propriétés nécessaires sur les codes linéaires pour que cette dernière soit applicable.
D'après l'analogie entre les contre-mesures de masquage et les schémas de partage de secret dans le contexte du calcul multi-parties, on propose alors de considérer la construction de Massey dans le contexte des attaques par canaux cachés, afin de définir une méthode générique pour construire des contre-mesures de masquage à l'ordre $t+1$. À titre d'exemple, cette dernière permet de définir le masquage booléen et celui de Shamir, en considérant respectivement le code de parité et le code de Reed-Solomon qui ont tous

deux la particularité d'être des codes MDS. De plus, avec une telle construction, on peut envisager de déterminer des contre-mesures de masquage les plus appropriées en fonction des opérations à effectuer. Dans ce sens, on rappelle les propriétés inhérentes de cette construction dont certaines étaient jusqu'à présent diffuses. De plus, on décrit différentes opérations de manière résistante aux attaques d'ordre $t$. En particulier, on montre que la procédure de multiplication généralisée par Ronald Cramer *et al.* en termes de codes linéaires peut être améliorée dans notre contexte.

Pour illustrer l'intérêt de cette construction de contre-mesure de masquage, on s'intéresse dans le chapitre 4 à l'implantation du chiffrement AES de manière résistante aux attaques d'ordre $t$. Jusqu'à présent, seules les contre-mesures de masquage construites à partir de codes MDS sont étudiées en pratique, car elles s'avèrent adéquats pour implanter les transformations linéaires des algorithmes avec un minimum de parts. Cependant l'utilisation de tels codes est-elle toujours la plus pertinente pour implanter la transformation $SubBytes$ ? Dans cette optique, on propose deux contre-mesures de masquage construites à partir de codes linéaires non MDS, mais ayant des propriétés particulières. Comme l'implantation de la transformation $SubBytes$ peut être effectuée à l'aide de procédures de multiplication, la première contre-mesure que l'on propose est construite à partir d'un code d'évaluation proche de celui du code de Reed-Solomon pour lequel la procédure de multiplication est applicable. De plus, pour minimiser le coût de cette dernière, on propose de partager chaque donnée sensible définies sur $\mathbb{F}_{2^8}$, en $t+2$ parts définies sur $\mathbb{F}_{2^4}$. La seconde contre-mesure que l'on propose considère quant à elle une famille de codes pour laquelle la procédure de multiplication est également applicable, mais de plus que les élévations au carré ou à la puissance 4 puissent être effectuées comme des opérations linéaires. Pour ce faire, on choisit de construire notre contre-mesure de masquage à partir de la famille des codes auto-duaux et faiblement auto-duaux ayant leur matrice génératrice à coefficient sur $\mathbb{F}_2$ ou $\mathbb{F}_4$. Cette seconde contre-mesure, ainsi que le chapitre 3 ont fait l'objet d'une publication à IMA CC 2013 [CRZ13]. Par ailleurs, on peut noter que les deux solutions que l'on propose pour implanter l'AES sont toutes les deux plus efficaces que la solution récemment publiée, basée sur le masquage de Shamir. De plus la seconde contre-mesure s'avère quant à elle compétitive comparée aux solutions proposées dans la littérature pour $t = 1$.

La structure de la thèse est la suivante : le chapitre 1 décrit les principales attaques par canaux cachés considérées dans le monde de l'embarqué ainsi que les principales contre-mesures pour s'en prémunir. Le chapitre 2 présente en détail le fonctionnement des attaques statistiques d'ordre supérieur puis explicite le lien implicite entre les contre-mesures de masquage et les schémas de partage de secret. De plus, on insiste sur le choix du modèle de fuite choisi pour mettre en place une contre-mesure. Ensuite, le chapitre 3 formalise la construction des schémas de partage de secret linéaires à partir des résultats de la théorie des codes dans le contexte des attaques par canaux cachés. De plus, dans le but d'implanter des cryptosystèmes résistants aux attaques d'ordre $t$, on analyse les

propriétés des codes nécessaires pour implanter certaines opérations dans le contexte des attaques par canaux cachés. En particulier, on propose une amélioration de la procédure de multiplication. Enfin, on suggère, dans le chapitre 4, l'application de deux contre-mesures de masquage d'ordre $t + 1$ construits à partir de code non MDS pour implanter un chiffrement AES résistant aux attaques d'ordre $t$.

# Chapitre 1

# Attaques par canaux cachés dans le monde de la carte à puce

## Sommaire



## 1.1 Introduction

C'est en 1974, par les brevets fondateurs de Roland Moreno, qu'officiellement la carte à puce fut inventée. Ses premières utilisations en France remontent aux *télécartes* proposées comme alternative au paiement en monnaie dans les cabines téléphoniques publiques, sujettes au vandalisme. La fonctionnalité de ces premières cartes consistait simplement à

décrémenter le nombre d'unités restantes, où chaque unité représentait un certain temps de communication. Depuis, par le développement du domaine du semi-conducteur et par l'essor de la cryptographie dans le domaine public depuis les années 1970, les cartes à puce ont beaucoup évolué. Elles fonctionnent aujourd'hui comme de véritables micro-ordinateurs très facilement transportables permettant d'effectuer des fonctions nécessitant un haut niveau de sécurité. Notamment, elles permettent de stocker des données confidentielles telles que des clés de chiffrement, de s'authentifier, d'assurer l'intégrité des données échangées...
Pour ces raisons, les cartes à puce occupent un rôle omniprésent dans notre vie quotidienne. On les retrouve dans le secteur bancaire, par l'utilisation des cartes de crédit et des porte-monnaie électroniques ; dans le secteur de la téléphonie mobile, par l'utilisation des cartes SIM ; dans le secteur de la santé, par l'utilisation des cartes vitale...
Assurément, les fonctions de haute sécurité requises par les diverses applications des cartes à puce, font appel à des cryptosystèmes symétriques et asymétriques, mais aussi à l'exploitation de fonctions de hachage [FIP07]. Par exemple, le cryptosystème DES [FIP77, FIP99] et son successeur l'AES [FIP01] sont utilisés pour chiffrer et vérifier l'intégrité des MAC (*Messaging Authentification Code*). L'algorithme RSA [RSA78] est par exemple utilisé par les cartes bancaires lors de la vérification du PIN chiffré, et ceux basés sur l'utilisation des courbes elliptiques tels que l'ECDSA [ANS05] sont utilisés dans les passeports électroniques. Pour plus d'information sur ces cryptosystèmes, les ouvrages suivants peuvent être consultés [Sti95, MvOV97].

Cependant les implémentations de ces cryptosystèmes sur carte à puce ont dû être adaptées afin de prendre en compte les inconvénients liés à ce dispositif embarqué. En particulier, une carte à puce dispose d'un espace mémoire limité et d'une puissance de calcul réduite, ce qui demande des optimisations d'implémentation. De plus, très vite la plupart des algorithmes cryptographiques embarqués supposés sûrs dans le modèle d'attaque à message et chiffré choisis, se sont révélés vulnérables à une famille d'attaques très particulière : les **attaques par canaux cachés**. Ces attaques tirent profit des fuites d'informations produites lors de l'exécution d'un algorithme, comme par exemple, le temps d'exécution, la consommation électrique ou encore le rayonnement électromagnétique émis. Ce type d'attaques est probablement utilisé depuis très longtemps par les services de renseignement de nombreux pays, cependant, ce n'est qu'à partir de 1996 que les attaques par canaux cachés furent appliquées sur des dispositifs embarqués tels que la carte à puce [Koc96]. En effet, l'article de Paul Kocher [Koc96] révéla l'efficacité de l'exploitation du temps d'exécution pour retrouver les clés secrètes de nombreux cryptosystèmes. Cette première publication fut suivie deux ans plus tard d'un rapport technique décrivant cette fois les attaques par analyse de consommation électrique [KJJ98, KJJ99]. Par la suite, des attaques similaires furent décrites en considérant le rayonnement électromagnétique au lieu de l'analyse de consommation électrique [GMO01, QS01]. Depuis leur introduction, cette famille d'attaques suscite un grand intérêt à la fois dans la communauté académique et la communauté industrielle. En particulier, ce type d'attaques a un impact très impor-

tant sur l'industrie de la carte à puce, qui doit garantir la sécurité des cryptosystèmes embarqués. Pour y parvenir, de nombreuses solutions, appelées contre-mesures sont proposées et étudiées par les deux communautés.

Dans la suite, après avoir décrit ce qu'est une carte à puce, on présente les trois principaux groupes d'attaques par canaux cachés, à savoir les attaques temporelles, les attaques simples par analyse de courant (ou par analyse du rayonnement électromagnétique) et les attaques statistiques par analyse de courant (ou par analyse du rayonnement électromagnétique). Puis, on présente les principales contre-mesures mises en place de nos jours pour assurer la sécurité des cartes à puces en pratique.

## 1.2 Description d'une carte à puce

Une carte à puce est une carte en plastique PVC dans laquelle est insérée un composant électronique. Le format de la carte et la position de son composant sont définis de manière très précise selon la norme ISO/IEC 7810 [ISO03]. Le composant électronique dont la surface ne doit pas excéder 25mm$^2$ est principalement composé d'un micro-processeur de 8, 16 ou 32 bits, d'un ou plusieurs crypto-processeurs permettant d'effectuer certains calculs cryptographiques (DES, AES, multiplications modulaires...), d'un générateur de nombres aléatoires, de détecteurs de sécurité destinés à détecter des conditions anormales de fonctionnement et de différents types de mémoire : RAM, ROM, EEPROM, Flash. La mémoire RAM est utilisée pour le fonctionnement du processeur, la mémoire ROM est quant à elle utilisée pour stocker le système d'exploitation et les diverses applications, enfin l'EEPROM et la mémoire Flash [1] sont des mémoires de stockage permettant de contenir les données spécifiques du possesseur de la carte. La table 1.1 explicite les caractéristiques courantes des différentes mémoires sur carte à puce. On peut noter que l'espace mémoire d'une carte à puce est très limité par rapport celui d'un ordinateur.

| RAM | ROM | EEPROM | Flash |
|---|---|---|---|
| 1 à 32 Ko | 16 à 512 Ko | 2 à 400 Ko | 96 à 1 Mo |

TABLE 1.1 – Taille courante des différentes mémoires sur carte à puce

Deux types de communication existent afin que la carte puisse communiquer avec le monde extérieur : la communication **avec contact** et la communication **sans contact**. Dans le cas de la communication avec contact, la carte à puce possède huit contacts visibles sur la pièce métallique dorée de la carte à puce, (cf. figure 1.1). Les caractéristiques liées

1. La Flash peut aussi contenir le système d'exploitation.

à ces contacts sont décrites dans les normes [ISO98, ISO99, ISO97]. En particulier, ces huit contacts ont un rôle bien défini :

- Le contact *VCC* correspond à la tension d'alimentation (5, 3 ou 1,8 volts) de la carte fournie par le lecteur.
- Le contact *RST* correspond à la commande de remise à zéro de la carte.
- Le contact *CLK* correspond à l'horloge fournie à la carte par le lecteur.
- Le contact *GND* correspond au potentiel de référence (masse) de la carte.
- Le contact *SWP* permet de communiquer comme son nom l'indique en SWP (acronyme anglais pour *Single Wire Protocol*).
- Le contact *I/O* correspond aux entrées et sorties de données entre la carte et le monde extérieur.
- La dénomination RFU signifie *Reserved for Future Use*. De nos jours, ces deux contacts servent à la communication en USB.

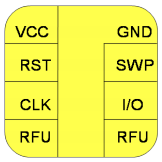



FIGURE 1.1 – Les contacts d'une carte à puce

La communication sans contact est quant à elle établie par une antenne intégrée dans la carte plastique utilisant la technologie RFID (*Radio Frequency Identification*).

## 1.3 Attaque temporelle

Une **attaque temporelle**, connue sous le nom de *Timing Attack* en anglais, exploite les différences de temps d'exécution de certains calculs cryptographiques afin d'en déduire des informations secrètes. En effet, certaines optimisations d'implémentation entraînent des sauts conditionnels qui dépendent des données traitées. Par exemple, considérons la vérification de code PIN donnée par l'algorithme 1.

**Algorithme 1** Vérification de code PIN

ENTRÉES: Soient TestPIN[.] un tableau de 4 digits à tester et CorrectPIN[.] un tableau contenant le code PIN correct
SORTIE: PIN valide ou invalide

1. **Pour** $i = 0$ *à* 3
2. **Si** (TestPIN[$i$] $\neq$ CorrectPIN[$i$]) **alors Retourner** PIN invalide
3. **Retourner** PIN valide

À partir de cette implémentation, un attaquant peut déterminer si le premier chiffre testé est correct ou non en mesurant le temps d'exécution de l'algorithme. Effectivement, lorsque le premier chiffre est correct, le second chiffre est ensuite vérifié, alors que lorsqu'il est incorrect, le résultat « PIN invalide » est directement retourné. Le temps de réponse lorsque le premier chiffre est correct est donc plus long que lorsqu'il est incorrect. En appliquant le même raisonnement sur les autres chiffres, on peut alors en déduire la bonne valeur du PIN.

Un autre exemple classique est celui publié par Paul Kocher dans [Koc96] en 1996 à l'encontre des cryptosystèmes asymétriques implantés à partir d'une simple exponentiation modulaire. Plus précisément, considérons une implémentation de la signature RSA utilisant l'exponentiation modulaire décrite par l'algorithme 2. Dans cet algorithme, la boucle d'itération requiert plus ou moins d'opérations en fonction de l'exposant privé manipulé. En effet, si le bit de l'exposant manipulé vaut 0, seule l'élévation au carré sera effectuée, alors que si le bit vaut 1, une opération supplémentaire est effectuée. Ainsi, le même principe s'applique sur la signature RSA car le temps d'exécution dépend du nombre de bits à 1 de l'exposant.

**Algorithme 2** Exponentiation modulaire utilisant la méthode *Square & Multiply*

ENTRÉES: Soient $m$ un message, $d = (1, d_{s-2}, \ldots, d_0)_2$ un exposant privé représenté sous forme binaire et $N$ un module public

SORTIE: $m^d \mod N$

1. $r \leftarrow m$
2. **Pour** $i = s - 2$ *à* $0$
3. $\quad r \leftarrow r^2 \mod N$
4. $\quad$ **Si** $d_i = 1$ **alors** $r \leftarrow mr \mod N$
5. **Retourner** $r$

Depuis l'attaque initiale de Paul Kocher [Koc96], seules quelques autres publications sur ce sujet sont parues, comme par exemple [HH98, DKL+00]. La raison principale est que les attaques temporelles s'avèrent souvent assez facile à contrer par une implémentation logicielle sur carte à puce s'exécutant en temps constant (cf. sous-section 1.6.1). Cependant il a été souligné que ces attaques devaient être considérées sérieusement, car certains mécanismes d'optimisation induits par les processeurs modernes tels que la mémoire cache [Pag02, TSS+03] ou encore par les branches de prédiction [AcKKS07] favoriseraient ce type d'attaque.

## 1.4 Attaque simple par analyse de courant (ou par analyse du rayonnement électromagnétique)

Suite à l'attaque temporelle décrite dans [Koc96], une autre attaque par canaux cachés appelée **attaque simple par analyse de courant**, (SPA : *Simple Power Analysis* en anglais) a été mise en évidence par Paul Kocher, Joshua Jaffe et Benjamin Jun dans [KJJ98, KJJ99]. Il a ensuite été montré que l'analyse du rayonnement électromagnétique émis par le composant peut également être exploitée pour monter des attaques similaires [GMO01, QS01]. Lorsque le rayonnement électromagnétique est exploité au lieu de la consommation de courant, l'attaque est alors désignée par SEMA pour *Simple Electro-Magnetic Analysis*.
Depuis les premières publications sur les attaques SPA et SEMA, celles-ci ont été considérablement exploitées pour attaquer de nombreux algorithmes cryptographiques à la fois symétriques et asymétriques. Par exemple pour les algorithmes symétriques tels que le DES et son successeur l'AES, ce sont essentiellement les transformations de diversification de clé qui ont été attaquées [BS99, Man02]. Pour les algorithmes asymétriques, les cibles visées par ces attaques sont principalement l'exponentiation modulaire, notamment dans le cas de la signature RSA [MDS99b] ainsi que la multiplication scalaire des courbes elliptiques [Osw02].

Dans la suite, on présente le principe général d'une attaque simple par analyse de courant. Il est à noter qu'une attaque par analyse électromagnétique se déroule de façon similaire, seule la méthode de mesure de la fuite diffère.

### 1.4.1 Principe général

Une attaque simple par analyse de courant permet d'extraire de l'information dépendant d'une donnée secrète, en observant la consommation de courant produite lors d'une seule exécution de l'algorithme ciblé. Pour mener une telle attaque, l'attaquant doit disposer d'une carte à puce placée dans un lecteur de carte, d'un ordinateur, d'un oscilloscope et d'une sonde pour mesurer la consommation de courant. Dans un premier temps, l'ordinateur envoie une commande au lecteur de carte, par exemple l'exécution d'un chiffrement d'un message donné, qui est ensuite exécutée par la carte à puce. En parallèle, l'oscilloscope affiche la consommation de courant obtenue via une sonde mesurant la différence de tension au borne d'une résistance placée sur la sortie VCC de la carte à puce.

**Remarque 1.4.1.** *Pour mener une attaque SEMA, la sonde est placée au plus près du composant de la carte. Ce type de mesure permet d'observer une partie précise du composant, tel que le crypto-processeur, à la différence de l'analyse de consommation électrique qui reflète l'activité globale de la puce.*

Une fois la mesure de consommation effectuée, elle peut directement être interprétée. En supposant que la consommation d'un micro-processeur reflète l'activité interne du composant, son analyse peut permettre de déterminer soit un événement particulier (comme une suite d'instructions), soit une valeur manipulée. Plus précisément, on présente dans la sous-section suivante la manière de tirer profit de ces deux types d'observation lorsqu'ils dépendent d'une donnée secrète.

### 1.4.2 Interprétation des fuites observées

#### 1.4.2.1 Observation d'un événement

Lorsqu'une suite d'instructions particulières est exécutée en fonction de la valeur d'une clé secrète, l'observation de tels événements permet alors de retrouver cette valeur. Par exemple, l'observation illustrée par la figure 1.2 correspond à une mesure du rayonnement électromagnétique émis durant l'exécution de la signature RSA implantée avec la méthode *Square* & *Multiply* (Alg. 2). Cette méthode d'exponentiation effectue à chaque

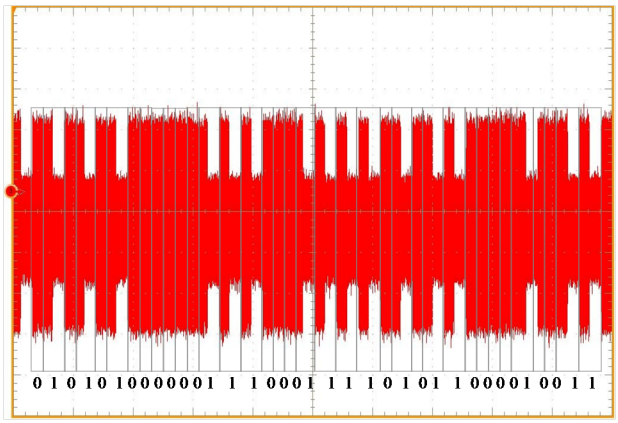



FIGURE 1.2 – Rayonnement électromagnétique émis durant l'exécution d'une exponentiation modulaire utilisant la méthode *Square* & *Multiply* (Alg. 2)

tour de boucle une élévation au carré et une multiplication seulement si la valeur du bit de l'exposant privé vaut 1. Un attaquant est donc capable de retrouver chacun des bits de l'exposant privé en déterminant lors de chaque tour de la boucle, si un motif supplémentaire est effectué ou non.

#### 1.4.2.2 Observation d'une valeur

Afin de pouvoir déterminer la valeur d'une clé secrète manipulée lors de l'exécution d'un algorithme cryptographique, les fuites d'information émises par le composant doivent être caractérisées par un **modèle de fuite** aussi pertinent que possible. Généralement

dans l'environnement de la carte à puce, la consommation du composant est supposée proportionnelle au poids de Hamming des octets manipulés (cf. sous-section 1.5.1.1). Pour retrouver le poids de Hamming d'un octet de la valeur d'une clé secrète manipulée, l'attaquant se constitue dans un premier temps une bibliothèque de mesures de consommation suivant les 9 valeurs de poids possibles. Ensuite, il exécute l'algorithme à attaquer et récupère la mesure de consommation observée lors de la manipulation de la clé secrète. En comparant cette mesure de consommation avec celles de sa bibliothèque, il pourra alors en déduire la valeur du poids de Hamming de chaque octet secret.

**Remarque 1.4.2.** *Par la suite, l'exploitation de la dépendance d'une mesure en fonction des données manipulées a conduit à la classe d'attaques appelée en anglais Template Attacks [CRR02, RO04, APSQ06].*

## 1.5 Attaque statistique par analyse de courant (ou par analyse du rayonnement électromagnétique)

L'analyse de la consommation de courant (ou du rayonnement électromagnétique) peut aussi être exploitée par les attaques dites statistiques. À la différence d'une attaque simple, une telle attaque nécessite l'observation de fuites d'informations mesurées lors de plusieurs exécutions utilisant la même clé secrète. Dans ce cas, l'attaque consiste à effectuer une recherche exhaustive sur un sous-ensemble de $l$ bits de cette clé fixe (typiquement $l = 1$ ou $l = 8$) à l'aide d'un traitement statistique. Plus précisément, le fonctionnement général d'une attaque statistique par analyse de courant est présenté en sous-section 1.5.1, puis afin d'illustrer la mise en pratique de ce type d'attaque, deux exemples sont décrits en sous-section 1.5.2.

**Remarque 1.5.1.** *Les attaques statistiques par analyse du rayonnement électromagnétique peuvent être menées de façon similaire aux attaques par analyse de courant.*

**Notation 1.5.2.** *Pour plus de clarté, on notera dans la suite en lettre majuscule (par exemple : $X$, $Z, \ldots$) les variables uniformément distribuées sur un ensemble donné et en lettre minuscule (par exemple : $x$, $z, \ldots$) les valeurs prises par ces variables.*

### 1.5.1 Principe général

Une attaque statistique requiert les mêmes dispositifs que lors d'une attaque simple, à savoir une carte à puce placée dans un lecteur de carte, un ordinateur, un oscilloscope et une sonde (cf. sous-section 1.4.1). Notons $k^*$, $l$ bits fixes d'une clé secrète à retrouver lors de chaque exécution d'un algorithme prenant en entrée un message. L'attaque consiste dans un premier temps à identifier une variable sensible $X_{k^*}$ (manipulée par l'algorithme).

**Définition 1.5.3** (Variable sensible)**.** *Une variable est dite sensible si elle dépend de $k^*$, un petit nombre $l$ de bits d'une clé secrète fixe, et d'une variable $M$ uniformément distribuée sur un ensemble fini de messages :*

$$X_{k^*} = f(M, k^*) \ , \tag{1.1}$$

*où la fonction $f$ est appelée* ***fonction de sélection****.*

Une fois la variable $X_{k^*}$ ciblée, l'ordinateur envoie une commande au lecteur de carte de sorte que la carte à puce exécute $n > 1$ fois l'algorithme avec la même clé secrète et avec des messages $m_i$ connus de l'attaquant, pour $i = 1, \ldots, n$. En parallèle, l'attaquant mesure la consommation de courant produite lors de chacune de ces exécutions. Plus précisément, à chaque exécution $i$ de l'algorithme prenant en entrée $m_i$, il récupère une courbe de points représentant la consommation de courant mesurée en fonction d'un intervalle de temps $\tau = \{t_0, \ldots, t_{T-1}\}$ :

$$\mathcal{L}_i = \{\mathcal{L}_i(t_0), \ldots, \mathcal{L}_i(t_{T-1})\} \ , \tag{1.2}$$

avec $\mathcal{L}_i(t) \in \omega$ pour tout $i = 1, \ldots, n$ et pour tout $t \in \tau$, où $\omega$ est un ensemble discret quelconque.

Pour retrouver les $l$ bits de sous-clé $k^*$, l'attaquant suppose que pour tout $i = 1, \ldots, n$, il existe un même point $t \in \tau$ pour lequel la consommation mesurée $\mathcal{L}_i(t)$ dépend de la valeur $x_{i,k^*} = f(m_i, k^*)$. De plus, il fait les hypothèses suivantes en fonction des $l$ bits possibles de sous-clé $k$ :

- Si $k \neq k^*$, alors les ensembles $\{\mathcal{L}_i(t)\}_{i=1}^n$ et $\{x_{i,k} = f(m_i, k)\}_{i=1}^n$ sont indépendants,
- Si $k = k^*$, alors les ensembles $\{\mathcal{L}_i(t)\}_{i=1}^n$ et $\{x_{i,k} = f(m_i, k)\}_{i=1}^n$ sont corrélés.

Pour mettre en évidence de telles hypothèses, l'attaquant détermine pour chaque $l$ bits possibles de sous-clé $k$, les valeurs pouvant être prises par la relation (1.1) sachant les messages $m_i$ pour $i = 1, \ldots, n$. Pour chaque $l$ bits possibles $k$, il calcule les ensembles suivants :

$$\{x_{i,k} = f(m_i, k)\}_{i=1}^n \ . \tag{1.3}$$

Ensuite, il modélise la consommation de la carte (cf. sous-section 1.5.1.1) afin de déterminer pour chaque hypothèse de sous-clé $k$, l'ensemble des consommations théoriques $\{y_{i,k}\}_{i=1}^n$ dépendant de l'ensemble $\{x_{i,k}\}_{i=1}^n$. Enfin, il compare l'ensemble des consommations mesurées $\{\mathcal{L}_i(t)\}_{i=1}^n$ avec chacun des ensembles de consommations théoriques $\{y_{i,k}\}_{i=1}^n$, pour tout $k$, via l'application de traitements statistiques. L'attaquant pourra alors en déduire la bonne valeur de sous-clé $k = k^*$, en distinguant la dépendance maximale parmi les $n$ comparaisons effectuées. En pratique, les deux traitements statistiques les plus utilisés sont :

- la différence des moyennes,
- le coefficient de corrélation linéaire de Pearson.

Ces deux traitements statistiques sont présentés dans la sous-section 1.5.1.2.

Jusqu'à présent, on a supposé l'existence d'un instant précis $t \in \tau$ de l'algorithme pour lequel la consommation $\mathcal{L}_i(t)$ dépend de la valeur $x_{i,k^*}$, pour tout $i = 1, \ldots, n$. Cependant, en pratique, cet instant n'est pas connu au préalable. Le traitement statistique est alors appliqué en chacun des points de l'intervalle de temps $\tau$. Une attaque statistique requiert alors d'appliquer $2^l$ fois le traitement statistique considéré pour chacun des points de $\tau$. Cet intervalle dépend du réglage de l'oscilloscope, mais aussi de la possibilité d'encadrer plus ou moins précisément l'instant $t$ où la variable $X_{k^*}$ est manipulée. En pratique $\tau$ varie de plusieurs milliers de points à des dizaines de milliers. De plus, le nombre de courbes nécessaires pour réussir une attaque (*i.e.,* $n$) dépend de plusieurs paramètres, notamment, du modèle de consommation du composant attaqué ainsi que du bruit des mesures. Le nombre de courbes utilisées pour tenter de réussir une attaque varie de plusieurs centaines de courbes à 10.000.000. Au-delà, il est courant de considérer l'attaque statistique comme ayant échoué. Un compromis entre le paramètre $n$ et l'intervalle $\tau$ est alors à déterminer au début de l'attaque afin de la rendre réalisable en un temps donné.

#### 1.5.1.1 Modèles de fuite

De nos jours, la plupart des circuits intégrés sont basés sur la technologie CMOS (*Complementary Metal Oxide Semiconductor*, en anglais). Un tel micro-processeur est modélisé par une machine d'état dont la consommation dépend du nombre de bits qui bascule d'un état à l'autre entre deux coups d'horloge [CJRR99b]. Par exemple, considérons $X \in \mathbb{F}_{2^l}$ une variable qui va être stockée à un instant $t$ dans un registre (notée sous forme binaire par : $(x_0, \ldots, x_{l-1})_2$), et $R \in \mathbb{F}_{2^l}$ l'état de référence qui correspond au précédent contenu de ce registre (noté sous forme binaire par : $(r_0, \ldots, r_{l-1})_2$). Notons $a_{01}$ (resp. $a_{10}$) la quantité de courant nécessaire pour faire basculer un bit de l'état 0 à l'état 1 (resp. de 1 vers 0). La consommation après l'écriture de $X$ dans le registre (à un instant $t$) peut alors être modélisée par la relation suivante [BK02] :

$$\sum_{i=0}^{l-1} a_{01} \times (1 - x_i) \times r_i + \sum_{i=0}^{l-1} a_{10} \times (1 - r_i) \times x_i + B \ , \tag{1.4}$$

où la variable $B$ est un bruit de moyenne $\mu$ représentant la consommation du reste du circuit.
Lorsque les quantités $a_{01}$ et $a_{10}$ sont constantes et égales à une même valeur $a$, alors le modèle de consommation décrit par la relation (1.4) peut être simplifié par le **modèle lié à la distance de Hamming** comme proposé dans [BCO03, BCO04].

**Définition 1.5.4** (Modèle lié à la distance Hamming)**.** *La consommation après l'écriture de $X$ dans un registre (à un instant $t$) modélisée par le modèle lié à la distance de Hamming*

*est définie par la relation suivante :*

$$a \times \mathrm{dist}(X, R) + B \ , \tag{1.5}$$

*où $R$ est l'état de référence du registre et* dist *est la distance de Hamming définie par :*

$$\mathrm{dist}(X, R) = \mathrm{card}(\{i : i \in \{0, \ldots, l-1\}, x_i \neq r_i\}) \ . \tag{1.6}$$

De plus, si on considère que l'état de référence $R$ est toujours égal à zéro, le modèle de consommation se simplifie par le **modèle lié au poids de Hamming** de la variable manipulée [KJJ99, MDS99a].

**Définition 1.5.5** (Modèle lié au poids de Hamming)**.** *La consommation après l'écriture de $X$ dans un registre (à un instant $t$) modélisée par le modèle lié au poids de Hamming est définie par la relation suivante :*

$$a \times \mathrm{w_H}(X) + B \ , \tag{1.7}$$

*où $\mathrm{w_H}(X)$ est la fonction poids de Hamming définie par :*

$$\mathrm{w_H}(X) = \mathrm{card}(\{i : i \in \{0, \ldots, l-1\}, x_i \neq 0\}) \ . \tag{1.8}$$

**Remarque 1.5.6.** *Plusieurs études ont démontré la validité des modèles liés à la distance de Hamming et au poids de Hamming en pratique, comme par exemple [MOP07, PSQ07]. Pour caractériser le modèle de consommation de la carte à attaquer, une solution consiste à récupérer la courbe de consommation de courant produite lors de l'exécution d'une boucle d'instructions manipulant des valeurs aléatoires connues, puis à comparer cette mesure avec les valeurs manipulées.*

#### 1.5.1.2 Méthodes statistiques

Dans cette sous-section, on présente deux des traitements statistiques les plus utilisés en pratique pour effectuer une attaque statistique par analyse de courant, à savoir la différence des moyennes et le coefficient de corrélation linéaire de Pearson. En particulier, lorsque l'attaque statistique utilise la différence des moyennes (resp. le coefficient de corrélation de Pearson), l'attaque est alors appelée DPA : acronyme anglais pour *Differentiel Power Analysis* (resp. CPA : acronyme anglais pour *Correlation Power Analysis*). Un comparatif de ces deux méthodes a été mené par Régis Bévan dans son mémoire de thèse [Bév04]. En général, le coefficient de corrélation linéaire donne de meilleurs résultats que la méthode utilisant la différence des moyennes. Notamment, le nombre de courbes de mesures $n$ nécessaires pour réussir l'attaque est moindre pour la CPA. Ceci s'explique par le fait que le traitement statistique utilisé pour les CPA est un outil plus fin prenant en compte des moments d'ordre 2, contrairement à la DPA se limitant à l'analyse des moments d'ordre 1.

**Différence des moyennes**

La différence des moyennes est le premier traitement statistique à avoir été proposé pour effectuer une attaque statique par analyse de courant [KJJ98, KJJ99]. Initialement, cette méthode permettait de retrouver la clé secrète bit par bit. Puis, peu de temps plus tard, la DPA s'est généralisée afin de retrouver la clé secrète $l$ bits par $l$ bits, avec $l \geqslant 1$ [Mes00a, MDS02].

Présentons cet outil statistique ciblant un octet de sous-clé. D'après la remarque 1.5.6, supposons la consommation de la carte liée au poids de Hamming des variables manipulées. Avec les mêmes notations que précédemment, le traitement consiste, pour tous les octets de sous-clé possibles $k$, à séparer l'ensemble des courbes de consommation mesurées $\{\mathcal{L}_i\}_{i=1}^{n}$, en deux groupes définis par :

$$\begin{array}{l} G_{0,k} = \{\mathcal{L}_i \mid \mathrm{w_H}(x_{i,k}) \leqslant 4 : i = 1, \ldots, n\} \\ G_{1,k} = \{\mathcal{L}_i \mid \mathrm{w_H}(x_{i,k}) > 4 : i = 1, \ldots, n\} \end{array} \tag{1.9}$$

L'attaquant détermine ensuite, pour tout $k$, la différence des moyennes entre chacun des deux groupes, en chacun des points $j \in \tau$ :

$$\Delta_k(j) = \frac{\sum_{\mathcal{L}_i \in G_{0,k}} \mathcal{L}_i(j)}{\mathrm{card}(G_{0,k})} - \frac{\sum_{\mathcal{L}_i \in G_{1,k}} \mathcal{L}_i(j)}{\mathrm{card}(G_{1,k})} \ . \tag{1.10}$$

D'après le modèle de fuite considéré (cf. définition 1.5.5), supposons à l'instant $t$ où la variable $X_{k^*}$ est manipulée, la consommation de courant modélisée par une variable $Y$ définie par :

$$Y = a \times \mathrm{w_H}(X_{k^*}) + B \ , \tag{1.11}$$

avec $a$ une constante non nulle, et $B$ un bruit de moyenne $\mu$. D'après cette hypothèse, l'attaquant peut alors supposer que pour tout $k$, la différence des moyennes $\Delta_k(t)$ calculée se rapproche de la différence entre l'espérance de $Y$ sachant $\mathrm{w_H}(X_k) \leqslant 4$ et l'espérance de $Y$ sachant $\mathrm{w_H}(X_k) > 4$. Deux cas de figure se présentent alors :

- Lorsque $k = k^*$, l'espérance de $Y$ sachant $\mathrm{w_H}(X_k) \leqslant 4$ vaut $\frac{512a}{163} + \mu$ , tandis que l'espérance de $Y$ sachant $\mathrm{w_H}(X_k) > 4$ vaut $\frac{512a}{93} + \mu$ . Par conséquent, la différence des moyennes $\Delta_k(t)$, lorsque $k = k^*$, s'approche d'une constante non nulle (*i.e.,* de $\frac{131072a}{15129} \approx 8,66a$).
- Lorsque $k \neq k^*$, les variables $Y$ et $X_k$ sont indépendantes, et donc l'espérance de $Y$ sachant $\mathrm{w_H}(X_k) \leqslant 4$ et celle de $Y$ sachant $\mathrm{w_H}(X_k) > 4$ sont toutes deux égales à l'espérance de $Y$. Dans ce cas, la différence des moyennes $\Delta_k(j)$, pour tout $j \in \tau$ et pour tout $k \neq k^*$ se rapproche de 0.

Parmi les 256 octets de sous-clé possibles $k$, l'attaquant pourra alors en déduire la bonne hypothèse $k = k^*$ en déterminant l'octet $k$ pour lequel la courbe de corrélation $\{\Delta_k(j) : j \in \tau\}$ possède un point maximal en valeur absolue (*i.e.,* le plus proche de $\frac{131072a}{15129}$). Par

ailleurs, ce point permet d'identifier l'instant précis $t \in \tau$ où les valeurs $\{x_{i,k^*}\}_{i=1}^{n}$ sont manipulées.

On peut noter que par cette méthode, l'influence du bruit est supposée ne pas intervenir. Cependant, en pratique, les différences des moyennes calculées peuvent être fortement influencées par les moyennes du bruit calculées, si ces dernières sont non négligeables. Ceci pourra avoir comme effet de rendre la courbe de corrélation associée à la bonne hypothèse de clé indistinguable parmi les autres courbes.

### Coefficient de corrélation linéaire de Pearson

Le coefficient de corrélation de Pearson est un outil statistique qui permet de quantifier la dépendance linéaire entre deux variables $X$ et $Y$. Ce coefficient est une normalisation de la covariance par le produit des écarts-types des variables :

$$\rho_{X,Y} = \frac{\mathrm{cov}\,(X,Y)}{\sigma_X \sigma_Y}\ . \tag{1.12}$$

On peut noter que lorsque les deux variables $X$ et $Y$ sont indépendantes, alors $\rho_{X,Y} = 0$. Lorsque l'on considère $Y = aX + b$, avec $a$ et $b$ deux constantes non nulles, alors la corrélation $\rho_{X,Y}$ est égale au signe de $a$. En effet, on obtient :

$$\rho_{X,Y} = \frac{cov\,(X, aX+b)}{\sigma_X \sigma_{aX+b}} = \frac{a\ cov(X,X)}{|a|\sigma_X \sigma_X} = \frac{a\ var(X)}{|a| var(X)} = sign(a)\ . \tag{1.13}$$

De plus, lorsque l'on considère $Y = aX + B$, avec $a$ une constante non nulle et $B$ une variable aléatoire indépendante de $X$, on a :

$$\begin{array}{rcl} cov(X,Y) & = & a\ cov(X, X+B) = a\ (E[X(X+B)] - E[X]E[X+B]) \\ & = & a\ (E[X^2] + E[XB] - E[X]^2 - E[X]E[B]) \\ & = & a\ cov(X,X) = a\ var(X) \end{array} \tag{1.14}$$

et

$$\begin{array}{rcl} var_Y & = & var_{aX+B} = E[(aX+B)^2] - E[aX+B]^2 \\ & = & E[a^2X^2 + B^2 + 2aXB] - E[aX]^2 - E[B]^2 - 2E[aX]E[B] \\ & = & a^2(E[X^2] - E[X]^2) + (E[B^2] - E[B]^2) \\ & = & a^2\ var(X) + var(B)\ . \end{array} \tag{1.15}$$

Dans ce cas, la corrélation $\rho_{X,Y}$ devient alors :

$$\rho_{X,Y} = \frac{a\ var\,(X)}{\sqrt{var\,(X)}\sqrt{a^2 var(X) + var(B)}} = sign(a)\ \frac{\sqrt{var\,(X)}}{\sqrt{var(X) + \frac{var(B)}{a^2}}}\ . \tag{1.16}$$

Lorsque le coefficient de corrélation de Pearson est appliqué sur des échantillons de même taille : $\{x_i\}_{i=1}^n$ et $\{y_i\}_{i=1}^n$, on parle alors du coefficient de corrélation de Pearson empirique, qui est alors défini par :

$$\widehat{\rho}(\{x_i\}_{i=1}^n, \{y_i\}_{i=1}^n) = \frac{n\sum_{i=1}^n x_i y_i - \left(\sum_{i=1}^n x_i\right)\left(\sum_{i=1}^n y_i\right)}{\sqrt{n\sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i\right)^2}\sqrt{n\sum_{i=1}^n y_i^2 - \left(\sum_{i=1}^n y_i\right)^2}} \ . \tag{1.17}$$

Plus ce coefficient de corrélation est proche de 1 (en valeur absolue), plus la relation de dépendance entre $\{x_i\}_{i=1}^n$ et $\{y_i\}_{i=1}^n$ est forte.

Le coefficient de corrélation de Pearson empirique est donc un outil bien adapté pour retrouver la bonne hypothèse de sous-clé $k^*$ [dBLW02, BCO04, Man04]. En effet, lorsque le modèle de fuite de la carte est supposé lié au poids de Hamming, chaque consommation mesurée $\{\mathcal{L}_i\}$, pour $i = 1, \ldots, n$, est supposée égale à $a \times \mathrm{w_H}(x_{i,k^*}) + B$ (cf. sous-section 1.5.1.1). Dans ce cas, pour tout $l$-uplet de bits de sous-clé possible $k$, l'attaquant applique le coefficient empirique de Pearson entre l'ensemble des valeurs prédites $\{y_{i,k}\}_{i=1}^n = \{\mathrm{w_H}(x_{i,k})\}_{i=1}^n$ et l'ensemble des consommations mesurées $\{\mathcal{L}_i\}_{i=1}^n$ $(= a \times \mathrm{w_H}(x_{i,k^*}) + B)$. L'attaquant détermine alors pour tout $k$ et en chacun des points $j \in \tau$,

$$\widehat{\rho}(\{y_{i,k}\}_{i=1}^n, \{\mathcal{L}_i(j)\}_{i=1}^n) \ . \tag{1.18}$$

L'attaquant obtient ainsi pour chacune des hypothèses de sous-clés possibles $k$, une courbe de corrélation : $\{\widehat{\rho}(\{y_{i,k}\}_{i=1}^n, \{\mathcal{L}_i(j)\}_{i=1}^n) : j \in \tau\}$. À partir de la relation (1.16), on peut voir que plus la variance du bruit $B$ est proche de zéro, plus le coefficient empirique associé à l'ensemble $\{x_{i,k^*}\}_{i=1}^n$ se rapproche de $sign(a)$. Pour les autres hypothèses de sous-clé $k \neq k^*$, les ensembles comparés par le coefficient empirique étant statistiquement indépendants, chacun des points des courbes de corrélation associées se rapprochent donc de zéro. Parmi les courbes de corrélation calculées, l'attaquant pourra alors en déduire la bonne hypothèse $k = k^*$ en déterminant la valeur de $k$ pour laquelle la courbe de corrélation associée possède un point maximal en valeur absolue (*i.e.,* le plus proche de 1).

## 1.5.2 Exemples

Dans cette sous-section, on présente deux exemples d'attaques statistiques par analyse de courant utilisant le coefficient de corrélation linéaire de Pearson empirique : l'un ciblant un chiffrement AES-128 et l'autre ciblant une signature RSA.

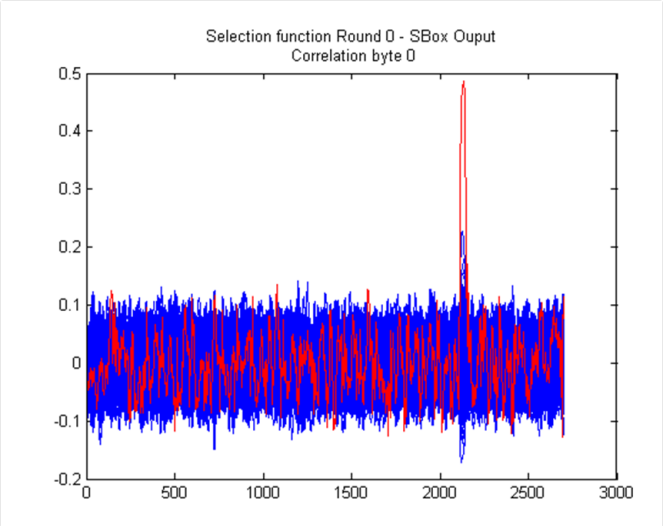
### 1.5.2.1 Cas d'un chiffrement AES

Pour retrouver le premier octet de la clé de chiffrement, l'attaquant peut, par exemple, identifier comme variable sensible le premier octet issu de la transformation *SubBytes* au

premier tour du chiffrement AES :

$$X_{k^*} = Sbox(M + k^*), \tag{1.19}$$

où la variable $M$ et $k^*$ représentent respectivement le premier octet du message utilisé lors d'un chiffrement et le premier octet de la clé secrète.
Avec les mêmes notations que précédemment, l'attaquant effectue alors $n$ chiffrements avec des messages indépendants de sorte qu'il puisse récupérer les couples $(m_i,\mathcal{L}_i)$, pour $i = 1, \ldots, n$. D'après la remarque 1.5.6, l'attaquant peut considérer le modèle de fuite de consommation lié au poids de Hamming. Dans ce cas, il prédit, pour chaque hypothèse de sous-clé $k$, l'ensemble $\{y_{i,k}\}_{i=1}^n = \{\mathrm{w_H}(x_{i,k})\}_{i=1}^n$ qui correspond à l'ensemble des quantités ayant une corrélation affine avec la consommation à l'instant $t$ où les valeurs hypothétiques $\{x_{i,k}\}_{i=1}^n$ sont manipulées. Enfin, il applique le coefficient empirique de Pearson entre l'ensemble des courbes de consommation mesurées $\{\mathcal{L}_i\}_{i=1}^n$ et l'ensemble $\{y_{i,k}\}_{i=1}^n$, pour tout $k$, et en chacun des points $j \in \tau$. Il obtient alors 256 courbes de corrélations, comme illustré par la figure 1.3 qui indique le coefficient de corrélation calculé en fonction du temps (en $\mu s$) pour chaque hypothèse de sous-clé. Il pourra en déduire la bonne hypothèse de sous-clé




*L'ordonnée indique le coefficient de corrélation calculé suivant l'hypothèse de sous-clé considérée en fonction du temps en $\mu s$ donné en abscisse. La courbe rouge correspond à la courbe de corrélation associé à la bonne hypothèse de sous-clé, tandis que les 255 autres courbes bleues correspondent aux 255 courbes de corrélation associées aux mauvaises hypothèses de sous-clés.*

FIGURE 1.3 – Résultat d'une CPA menée en pratique sur une carte à puce

$k$, en déterminant la courbe de corrélation qui admet un point de corrélation maximal. En

particulier, sur la figure 1.3, le point de corrélation entre $2000\mu s$ et $2500\mu s$ appartenant à la courbe explicitée en rouge, permet à l'attaquant d'en déduire la bonne hypothèse de sous-clé. De plus, sur cet exemple, il a fallu environ 3000 courbes (*i.e.,* prendre $n \geqslant 3000$) afin de rendre ce point de corrélation distinguable.

#### 1.5.2.2 Cas d'une signature RSA

À présent, considérons un attaquant ciblant un bit de l'exposant privé $d$ utilisé lors de $n$ signatures RSA avec des messages ($m_i \mod N$) indépendants. Notons la représentation binaire de $d$ par $(1, d_{s-2}, \ldots, d_0)_2$ et supposons l'exponentiation modulaire implantée en utilisant la méthode *Montgomery Ladder* (Alg. 4). Pour retrouver le bit $d_{s-2}$, l'attaque consiste dans un premier temps à identifier les variables $M^j \mod N$ pouvant être manipulées en fonction de la valeur prise par le bit $d_{s-2}$. D'après la table 1.2, on peut constater que :

- la variable $M^2 \mod N$ est manipulée seulement lorsque $d_{s-2} = 0$,
- la variable $M^7 \mod N$ est manipulée seulement lorsque $d_{s-2} = 1$.

| $(1, d_{s-2}, d_{s-3})$ | valeurs manipulées modulo $N$ : |
|---|---|
| (1,0,0) | $m^3$, $m^2$, $m^5$, $m^4$, $m^9$, ... |
| (1,0,1) | $m^3$, $m^2$, $m^5$, $m^6$ , $m^{11}$, ... |
| (1,1,0) | $m^3$, $m^4$, $m^7$, $m^6$ , $m^{13}$, ... |
| (1,1,1) | $m^3$, $m^4$, $m^7$, $m^8$ , $m^{15}$, ... |

TABLE 1.2 – Valeurs manipulées en fonction des valeurs de bits $(1, d_{s-2}, d_{s-3})$ lors d'une exponentiation modulaire ($m^d \mod N$) utilisant la méthode *Montgomery Ladder* (Alg. 4), où $d = (1, d_{s-2}, \ldots, d_0)$

Par conséquent, l'attaquant peut choisir les variables sensibles suivantes :

$$\begin{cases} X_0 = M^2 \mod N \\ X_1 = M^7 \mod N \end{cases} \tag{1.20}$$

Une fois les couples (message $m_i$, courbe de consommation $\mathcal{L}_i$) récupérés, pour $i = 1, \ldots, n$, en supposant le modèle de fuite lié au poids de Hamming, l'attaquant prédit les valeurs pouvant être prises par les variables :

$$\begin{cases} Y_0 = {\mathrm{w_H}}^+(M^2 \mod N) \\ Y_1 = {\mathrm{w_H}}^+(M^7 \mod N) \end{cases} \tag{1.21}$$

où ${w_H}^+(Z)$ correspond au poids de Hamming de l'octet de poids fort de chacune des valeurs prise par la variable $Z$. Ensuite en comparant l'ensemble des courbes de consommation obtenu lors de l'exécution des signatures RSA (*i.e.,* $\{\mathcal{L}_i\}_{i=1}^n$), avec les valeurs prises par variables $Y_0$ et $Y_1$, la valeur du bit $d_{s-2}$ pourra être déduite :

- si une corrélation apparaît pour le traitement statistique entre $\{\mathcal{L}_i\}_{i=1}^n$ et $\{{w_H}^+(m_i^2 \mod N)\}_{i=1}^n$, alors $d_{s-2} = 0$,
- si une corrélation apparaît pour le traitement statistique entre $\{\mathcal{L}_i\}_{i=1}^n$ et $\{{w_H}^+(m_i^7 \mod N)\}_{i=1}^n$, alors $d_{s-2} = 1$.

Le reste des bits de $d$ peuvent alors être retrouvé en appliquant le même traitement sachant les valeurs déjà déduites.

## 1.6 Contre-mesures classiques

Suite aux premières publications des attaques par canaux cachés, des solutions ont dû être mises en place pour protéger les cryptosystèmes embarqués sur carte à puce. De nos jours, trois principales catégories de contre-mesures sont couramment utilisées, à savoir les contre-mesures décrivant des méthodes d'implémentations équilibrées, les contre-mesures dégradant les fuites d'information utiles les rendant ainsi plus difficile à exploiter et celles appliquant un masquage sur les variables sensibles. Ces différentes catégories de contre-mesures peuvent être appliquées en parallèle afin de rendre difficile, voire impossible l'ensemble des attaques par canaux cachés décrites précédemment. Dans les sous-sections suivantes, on décrit plus précisément chacune d'entre elles.

### 1.6.1 Implémentation équilibrée

Les méthodes d'implémentations équilibrées sont spécifiques pour protéger les cryptosystèmes à la fois des attaques temporelles (cf. sous-section 1.3) et des attaques simples (cf. sous-section 1.4). Cette contre-mesure assure que chaque exécution s'effectue avec exactement le même comportement quelles que soient les variables manipulées. Elle consiste par exemple à modifier les tests conditionnels en rajoutant des opérations factices [JY02, CMCJ04].

En particulier, dans le cadre de la signature RSA, cette contre-mesure consiste à effectuer les mêmes opérations quelle que soit la valeur du bit de l'exposant $d$. Deux méthodes efficaces pour y parvenir sont les méthodes : *Square & Multiply Always* [CFG+11] et *Montgomery Ladder* [JY02]. La méthode *Square & Multiply Always* (notée dans la suite *S&M Always*) consiste à effectuer une multiplication quelle que soit la valeur du bit $d$ (Alg. 3).

**Algorithme 3** Exponentiation modulaire utilisant la méthode *S&M Always* [CFG+11]

ENTRÉES: Soient $m$ un message, $d = (1, d_{s-2}, \ldots, d_0)_2$ un exposant privé et $N$ un module public
SORTIE: $m^d \mod N$

1. $r_0 \leftarrow m$
2. **Pour** $i = s - 2$ *à* $0$
3. $\quad r_0 \leftarrow r_0^2 \mod N$
4. $\quad r_1 \leftarrow m r_0 \mod N$
5. $\quad r_0 \leftarrow r_{d_i}$
6. **Retourner** $r_0$

La méthode *Montgomery Ladder* consiste quant à elle à utiliser l'exponentiation de Montgomery [Mon87]. Pour obtenir le résultat de l'exponentiation $m^d \mod N$, une multiplication suivie d'une élévation au carré est effectuée à chaque tour de boucle. En particulier, à la fin de l'exécution, le couple $(m^d \mod N, m^{d+1} \mod N)$ est obtenu (Alg. 4).

**Algorithme 4** Exponentiation modulaire utilisant la méthode *Montgomery Ladder* [JY02]

ENTRÉES: Soient $m$ un message, $d = (1, d_{s-2}, \ldots, d_0)_2$ un exposant privé et $N$ un module public
SORTIE: $m^d \mod N$

1. $r_0 \leftarrow m$
2. $r_1 \leftarrow m^2 \mod N$
3. **Pour** $i = s - 2$ *à* $0$
4. $\quad r_{\bar{d}_i} \leftarrow r_0 r_1 \mod N$
5. $\quad r_{d_i} \leftarrow r_{d_i}^2 \mod N$
6. **Retourner** $r_0$

Ces deux méthodes d'exponentiation sont également couramment utilisées pour effectuer des multiplications scalaires dans le cadre des cryptosystèmes basés sur les courbes elliptiques. Dans ce cas, les multiplications et les élévations au carré sont alors remplacées par les opérations d'addition et de doublement définies sur les courbes elliptiques [Cor99, JY02].

### 1.6.2 Dégradation des fuites d'information

Les contre-mesures qui tentent de dégrader les fuites d'information pouvant être exploitées par l'attaquant, s'avèrent pertinentes pour rendre l'ensemble des attaques par canaux cachés plus difficile à mettre en place. Pour ce faire, les principales techniques utilisées sont présentées ci-après.

La **dé-synchronisation** du processus permet d'interférer sur chaque exécution d'un algorithme de sorte que d'une exécution à une autre, les instructions du programme soient déplacées aléatoirement dans le temps. Par conséquent, d'une exécution à une autre, un même événement ne s'effectuera alors pas au même instant $t$, ce qui aura pour effet de ne pas faire apparaître de point de corrélation lors d'une attaque statistique (cf. section 1.5). Ce type de contre-mesure peut par exemple être effectué en insérant du code factice comme des boucles d'attente à des instants aléatoires [TB07, CK09], en ajoutant des cycles d'horloge permettant d'interrompre l'exécution du code à des instants aléatoires ou encore en modifiant aléatoirement la fréquence de l'horloge du composant [CCD00]. Ce type de contre-mesure est assez efficace, notamment contre les attaques statistiques. Cependant, en effectuant une phase très délicate de re-synchronisation sur les courbes de consommation mesurées, des événements peuvent être réalignés et ainsi permettre la réussite de telles attaques.

La contre-mesure appelée *shuffling* en anglais, consiste à **cacher l'exécution de l'algorithme à protéger parmi plusieurs exécutions** effectuées avec des données factices [RPD09, VCMKS12, CH12]. L'ordre des exécutions est aléatoire de sorte que l'attaquant ne puisse pas prédire le moment de la « vraie » exécution retournant le résultat final attendu. Par conséquent, les exécutions factices jouent un rôle de bruit ayant pour but de rendre les attaques statistiques plus difficiles à exploiter. Le nombre d'exécutions factices doit être déterminé en fonction de la résistance du composant utilisé, mais aussi en fonction du temps nécessaire à effectuer une exécution. Un compromis doit alors être déterminé pour pouvoir appliquer cette contre-mesure efficacement tout en restant performant en termes de temps d'exécution.

Certaines contre-mesures consistent à **brouiller la consommation électrique** du composant en rendant les fuites d'information floues et imprécises [Sha00]. Par exemple, un circuit filtrant peut être mis en place lors de la fabrication du composant. Il s'agit de condensateurs placés sur des lignes d'alimentation de la carte à puce ayant pour effet de lisser les variations de consommation de courant du composant. Un second exemple courant permettant également de brouiller la consommation électrique du composant, est l'ajout de bruit qui s'effectue en activant diverses sources de consommation du composant comme le crypto-processeur. Il est à noter que ce type de contre-mesure peut parfois être contourné en moyennant les courbes de consommation mesurées ou en observant le rayonnement électromagnétique au lieu de la consommation électrique [GMO01, AARR02].

La **technologie duale**, appelée *precharged dual-rail logic* en anglais, rend la consommation la plus indépendante possible des données manipulées par une conception équilibrée au niveau des portes logiques [SMBY04, PM05, BGLT06]. Cette contre-mesure consiste à doubler tous les bus du composant de sorte que lorsque l'un prend une valeur, le second prend sa valeur complémentée. Par conséquent, si la consommation produite par

chaque paire de bus est parfaitement équilibrée, on s'attend à ce que la consommation soit constante. Cependant, en pratique, des petites variations de consommation peuvent être observées [CZ06]. Notamment, en observant les fuites émises par rayonnement électromagnétique, des déséquilibres peuvent être plus ou moins distingués en fonction de la position de la sonde de mesure (cf. remarque 1.4.1).

### 1.6.3 Masquage des données

Comme décrit dans la sous-section 1.5, la réussite d'une attaque statistique repose sur plusieurs hypothèses. En particulier, l'attaquant doit pouvoir prédire les valeurs prises par une variable sensible en distinguant une dépendance entre l'un des ensembles des valeurs possibles et les courbes de consommation mesurées (avec les mêmes notations que précédemment : entre $\{y_{i,k}\}_{i=1}^{n}$ et $\{\mathcal{L}_i\}_{i=1}^{n}$). Par conséquent, une solution pour faire échouer ce type d'attaque, consiste à rendre chacune des valeurs pouvant être prédites par l'attaquant indépendante de la consommation émise lors de l'exécution de l'algorithme. Une solution couramment utilisée pour protéger les cryptosystèmes symétriques et asymétriques est le masquage. Cette contre-mesure consiste principalement à appliquer à chaque variable sensible, une *variable aléatoire*, appelée masque avant l'exécution de l'algorithme. Ainsi tout le long de l'exécution chaque variable sensible $Z$ de l'algorithme est substituée par une variable dite masquée. Assurément, à la fin de l'exécution de l'algorithme sécurisé, les variables masquées doivent être démasquées de sorte que le résultat final retourné soit celui attendu [CJRR99b, GP99].

#### 1.6.3.1 Cas des cryptosystèmes asymétriques

Dans le cas des cryptosystèmes asymétriques, chaque variable sensible est masquée en lui additionnant une variable aléatoire de sorte que la réduction modulaire de cette dernière par le module retourne la variable démasquée. Par exemple, la signature RSA, $S = M^d \mod N$, peut être effectuée de manière masquée en calculant :

$$(M + R_1 N)^{d + R_2 \varphi(N)} \mod R_3 N \ , \tag{1.22}$$

où $R_1, R_2$ et $R_3$ sont des variables aléatoires.
Pour retourner le résultat final attendu, il suffit de réduire la signature masquée modulo $N$ [MDS99b, CM04, AFV07]. Masquer à la fois le message et l'exposant privé permet de prémunir la signature des attaques statistiques, notamment de celle présentée dans la sous-section 1.5.2.2. De plus, le masquage permet de résister à certaines attaques SPA [Nov02, OT04].

#### 1.6.3.2 Cas des cryptosystèmes symétriques

Dans le cas des cryptosystèmes symétriques, le masquage dit *booléen* est le principal appliqué. Ce masquage consiste principalement à additionner initialement chaque secret par une variable aléatoire de même taille. Par exemple, dans le cas du chiffrement AES, chaque octet $k$ de clé de tour, est remplacé par

$$k+R \ , \tag{1.23}$$

où $R$ est une variable uniformément distribuée sur $\mathbb{F}_{256}$.
Par construction, les valeurs de la variable masquée $(k+R)$ sont alors indépendantes d'une exécution à une autre. Ainsi durant toute l'exécution de l'algorithme, en manipulant la variable masquée $(k + R)$ (au lieu de $k$) et en propageant le masque indépendamment, aucune valeur intermédiaire ne peut être prédite par l'attaquant. Ce type de masquage est particulièrement bien adapté lors de l'exécution de transformations linéaires. En effet, dans ce cas, la propagation du masque consiste simplement à le mettre à jour en lui appliquant les transformations. Cependant, lorsqu'une transformation non linéaire est requise, notamment la transformation *SubBytes*, la propagation du masque est plus délicate.

Par exemple, présentons la solution décrite dans [PR08] que l'on considérera ensuite dans le chapitre 2. Cette solution consiste à utiliser une table pré-calculée de sorte que la nouvelle transformation *SubBytes* sur une variable masquée $X + R$, retourne le résultat $\mathsf{SubBytes}(X) + S$, avec $S$ un nouveau masque uniformément distribué sur $\mathbb{F}_{256}$. L'implémentation de cette solution est donnée par l'algorithme 5.

**Algorithme 5** Implémentation de la transformation *SubBytes* masquée [PR08]

ENTRÉES: Soient $\tilde{x} = x + r$ une valeur masquée, $r$ un masque d'entrée, $s$ un masque de sortie, $F[.]$ une table pré-calculée telle que $F[x] = \mathsf{SubBytes}(x)$, pour tout $x \in \mathbb{F}_{256}$
Soient $b$ un bit généré aléatoirement et $compare(x, y)$ une fonction de comparaison sécurisée retournant $b$ si $x = y$ et $\bar{b}$ sinon. Notons $\mathrm{R}_0$ et $\mathrm{R}_1$ deux registres distincts
SORTIE: $\mathsf{SubBytes}(x) + s$

1. **Pour** $a = 0$ *à* $255$
2. $\quad cmp \leftarrow compare(a, r)$
3. $\quad \mathrm{R}_{cmp} \leftarrow F[\tilde{x} + a] + s$
4. **Retourner** $\mathrm{R}_b$

Pour déterminer le résultat masqué (*i.e.,* $\mathsf{SubBytes}(x) + s$), l'idée de l'algorithme 5 est de lire successivement toutes les valeurs de la table $F$. Pour ce faire, le paramètre $a$ est incrémenté de 0 à 255. Ainsi toutes les valeurs de la table $F$ vont être lues en parcourant cette table de $\tilde{x}$ à $\tilde{x} + 255 \mod 256$. En particulier, à chaque tour de boucle, la fonction *compare* permet de terminer si $a = t$ ou non, en retournant une valeur de bit $b$ ou $\bar{b}$, où

$b$ est un bit généré aléatoirement au début de l'algorithme. Lorsque $a = t$, alors $cmp = b$ et ainsi la valeur :

$$F[\tilde{x} + a] + s = F[x + r + a] + s = F[x] + s = \mathsf{SubBytes}(x) + s \tag{1.24}$$

est stockée dans le registre $\mathrm{R}_b$. Dans les 255 autres cas (*i.e.,* $a \neq r$), les valeurs $F[\tilde{x}+a]+s$ sont stockés dans le second registre $\mathrm{R}_{\bar{b}}$. En supposant la fonction de comparaison implantée de manière sécurisée, cette procédure est résistante aux attaques statistiques car un attaquant ne peut pas prévoir à quel moment la valeur $\mathsf{SubBytes}(x)$ est manipulée. En effet $r$ étant indépendant d'une exécution à une autre, le paramètre $a$ vérifiant $a = r$, va donc également être indépendant d'une exécution à une autre. De plus comme $b$ est un bit généré uniformément dans $\{0, 1\}$, les rôles des registres $\mathrm{R}_0$ et $\mathrm{R}_1$ ne dépendent pas non plus de la valeur correcte $\mathsf{SubBytes}(x)$.

En pratique, l'utilisation de masquage des variables sensibles par une variable aléatoire est particulièrement efficace pour sécuriser les algorithmes cryptographiques contre les attaques statistiques. Cependant cette contre-mesure est vulnérable à un type d'attaque par canaux cachés que l'on a omis de mentionner dans ce chapitre, à savoir les attaques d'ordre supérieur. Ce type d'attaque consiste à analyser plusieurs points de mesure dans le but de tenter d'« annuler » l'effet du masquage. Plus précisément, ce type d'attaque est présenté dans le chapitre suivant, où l'on étudie l'une des principales contre-mesures pour y remédier : l'utilisation de schéma de partage de secret qui est une généralisation du masquage booléen.

# Chapitre 2

# Attaques statistiques d'ordre supérieur et schémas de partage de secret

## Sommaire



## 2.1 Introduction

Suite à l'efficacité du masquage pour prémunir les cryptosystèmes embarqués des attaques statistiques [CJRR99b, GP99] (cf. sous-section 1.6.3), les attaquants se sont intéressés aux attaques **d'ordre supérieur** initialement introduites dans [KJJ98, KJJ99]. Ce type d'attaque, présenté plus formellement par Thomas S. Messerges dans [Mes00a,

Mes00c], consiste à combiner l'observation de $t$ points de fuite d'une même courbe de consommation [1] mesurée. Le but de ce type d'attaque est de retrouver une petite partie de clé secrète (un bit ou un octet) en cherchant à « annuler » l'effet du masquage pour ensuite appliquer une attaque statistique classique. Pour illustrer l'efficacité de telles attaques, Thomas S. Messerges a présenté dans [Mes00c] une attaque d'ordre 2 ciblant une implantation résistante contre les attaques statistiques classiques qu'il a ensuite mené en pratique. Suite à cette première attaque, plusieurs attaques ont été présentées ciblant des cryptosystèmes résistants aux attaques statistiques classiques [WW04, JPS05, OMHT06]. On peut noter que les chiffrements par blocs sont particulièrement touchés par les attaques statistiques d'ordre supérieur contrairement aux cryptosystèmes asymétriques où elles ne semblent pas s'appliquer.

L'élaboration de contre-mesures pour protéger les chiffrements par blocs contre les attaques d'ordre 2 et plus généralement contre les attaques d'ordre $t$ est très vite devenue primordiale. Lorsqu'un attaquant est capable d'observer la consommation de $t$ données intermédiaires, l'adaptation du masquage à l'ordre $t+1$ qui consiste à partager chaque variable sensible en $t+1$ parties, s'avère alors une solution naturelle [CJRR99b, GP99]. De nombreuses solutions ont été proposées, comme par exemple [Mes00b, OMP04, BMK04, OS06, SP06, PR08, RDP08, RP10, PR11a, GPQ11b]. Cependant, protéger l'implantation d'un cryptosystème contre ce type d'attaque est un exercice délicat. Les premières solutions proposées furent pour la plupart cassées dans les mois qui suivirent leur publication. Par exemple, dans le cas du DES, la méthode du masque unique, proposée en 2003 par Mehdi-Laurent Akkar et Louis Goubin dans [AG03] fut cassée un an plus tard par Mehdi-Laurent Akkar, Régis Bévan et Louis Goubin [ABG04]. De plus, l'amélioration qu'ils proposèrent dans cet article fut également cassée peu de temps plus tard [LH05, HP06]. De même pour le cas de l'AES, la première méthode de protection pour parer les attaques d'ordre supérieur proposée par Kai Schramm et Christof Paar en 2006 [SP06] fut cassée l'année suivante par Jean-Sébastien Coron, Emmanuel Prouff et Matthieu Rivain dans [CPR07]. En particulier, ils montrèrent que leur contre-mesure dite résistante à l'ordre $t$, est vulnérable aux attaques d'ordre 3, voire à l'ordre 2 en fonction de l'implantation de la transformation $MixColumns$.

L'étude de contre-mesures résistantes aux attaques d'ordre $t$ est toujours un sujet de recherche qui suscite un grand intérêt dans le domaine académique mais aussi dans le domaine industriel [RP10, FMPR10, GPQ11b, BFGV12, CPRR13, GSF13]. Assurément, une des principales raisons est due au besoin des industriels de cartes à puce de mettre en place des contre-mesures adaptées [2] afin de sécuriser leur produit. Aujourd'hui, la principale solution étudiée pour parvenir à cet objectif est basée sur l'utilisation de schéma de

1. Comme dans le chapitre 1, la consommation de courant peut être remplacée par l'analyse du rayonnement électromagnétique.

2. En termes de temps d'exécution et de consommation mémoire.

partage de secret qui est une généralisation du masquage booléen. Les schémas de partage de secret sont principalement utilisés dans le contexte du calcul multi-parties, où certaines problématiques peuvent se rapprocher du contexte des attaques d'ordre supérieur. Il est alors naturel de chercher à adapter les solutions étudiées dans le contexte du calcul multi-parties, pour proposer des contre-mesures résistantes aux attaques d'ordre $t$. C'est par exemple, le cas des solutions [RP10, GM11, BFGV12] proposées pour l'AES.

Dans ce chapitre, après avoir présenté plus en détail le fonctionnement des attaques d'ordre supérieur, on présente les notions générales liées aux schémas de partage de secret. Ensuite on explicite l'utilisation de ces schémas pour élaborer des contre-mesures résistantes aux attaques d'ordre supérieur. Enfin dans la section 2.5, on souligne l'importance du modèle de fuite pour implanter des contre-mesures en pratique.

## 2.2 Attaque statistique d'ordre supérieur

Contrairement à une attaque statistique classique, décrite dans la section 1.5, qui analyse l'information d'un seul point de fuite parmi plusieurs courbes de consommation, une attaque d'ordre supérieur analyse l'information de plusieurs points de fuite distincts. Ce type d'attaque cherche à mettre en évidence une corrélation entre des points de fuite combinés entre eux et une petite partie du secret manipulé (typiquement un bit ou un octet). Thomas S. Messerges dans [Mes00a, Mes00c] formalise la définition suivante :

**Définition 2.2.1** (Attaque d'ordre $t$)**.** *Une attaque d'ordre $t$ utilise $t$ points de fuite de la consommation mesurée qui correspondent à $t$ valeurs intermédiaires différentes manipulées durant l'exécution de algorithme attaqué.*

D'après cette définition, on peut remarquer qu'une attaque d'ordre 1 est une attaque statistique classique. En opposition, une attaque d'ordre $t > 1$ est appelée HO-DPA (resp. HO-CPA) pour *High-Order Differential Analysis* (resp. pour *High-Order Correlation Analysis*) lorsque la différence des moyennes (resp. le coefficient de corrélation de Pearson) est utilisée comme outil statistique.

Explicitons le principe général d'une attaque d'ordre $t$ sur un chiffrement par blocs, défini sur $\mathbb{F}_2^l$, ciblant une petite partie de la clé secrète (fixe), notée $k^*$. Pour ce faire, considérons, une variable sensible $X = f(M, k^*)$ définie sur $\mathbb{F}_2^l$, avec $M$ une variable uniformément distribuée sur l'ensemble des messages. De plus, supposons que $X$ est remplacée par $t$ variables intermédiaires $X_1, \ldots, X_t$ satisfaisant $X = X_1 + \ldots + X_t$.
Pour retrouver $k^*$ en menant une attaque d'ordre $t$, un attaquant exécute $n$ chiffrements avec des $m_i \in M$ indépendants, pour $i = 1, \ldots, n$ et mesure en parallèle les courbes de consommation associées $\mathcal{L}_i$. Supposons qu'il soit capable de déterminer sur chacune des

courbes $\mathcal{L}_i$, pour $i = 1, \ldots, n$, $t$ points de fuite particuliers :

$$\mathcal{L}_i(t_1), \ldots, \mathcal{L}_i(t_t) \ , \tag{2.1}$$

où $\mathcal{L}_i(t_j)$ correspondent à la manipulation de la valeur prise par $X_j$ à l'exécution $i$, pour $i = 1, \ldots, n$ et $j = 1, \ldots, t$. L'attaquant suppose qu'en combinant, les points $\mathcal{L}_i(t_1), \ldots, \mathcal{L}_i(t_t)$ entre eux, pour $i = 1, \ldots, n$, il pourra obtenir de l'information dépendant de la variable sensible $X$. Il applique alors une **fonction de combinaison** sur les points de fuite de sorte à obtenir $n$ nouveaux points dits combinés, *i.e.,* il calcule pour tout $i = 1, \ldots, n$ :

$$\tilde{\mathcal{L}}_i \leftarrow \mathsf{Comb}\left(\mathcal{L}_i(t_1), \ldots, \mathcal{L}_i(t_t)\right) \ . \tag{2.2}$$

Finalement, l'attaquant applique une attaque statistique (cf. section 1.5) à partir de l'ensemble des points combinés $\{\tilde{\mathcal{L}}_i\}_{i=1}^{n}$ au lieu de l'ensemble des courbes de consommation mesurées afin de déterminer la bonne hypothèse de sous-clé $k^*$.

Initialement, deux fonctions de combinaison ont été proposées pour mettre en place une attaque d'ordre 2 :

– La différence absolue [Mes00c] qui consiste à calculer, pour $i = 1, \ldots, n$ :

$$\tilde{\mathcal{L}}_i = \left|\mathcal{L}_i(t_1) - \mathcal{L}_i(t_2)\right| \ . \tag{2.3}$$

Il est à noter que la combinaison de $t$ points par cette méthode retourne les points combinés suivants, pour $i = 1, \ldots, n$ :

$$\left|\ldots\left|\left|\mathcal{L}_i(t_1) - \mathcal{L}_i(t_2)\right| - \mathcal{L}_i(t_3)\right|\ldots - \mathcal{L}_i(t_t)\right| \ . \tag{2.4}$$

– La multiplication [CJRR99a, SP06] qui consiste à multiplier les points entre eux, pour $i = 1, \ldots, n$ :

$$\tilde{\mathcal{L}}_i = \mathcal{L}_i(t_1) \times \mathcal{L}_i(t_2) \ , \tag{2.5}$$

qui se généralise avec $t$ points de fuite par : $\mathcal{L}_i(t_1) \times \ldots \times \mathcal{L}_i(t_t)$.

Explicitons par exemple, la pertinence de la différence absolue présentée par Thomas S. Messerges dans [Mes00c] ciblant une HO-DPA d'ordre 2. Pour ce faire, considérons un attaquant identifiant une variable sensible $X = M + k^* \in \mathbb{F}_2^l$, avec $M$ et $k^* \in \mathbb{F}_2^l$ et supposons que dans l'implantation attaquée, cette variable est remplacée par deux variables $X + R$ et $R$, avec $R$ une variable uniformément distribuée sur $\mathbb{F}_2^l$.
Considérons le modèle de fuite du composant lié au poids de Hamming de moyenne nulle (cf. définition 1.5.5). Comme l'outil statistique utilisé est la différence des moyennes, le bruit peut alors être ignoré afin de simplifier les expressions. Dans ce cas, supposons l'attaquant capable de récupérer les points de fuite $\mathcal{L}_i(t_1)$ et $\mathcal{L}_i(t_2)$ obtenus au chiffrement $m_i$, pour $i = 1, \ldots, n$, modélisés respectivement par les variables :

$$\begin{array}{rcl} Y_1 & = & a \times \mathrm{w_H}(X + R) \\ Y_2 & = & a \times \mathrm{w_H}(R) \end{array} \tag{2.6}$$

Notons respectivement $k^*[j], M[j]$ et $R[j]$ le $j$-ème bit de $k^*$, $M$ et $R$, avec $j \in \{0, \ldots l-1\}$. Pour retrouver le bit $k^*[j]$ à l'aide d'une DPA, l'attaquant applique la différence absolue entre chaque paire de points de fuite $\mathcal{L}_i(t_1)$ et $\mathcal{L}_i(t_2)$, qui se modélise par la variable associée suivante :

$$Y = |a| \; |\mathrm{w_H}(X+R) - \mathrm{w_H}(R)| \; . \tag{2.7}$$

Ensuite, il détermine les groupes suivants, pour $k[j] = 0$ ou 1 :

$$\begin{array}{rcl} G_{0,k[j]} & = & \{\tilde{\mathcal{L}}_i \; | m_i[j] + k[j] = 0 : i = 1, \ldots, n\} \\ G_{1,k[j]} & = & \{\tilde{\mathcal{L}}_i \; | m_i[j] + k[j] = 1 : i = 1, \ldots, n\} \end{array} \tag{2.8}$$

On peut noter que l'on a par construction : $G_{0,0} = G_{1,1}$ et $G_{0,1} = G_{1,0}$. De ce fait, l'attaquant applique la différence des moyennes seulement entre les groupes $G_{0,0}$ et $G_{0,1}$ (ou seulement entre $G_{1,0}$ et $G_{1,1}$). À ce stade, l'attaquant peut en déduire la valeur de $k^*[j]$ car la différence entre l'espérance de $Y$ sachant $M[j] = 0$ et celle de $Y$ sachant $M[j] = 1$ dépend de la valeur de ce bit. En effet, on peut remarquer dans un premier temps que les valeurs prises par $\mathrm{w_H}(R)$ dépendent des valeurs prises par le bit $R[j]$, et celles prises par $\mathrm{w_H}(X+R)$ dépendent des valeurs prises par $k^*[j], M[j]$ et $R[j]$, on a alors :

$$\begin{array}{rcl} E[\mathrm{w_H}(R) | R[j] = 1] & = & (l+1)/2 \\ E[\mathrm{w_H}(R) | R[j] = 0] & = & (l-1)/2 \\ E[\mathrm{w_H}(X+R) | k^*[j] + M[j] + R[j] = 1] & = & (l+1)/2 \\ E[\mathrm{w_H}(X+R) | k^*[j] + M[j] + R[j] = 0] & = & (l-1)/2 \end{array} \tag{2.9}$$

Par définition de l'espérance conditionnelle, on a :

$$E[Y \mid M[j] = k^*[j] = 0] \;\; = \;\; \textstyle\sum_y y \, \mathbb{P}(Y = y \mid M[j] = k^*[j] = 0) \;\; . \tag{2.10}$$

De plus, la probabilité conditionnelle de la précédente relation se réécrit en fonction de la valeur de $R[j]$ comme :

$$\begin{array}{rcl} \mathbb{P}(Y = y \mid & M[j] = k^*[j] = 0) & \\ = & \mathbb{P}(Y = y \mid M[j] = k^*[j] = 0, R[j] = 0) \times \mathbb{P}(R[j] = 0 \mid M[j] = k^*[j] = 0) & \\ + & \mathbb{P}(Y = y \mid M[j] = k^*[j] = 0, R[j] = 1) \times \mathbb{P}(R[j] = 1 \mid M[j] = k^*[j] = 0) & \end{array} \tag{2.11}$$

Or comme la variable $R$ est indépendante de $M$ et $k^*$ et uniformément distribuée sur $\mathbb{F}_2^l$, on en déduit d'après la loi des probabilités totales :

$$\begin{array}{rcl} \mathbb{P}(Y = y \mid M[j] = k^*[j] = 0) = & \mathbb{P}(Y = y \mid M[j] = k^*[j] = 0, R[j] = 0) \times \mathbb{P}(R[j] = 0) & \\ + & \mathbb{P}(Y = y \mid M[j] = k^*[j] = 0, R[j] = 1) \times \mathbb{P}(R[j] = 1) & \\ = & \frac{1}{2}\mathbb{P}(Y = y \mid M[j] = k^*[j] = 0, R[j] = 0) & \\ + & \frac{1}{2}\mathbb{P}(Y = y \mid M[j] = k^*[j] = 0, R[j] = 1) & \end{array} \tag{2.12}$$

Donc l'espérance de $Y$ sachant $M[j] = 0$, lorsque $k^*[j] = 0$, vaut :

$$\begin{array}{rcl} E[Y \mid M[j] = k^*[j] = 0] & = & \frac{1}{2}\sum_y y\mathbb{P}\left[Y = y \mid M[j] = k^*[j] = R[j] = 0\right] \\ & + & \frac{1}{2}\sum_y y\mathbb{P}\left[Y = y \mid M[j] = k^*[j] = 0, R[j] = 1\right] \\ & = & \frac{1}{2}E[Y \mid M[j] = R[j] = k^*[j] = 0] \\ & + & \frac{1}{2}E[Y \mid M[j] = k^*[j] = 0, R[j] = 1] \end{array} \tag{2.13}$$

De plus, par la linéarité de l'espérance conditionnelle et d'après les relations de (2.9), on obtient :

$$\begin{array}{l} E[a(\mathrm{w_H}(X+R) - \mathrm{w_H}(R)) \mid M[j] = k^*[j] = R[j] = 0] \\ = a\ (\ E[\mathrm{w_H}(X+R) \mid M[j] = k^*[j] = R[j] = 0] \\ -E[\mathrm{w_H}(R) \mid M[j] = k^*[j] = R[j] = 0]\ ) \\ = a\ ((l-1)/2 - (l-1)/2\ ) \\ = 0 \end{array} \tag{2.14}$$

et

$$\begin{array}{l} E[a(\mathrm{w_H}(X+R) - \mathrm{w_H}(R)) \mid M[j] = k^*[j] = 0, R[j] = 1] \\ = a\ (\ E[\mathrm{w_H}(X+R) \mid M[j] = k^*[j] = 0, R[j] = 1] \\ -E[\mathrm{w_H}(R) \mid M[j] = k^*[j] = 0, R[j] = 1]\ ) \\ = a\ ((l+1)/2 - (l+1)/2\ ) \\ = 0 \end{array} \tag{2.15}$$

D'où $E[a(\mathrm{w_H}(X+R) - \mathrm{w_H}(R)) \mid M[j] = k^*[j] = 0] = 0$, et donc on a aussi $E[Y \mid M[j] = k^*[j] = 0] = 0$. Par un raisonnement similaire, on peut montrer que $E[Y \mid M[j] = 1, k^*[j] = 0] = |a|$. Ainsi, la différence entre ces deux espérances, lorsque $k^*[j] = 0$, vaut $-|a|$. De la même façon, lorsque $k_i^* = 1$, on obtient que la différence entre $E[Y \mid M[j] = 0, k^*[j] = 1]$ et $E[Y \mid M[j] = k^*[j] = 1]$ vaut $|a|$. Par conséquent, un attaquant effectuant une DPA sur des points combinés modélisés par la relation (2.7), peut retrouver la valeur du bit $k^*[j]$ car

- Lorsque la différence des moyennes entre les groupes $G_{0,0}$ et $G_{0,1}$ se rapproche de $-|a|$ (*i.e.,* est négative), alors il peut en déduire que $k^*[j] = 0$.
- En opposition, lorsque la différence des moyennes $G_{0,0}$ et $G_{0,1}$ se rapproche de $|a|$ (*i.e.,* est positive), alors il peut en déduire que $k^*[j] = 1$.

Plusieurs études ciblant une HO-DPA et une HO-CPA d'ordre 2, ont explicité la pertinence des fonctions de combinaison basée sur la différence absolue ou sur la multiplication de points de fuite lorsque le modèle de fuite du composant correspond au modèle lié au poids de Hamming [Mes00c, JPS05, SP06]. En fonction du modèle de fuite du composant, l'une ou l'autre méthode est plus efficace. Par exemple, dans [OMHT06], la fonction de combinaison considérant la différence absolue est soulignée comme étant plus efficace que la seconde lorsque le modèle de fuite est lié au poids de Hamming sans bruit. À l'inverse dans [PRB09], lorsque le modèle de fuite est lié au poids de Hamming, la fonction combinaison appliquant la multiplication de points est plus pertinente. D'autres

fonctions de combinaison ont été proposées dans la littérature comme par exemple dans [JPS05, OMHT06, OM07, PRB09]. En particulier, dans [PRB09], les auteurs proposent une amélioration de des deux fonctions de combinaison initiales qui consiste à normaliser les points de fuite :

– La différence absolue normalisée entre deux variables $L(t_1)$ et $L(t_2)$ retourne

$$| \ (L(t_1) - E[L(t_1)]) \ - \ (L(t_2) - E[L(t_2)]) \ | \ . \tag{2.16}$$

– La multiplication normalisée entre deux variables $L(t_1)$ et $L(t_2)$ retourne

$$(L(t_1) - E[L(t_1)]) \ \times \ (L(t_2) - E[L(t_2)]) \ . \tag{2.17}$$

De plus, il montrent que la multiplication normalisée s'avère être la méthode la plus efficace en comparaison des solutions proposées dans [JPS05, OMHT06, OM07].

La principale difficulté pour mettre en œuvre une attaque d'ordre $t$ est la dé-synchronisation. En effet, pour qu'une telle attaque fonctionne, il faut connaître les instants précis où les données à combiner sont manipulées. Ceci étant très difficile en pratique, la phase de combinaison consiste alors à combiner chaque $t$-uplet des points d'une courbe. De plus, il est prouvé théoriquement dans [CJRR99b] que le nombre de courbes nécessaires pour réussir une attaque d'ordre $t$ augmente de manière exponentielle en fonction de l'ordre $t$. Ce résultat a été d'ailleurs vérifié expérimentalement dans [SP06] pour des attaques d'ordre 1 à 4. Un compromis entre l'ordre $t$ considéré et la longueur des courbes de consommation exploitées, est à déterminer afin de rendre l'attaque réalisable en un temps raisonnable. En vue de la longueur des courbes exploitées (généralement, plusieurs milliers de points), seules les attaques de petit ordre peuvent être menées en pratique (*i.e.,* pour le moment $t \leqslant 2$).

Pour se prémunir d'une attaque d'ordre $t$, la principale solution considérée est le masquage d'ordre supérieur qui consiste à partager chaque variable sensible en plusieurs parts. Dans ce cas, un attaquant observant $t$ données intermédiaires lors de chaque exécution, ne pourra retrouver aucune information sur les données sensibles manipulées. Dans la suite, on va voir que cette méthode coïncide avec la notion de schéma de partage de secret que l'on introduit dans la section suivante.

## 2.3 Généralités sur les schémas de partage de secret

En 1968, un problème introduit par Chung Liu dans [Liu] soulevant le risque de consultation d'informations par des personnes non autorisées a suscité l'intérêt de systèmes d'accès appelés **schémas de partage de secret**. De tels schémas permettent non seulement de réduire le risque de perdre des informations secrètes, mais aussi de les rendre

difficiles à compromettre. Un exemple concret d'application est celui de la résolution du problème d'ouverture d'un coffre fort dans une banque, requérant la participation de plusieurs employés. Dans ce cas, le schéma de partage de secret le plus naturel est le schéma de partage secret basique, où la somme de toutes les parts entre elles permet de reconstituer le secret. Suite à cette problématique, d'autres schémas de partage de secret ont été proposés, notamment celui de Shamir dans [Sha79] et celui de Blakley dans [Bla79] introduit indépendamment la même année.

Dans cette section, on introduit dans un premier temps les principales notions liées aux schémas de partage de secret que l'on illustre ensuite par des exemples. Enfin on présente leur principal domaine d'utilisation, à savoir le calcul multi-parties.

### 2.3.1 Définitions

Dans un schéma de partage de secret, un secret est réparti entre plusieurs participants organisés en une structure d'accès recensant tous les groupes pouvant accéder au secret. L'objectif est de fournir une information propre à chaque participant de sorte que seul un groupe dit **qualifié** de participants puisse reconstruire le secret. Pour ce faire, un schéma de partage de secret entre $n$ participants est défini à partir de deux procédures appelées **procédure de partage** et **procédure de reconstruction**. Plus précisément, on donne la définition suivante :

**Définition 2.3.1** (Schéma de partage de secret)**.** *Soient $S, S_1, \ldots, S_n$ des ensembles non vides. Un schéma de partage de secret entre $n$ participants est défini par :*

- *Une **procédure de partage** probabiliste, notée* $\mathsf{partage}(.)$*, satisfaisant :*

$$\forall s \in S, \mathsf{partage}(s) \rightarrow v_s \in E_s \ , \tag{2.18}$$

  *où $v_s$, appelé **vecteur de partage**, est constitué des $n$ parts : $c_i \in S_i$ dont chacune est supposée détenue par un participant $i$, pour $i \in \{1, \ldots, n\}$. De plus, l'ensemble $E_s \subseteq S_1 \times \ldots \times S_n$ désigne l'ensemble des vecteurs de partage pouvant être obtenu en appliquant la procédure de partage à un secret $s \in S$.*

- *Une **procédure de reconstruction** déterministe, notée* $\mathsf{reconstruction}(.)$*, vérifiant :*

$$\forall v_s = (c_1, \ldots, c_n) \in E_s, \mathsf{reconstruction}((c_i)_{i \in Q}) \rightarrow s \in S \ . \tag{2.19}$$

  *où $Q$ est un groupe qualifié. En particulier, le nombre minimum de parts quelconques d'un vecteur de partage permettant de reconstituer le secret $s$ est nommé **paramètre de reconstruction** et est noté dans la suite $r$.*

*Un tel schéma de partage de secret est associé à un ensemble $E$ défini tel que :*

$$E = \{(s, v_s) : s \in S, v_s \in E_s \subseteq S_1 \times \ldots \times S_n\} \ . \tag{2.20}$$

**Définition 2.3.2** (Idéal)**.** *On dit qu'un schéma de partage de secret associé à l'ensemble* $E = \{(s, v_s) : s \in S, v_s \in E_s \subseteq S_1 \times \ldots \times S_n\}$ *est idéal si :*

$$S = S_1 = \ldots = S_n \ .$$

**Définition 2.3.3** (Linéaire)**.** *On dit qu'un schéma de partage de secret idéal associé à l'ensemble* $E = \{(s, v_s) : s \in S, v_s \in E_s \subseteq S_1 \times \ldots \times S_n\}$ *est linéaire, si les relations suivantes sont vérifiées sur* $E$ *:*

$$\forall (s, c_1, \ldots, c_n), (s', c'_1, \ldots, c'_n) \in E \ \Rightarrow \ (s + s', c_1 + c'_1, \ldots, c_n + c'_n) \in E \quad (2.21)$$

$$\forall (s, c_1, \ldots, c_n) \in E \ \Rightarrow \ (\gamma s, \gamma c_1, \ldots, \gamma c_n) \in E \ , \quad (2.22)$$

*avec* $\gamma$ *un scalaire.*

**Définition 2.3.4** (Paramètre de sécurité)**.** *Le paramètre de sécurité d'un schéma de partage de secret, noté* $t$*, indique le nombre maximum de parts d'un vecteur de partage pouvant être connu sans qu'aucune information sur le secret partagé ne soit révélée.*

**Définition 2.3.5** (Schéma à seuil)**.** *Considérons un schéma de partage de secret de paramètres de reconstruction* $r$ *et de sécurité* $t$*. On dit qu'un schéma de partage de secret est à seuil lorsque* $r = t + 1$*.*

**Notation 2.3.6.** *Soit* $A$ *un ensemble. Dans la suite, on notera* $a \in_R A$ *pour indiquer que* $a$ *est un élément généré uniformément sur* $A$*.*

## 2.3.2 Quelques exemples de schémas de partage de secret idéal

### 2.3.2.1 Schéma de partage de secret basique

Le schéma de partage de secret basique entre $n$ participants est le plus simple des schémas de partage de secret. Un tel schéma est idéal, *i.e.,* les ensembles des secrets sont chacun défini sur un corps fini $\mathbb{F}_q$ :

$$S = S_1 = \ldots = S_n = \mathbb{F}_q \ . \quad (2.23)$$

La procédure de partage appliquée à un secret $s \in \mathbb{F}_q$ retourne un vecteur de partage, $v_s = (c_1, \ldots, c_n)$, constitué de $n-1$ premières parts uniformément distribuées sur $\mathbb{F}_q$ et d'une $n$-ième obtenue en soustrayant les $n-1$ premières parts à la valeur secrète $s$, *i.e.,* :

$$\begin{cases} c_i \in_R \mathbb{F}_q, \text{ pour tout } i = 1, \ldots, n-1 \\ c_n = s - (c_1 + \ldots + c_{n-1}) \end{cases} \quad (2.24)$$

Par construction, ce schéma de partage de secret admet un unique groupe qualifié, à savoir $Q = \{1, \ldots, n\}$. La procédure de reconstruction appliquée à un vecteur de partage $v_s \in E_s$ est alors définie par :

$$\mathsf{reconstruction}(v_s) = \sum_{i=1}^{n} c_i = s \ . \quad (2.25)$$

Comme ce schéma de partage de secret admet un unique groupe qualifié $Q$, tel que $\text{card}(Q) = n$, on peut en déduire que le paramètre de reconstruction vaut $r = n$.
De plus, il est facile de voir que le paramètre de sécurité $t$ vaut $n-1$. En effet, comme chaque $(n-1)$-uplet de parts est uniformément distribué sur $\mathbb{F}_q^{n-1}$, ils sont donc chacun indépendants de $s$. Par conséquent, on a $r = t+1$, ce schéma est donc un schéma de partage de secret à seuil.

#### 2.3.2.2 Schéma de partage de secret multiplicatif

Le schéma de partage de secret multiplicatif entre $n$ participants est un schéma idéal construit de façon similaire au schéma de partage de secret basique. En particulier, ce schéma offre les mêmes paramètres de sécurité et de reconstruction, *i.e.,* $n = t+1 = r$. La seule différence est qu'il s'applique sur le groupe $(\mathbb{F}_q^*, \times)$ contrairement au schéma de partage de secret basique qui s'applique sur le groupe additif $(\mathbb{F}_q, +)$. La procédure de partage appliquée à un secret $s \in \mathbb{F}_q^*$ retourne alors un vecteur de partage, $v_s = (c_1, \ldots, c_n)$ construit tel que :

$$\begin{cases} c_i \in_R \mathbb{F}_q, \text{ pour tout } i = 1, \ldots, n-1 \\ c_n = s \times \prod_{i=1}^{n-1} c_i \end{cases} \tag{2.26}$$

La procédure de reconstruction appliquée à un vecteur de partage $v_s$ est quant à elle définie par :

$$\mathsf{reconstruction}(v_s) = (c_1 \times \ldots \times c_{n-1})^{-1} \times c_n = \prod_{i=1}^{n-1} c_i^{-1} \times s \times \prod_{i=1}^{n-1} c_i \ . \tag{2.27}$$

#### 2.3.2.3 Schéma de partage de secret de Shamir

Initialement introduit dans [Sha79], le schéma de partage de secret de Shamir est un schéma idéal défini sur $\mathbb{F}_q$ basé sur l'interpolation de polynômes. Ce schéma utilise le fait qu'à partir de $k+1$ couples $(x_i, y_i) \in (\mathbb{F}_q \times \mathbb{F}_q)$, où les $x_i$ pour $i \in \{1, \ldots, k+1\}$ sont tous distincts deux à deux, on peut construire un unique polynôme $f$ de degré $k$ tel que $f(x_i) = y_i$.

La procédure de partage entre $n$ participants appliquée à un secret $s \in \mathbb{F}_q$, consiste dans un premier temps à générer $k$ éléments secrets uniformément distribués sur $\mathbb{F}_q$, avec $k < n$, notés $(a_1, \ldots, a_k)$, afin de construire le polynôme suivant :

$$f(X) = s + \sum_{j=1}^{k} a_j X^j \ . \tag{2.28}$$

La procédure de partage retourne ensuite un vecteur de partage $v_s = (c_1, \ldots, c_n)$ où les parts correspondent à l'évaluation du polynôme $f$ en $n$ points, $x_1, \ldots, x_n \in \mathbb{F}_q$, non nuls et distincts, *i.e.,* :

$$c_i = f(x_i), \text{ pour tout } i = 1, \ldots, n \ . \tag{2.29}$$

Il est à noter que les $n$ points $x_1, \ldots, x_n$ peuvent être publics.

À partir d'un tel schéma, il est possible de reconstruire le secret partagé en utilisant les polynômes d'interpolation de Lagrange. Plus précisément, pour tout groupe qualifié $Q \subseteq \{1, \ldots, n\}$, la procédure de reconstruction appliquée à un vecteur de partage $v_s = (c_1, \ldots, c_n)$ est définie par :

$$\mathsf{reconstruction}((c_i)_{i \in Q}) = \sum_{i \in Q} c_i \beta_i(0) \ , \tag{2.30}$$

où $\beta_i(X) = \prod_{j \in Q, j \neq i} \frac{X - x_j}{x_i - x_j}$ sont les polynômes d'interpolation de Lagrange.

Par définition, un polynôme de degré $k$ peut être construit avec un minimum de $k+1$ points distincts deux à deux. Le paramètre de reconstruction associé au schéma de partage de secret de Shamir vaut donc $r = k + 1$. Le paramètre de sécurité vaut quant à lui $t = k$. En effet, avec $k$ parts, on peut trouver, pour chaque valeur de $s \in S$, un polynôme de degré au plus $k - 1$ qui interpole bien ces $k$ parts. Toutes les valeurs $s \in S$ sont alors possibles avec équiprobabilité. Par conséquent, on a $k + 1 = r = t + 1$, ce schéma est donc un schéma de partage de secret à seuil.

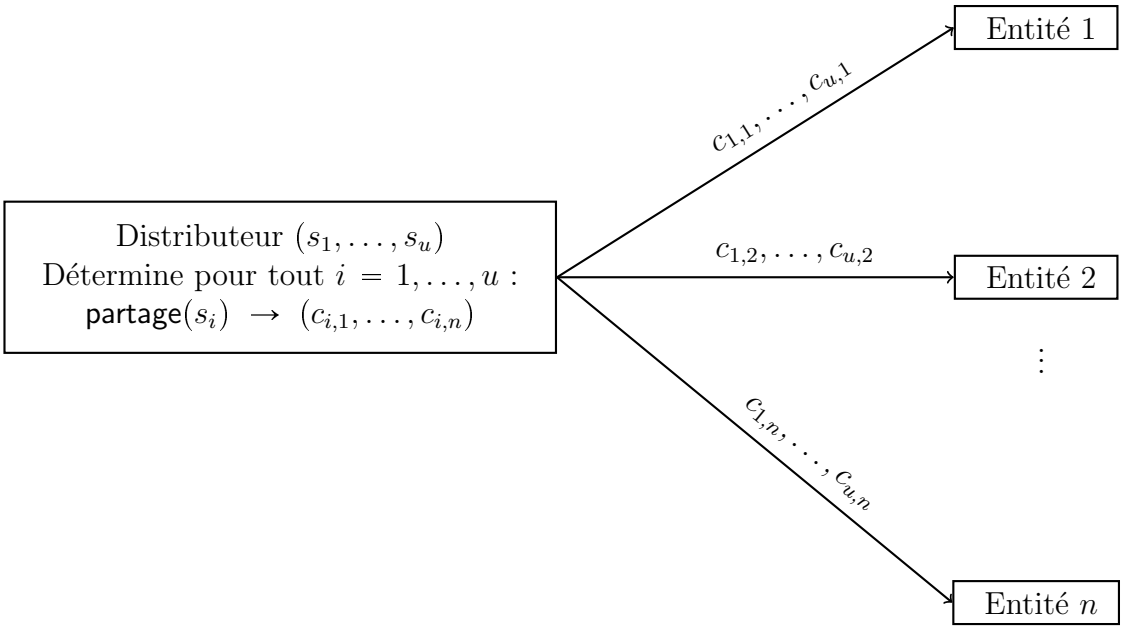
### 2.3.3 Principal domaine d'utilisation : le calcul multi-parties

L'apparition des premiers schémas de partage de secret a conduit au calcul distribué aussi appelé **calcul multi-parties**. Un premier exemple a été introduit par Andrew C. Yao dans [Yao82], puis de nombreux problèmes utilisant des schémas de partage de secret ont été étudiés comme par exemple dans [Yao86, GMW87, BOGW88, CCD88, GRR98, CDM00].

Dans ce contexte, les schémas de partage de secret sont utilisés afin d'exécuter des protocoles partagés entre plusieurs entités, de sorte qu'aucune information sensible ne soit révélée à un attaquant. Les protocoles sont développés en considérant deux types d'attaquants particuliers. Le premier **attaquant dit passif** est supposé capable de récupérer toutes les informations disponibles par plusieurs entités. Le second **attaquant dit actif** contrôle quant à lui le comportement complet de plusieurs entités, par exemple, il peut perturber les opérandes ou les instructions effectuées par les entités corrompues.

Par exemple, décrivons le principe d'un protocole partagé entre $n$ entités et retournant le résultat d'une fonction $f$ prenant en entrée $u$ éléments secrets : $s_1, \ldots, s_u \in S$. Considérons un ensemble $E = \{(s, v_s) : s \in S, v_s \in E_s \subseteq S_1 \times \ldots \times S_n\}$ associé à un schéma de partage de secret de paramètres de reconstruction $r$ et de sécurité $t$. L'exécution d'un protocole partagé requiert une entité supplémentaire appelé distributeur. Son rôle consiste dans un premier temps à appliquer une procédure de partage sur chacune des

entrées $s_1, \ldots, s_u$. Puis une fois les vecteurs de partage associés au données $s_i$ calculés, il distribue à chaque entité $i$ (pour $i \in \{1, \ldots, n\}$), chaque $i^{\text{ème}}$ coordonnée des vecteurs de partage obtenus :




Enfin chaque entité $i$, pour $i = 1, \ldots, n$, effectue des transformations à partir des entrées reçues afin de détenir une part finale $z_i \in S_i$ satisfaisant la relation suivante :

$$\mathsf{reconstruction}((z_i)_{i \in Q}) = f(s_1, \ldots, s_u) \ , \tag{2.31}$$

où $Q$ est un groupe qualifié.

Les transformations effectuées par les entités $i$, pour $i = 1, \ldots, n$ doivent se dérouler de sorte qu'aucune information sur les valeurs sensibles détenues par une entité ne soit révélée aux autres. Ainsi, lorsqu'un schéma de partage de secret de paramètre de sécurité $t$ est appliqué, un attaquant dit passif pourra récupérer toutes les informations disponibles par $t$ entités, mais ne pourra reconstruire aucun des secrets partagés $s_1, \ldots, s_u$.

On peut distinguer deux types de transformations : les transformations dites stables et par opposition celles non stables. Une transformation est dite **stable** lorsque chaque entité effectue localement des opérations avec ses propres parts sans la moindre interaction avec des parts détenues par d'autres entités. Par exemple, si l'on considère un schéma de partage de secret linéaire associé à un ensemble $E = \{(s, v_s) : s \in \mathbb{F}_q, v_s \in E_s \subseteq \mathbb{F}_q^n\}$, la somme et la multiplication par un scalaire $\gamma$ sur des vecteurs de partage sont des transformations stables sur $E$ qui peuvent être calculées localement :

$$\begin{cases} \forall (s, c_1, \ldots, c_n), (s', c'_1, \ldots, c'_n) \in E : (c_1 + c'_1, \ldots, c_n + c'_n) \in E_{s+s'} \\ \forall (s, c_1, \ldots, c_n) \in E : (\gamma c_1, \ldots, \gamma c_n) \in E_{\gamma s} \end{cases} \tag{2.32}$$

Si l'on considère un attaquant passif pouvant connaître toutes les parts détenues par $t$ entités, alors l'exécution d'une transformation stable peut s'effectuer de manière sûre car effectuer des opérations sur des parts connues par l'attaquant n'apporte pas plus d'information sur les secrets partagés.

En opposition, une transformation dite **non stable** nécessite quant à elle des interactions entre les entités. Par exemple, une transformation permettant de retourner la multiplication entre deux secrets $s$, $s'$ à partir de deux vecteurs de partage est non stable lorsque l'on considère un schéma de partage de secret dit linéaire tel que le schéma de partage de secret basique ou celui de Shamir. Ce problème a d'ailleurs été initialement étudié dans [BOGW88, CCD88] pour le schéma de partage de secret de Shamir (cf. sous-section 2.4.2.3).

## 2.4 Contre-mesure basée sur les schémas de partage de secret

### 2.4.1 Principe général

Comme explicité dans la sous-section 1.6.3, l'utilisation du masquage booléen est efficace pour protéger les cryptosystèmes symétriques des attaques statistiques classiques (*i.e.,* d'ordre 1). L'efficacité de cette méthode est basée sur le fait qu'un attaquant observant la manipulation d'une donnée ne pourra retrouver aucune information sensible, si chacune des valeurs sensibles est remplacée par sa valeur masquée et par le masque associé. Par le même raisonnement, cette solution peut être étendue afin de se prémunir des attaques d'ordre supérieur. Pour ce faire, chaque variable sensible $Z$ sera alors remplacée par une valeur masquée et par plusieurs masques de sorte qu'un attaquant observant $t$ données intermédiaires lors de chaque exécution, ne puisse retrouver aucune information sur la variable $Z$. Par exemple, le masquage booléen à l'ordre $t+1$ sur $Z \in \mathbb{F}_q$ remplace cette variable par $t+1$ éléments, $Z_1, \ldots, Z_{t+1} \in \mathbb{F}_q$, définis tels que :

$$\begin{cases} Z_i \in_R \mathbb{F}_q, \text{ pour tout } i = 1, \ldots, t \\ Z_{t+1} = Z - (Z_1 + \ldots + Z_t) \end{cases} \tag{2.33}$$

En comparant cette description avec la procédure de partage du schéma de partage de secret basique en $t+1$ parts, on peut voir que les $t+1$ éléments obtenus par l'application du masquage booléen correspondent à un vecteur de partage associé au schéma de partage de secret basique. Ce schéma étant un schéma à seuil, son paramètre de sécurité est égal à $t$, ce qui correspond à l'ordre de l'attaque contre laquelle on souhaite de protéger. En généralisant ces remarques, on peut en déduire que pour protéger une donnée sensible d'une attaque d'ordre $t$, on peut lui appliquer la procédure de partage fournie par un schéma de partage de secret de paramètre de sécurité égal à $t$. L'utilisation de schéma de

partage de secret est donc une généralisation du masquage booléen.

À présent, explicitons comment protéger l'implantation d'un cryptosystème des attaques d'ordre $t$ à l'aide de l'utilisation d'un schéma de partage de secret. Pour ce faire, considérons un ensemble $E = \{(s, v_s) : s \in S, v_s \in E_s \subseteq S_1 \times \ldots \times S_n\}$ associé à un schéma de partage de secret de paramètre de sécurité $t$. Avant l'exécution de l'algorithme, on suppose que chaque valeur sensible est remplacée par un vecteur de partage obtenu en lui appliquant la procédure de partage associé à $E$. En particulier, on suppose que les clés secrètes et les messages en entrée de l'algorithme sont ainsi partagés.
Ensuite, l'algorithme va devoir se dérouler en considérant ces vecteurs de partage obtenus. L'implantation des différentes transformations sur des vecteurs de partage est alors à déterminer. Pour ce faire, une solution naturelle est d'exploiter les solutions proposées dans le contexte du calcul multi-parties. Dans le contexte des attaques par canaux cachés, l'exécution d'un algorithme s'effectue sur une même entité : un composant de carte à puce, alors que dans le contexte du calcul multi-parties, l'exécution s'effectue sur plusieurs entités distinctes. Or en supposant le modèle de fuite du composant comme lié au poids de Hamming des valeurs manipulées, on peut considérer chacune des valeurs manipulées comme étant détenue par une entité distincte. De ce fait, durant chaque exécution de l'algorithme attaqué, les $t$ points de fuite observés par l'attaquant peuvent être considérés comme chacun détenus par $t$ entités distinctes. Par conséquent, un tel attaquant dans le contexte des attaques par canaux cachés, est une restriction du modèle d'attaquant passif décrit dans le contexte du calcul multi-parties dans le sens où son observation se réduit à $t$ valeurs manipulées vues comme des parts détenues par $t$ entités distinctes. Les solutions pour effectuer des transformations dans le contexte du calcul multi-parties peuvent alors être adaptées dans le contexte des attaques par canaux cachés.

L'implantation d'une transformation stable va alors pouvoir s'effectuer simplement en appliquant une transformation indépendamment sur chacune des coordonnées du vecteur de partage associé. L'implantation d'une transformation non stable consiste quant à elle, à effectuer les calculs comme décrits dans le modèle du calcul multi-parties de sorte à retourner un vecteur de partage final associé au résultat final attendu. Plus précisément, la procédure de reconstruction appliquée sur ce vecteur doit retourner le résultat attendu.

**Remarque 2.4.1.** *Pour exécuter une transformation stable résistante aux attaques d'ordre $t$, le coût supplémentaire imputé par l'utilisation d'un schéma de partage de secret entre $n$ parts est grossièrement $n$ fois plus élevé que son exécution non sécurisée. Par conséquent, pour de telles applications, l'utilisation d'un schéma de partage de secret avec un minimum de parts est à privilégier, comme par exemple l'utilisation de schémas à seuil.*

Il est à noter que l'adaptation des propositions étudiées dans le contexte du calcul multi-parties dans le contexte des attaques par canaux cachés, est une solution couramment exploitée. Par exemple, la procédure de multiplication définie dans [BOGW88, CCD88] décrite pour le schéma de partage de secret de Shamir a été adaptée dans le

contexte des attaques par canaux cachés dans [GM11, PR11b], celle proposée dans [ISW03] utilisant le schéma de partage de secret basique a été adaptée dans [RP10] et celle utilisant un schéma de partage de secret multiplicatif décrite dans [DF12] a été adaptée dans [BFGV12]. En particulier, dans la sous-section suivante, on présente différentes applications de schémas de partage de secret permettant d'implanter le chiffrement d'un AES-128 résistant aux attaques d'ordre $t$. On peut noter que l'application proposée dans [BFGV12] n'est pas présentée car cette solution propose l'utilisation d'un schéma de partage de secret non linéaire rendant l'implantation des transformations stables de l'AES extrêmement coûteuses. De plus une faille sur cette adaptation a été présenté dans [PRR13].

### 2.4.2 Application de schémas de partage de secret : cas de l'AES

Dans cette sous-section, on présente différentes méthodes d'implantations du chiffrement de l'AES-128 résistantes aux attaques d'ordre $t$ utilisant des schémas de partage de secret. Ce dernier est composé de plusieurs transformations définies sur le corps fini $\mathbb{F}_{256} \simeq \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$, à savoir les transformations *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey* et *KeyExpansion* [FIP01].

**Remarque 2.4.2.** *Dans la suite, on dira qu'une procédure est sécurisée lorsqu'elle est résistante aux attaques d'ordre $t$.*

**Remarque 2.4.3.** *Considérons le corps de l'AES de base $\mathcal{B}_{aes} = (\zeta^7, \zeta^6, \zeta^4, \zeta^3, \zeta^2, \zeta, 1)$. Chaque élément $b \in \mathbb{F}_{256}$ sera représenté dans la base $\mathcal{B}_{aes}$ par un polynôme de degré au plus 7 :*

$$b_7\zeta^7 + b_6\zeta^6 + b_5\zeta^5 + b_4\zeta^4 + b_3\zeta^3 + b_2\zeta^2 + b_1\zeta + b_0 \ , \tag{2.34}$$

*avec les $b_i \in \mathbb{F}_2$, pour $i = 0, \ldots, 7$.*
*Dans la suite, on notera chaque élément $b$ par sa représentation binaire $(b_7b_6b_5b_4b_3b_2b_1b_0)_2$ sous forme hexadécimale. Par exemple, la valeur hexadécimale $\{0\text{x}B5\}$ correspond à la représentation binaire $(10110101)_2$ ou encore au polynôme $\zeta^7 + \zeta^5 + \zeta^4 + \zeta^2 + 1$.*

Les transformations *ShiftRows*, *MixColumns* et *AddRoundKey* sont dites linéaires sur $\mathbb{F}_{256}$. L'utilisation d'un schéma de partage de secret linéaire idéal défini sur $\mathbb{F}_{256}$ et de paramètre de sécurité $t$ s'avère donc appropriée pour les implanter comme des transformations stables. Considérons alors un tel schéma associé à un ensemble $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \in E_s \subseteq (\mathbb{F}_{256})^n\}$. Décrivons comment ces trois transformations vont s'effectuer lors de chaque tour $j$ de l'AES, avec $j = 1, \ldots, 10$. Pour ce faire, supposons que chaque octet $k_i$, pour $i = 0, \ldots, 15$, de la $j^{\text{ème}}$ clé de tour est partagé en un vecteur de partage $v_{k_i} \in E_{k_i}$. De même, on suppose qu'à chaque octet $x_i$, pour $i = 0, \ldots, 15$, d'un résultat temporaire est substitué un vecteur de partage noté $v_{x_i} \in E_{x_i}$. Ainsi au lieu de manipuler les octets

$k_i$ et $x_i$ pour $i = 0, \ldots, 15$, on manipulera les vecteurs de partage associés pour effectuer ces transformations. Par conséquent, la transformation *ShiftRows* consiste à ré-ordonner les vecteurs de partage entre eux comme illustré ci-dessous :

$$\begin{array}{|c|c|c|c|} \hline v_{x_0} & v_{x_4} & v_{x_8} & v_{x_{12}} \\ \hline v_{x_1} & v_{x_5} & v_{x_9} & v_{x_{13}} \\ \hline v_{x_2} & v_{x_6} & v_{x_{10}} & v_{x_{14}} \\ \hline v_{x_3} & v_{x_7} & v_{x_{11}} & v_{x_{15}} \\ \hline \end{array} \xrightarrow{\text{ShiftRows}} \begin{array}{|c|c|c|c|} \hline v_{x_0} & v_{x_4} & v_{x_8} & v_{x_{12}} \\ \hline v_{x_5} & v_{x_9} & v_{x_{13}} & v_{x_1} \\ \hline v_{x_{10}} & v_{x_{14}} & v_{x_2} & v_{x_6} \\ \hline v_{x_{15}} & v_{x_3} & v_{x_7} & v_{x_{11}} \\ \hline \end{array} \tag{2.35}$$

La transformation *MixColumns* qui opère colonne par colonne, requiert des additions entre vecteurs partage et des multiplications par les scalaires {0x02} et {0x03} sur les vecteurs de partage. Cette transformation peut donc s'effectuer en considérant les vecteurs de partage au lieu des octets. Les nouvelles colonnes de vecteurs de partage $(v_{x'_{4i}}, v_{x'_{4i+1}}, v_{x'_{4i+2}}, v_{x'_{4i+3}})$ pour $i \in \{0, \ldots, 3\}$ sont donc obtenues en calculant :

$$\begin{cases} v_{x'_{4i}} &= \{0x02\} \cdot v_{x_{4i}} + \{0x03\} \cdot v_{x_{4i+1}} + v_{x_{4i+2}} + v_{x_{4i+3}} \\ v_{x'_{4i+1}} &= v_{x_{4i}} + \{0x02\} \cdot v_{x_{4i+1}} + \{0x03\} \cdot v_{x_{4i+2}} + v_{x_{4i+3}} \\ v_{x'_{4i+2}} &= v_{x_{4i}} + v_{x_{4i+1}} + \{0x02\} \cdot v_{x_{4i+2}} + \{0x03\} \cdot v_{x_{4i+3}} \\ v_{x'_{4i+3}} &= \{0x03\} \cdot v_{x_{4i}} + v_{x_{4i+1}} + v_{x_{4i+2}} + \{0x02\} \cdot v_{x_{4i+3}} \end{cases} \tag{2.36}$$

Enfin la transformation *AddRoundKey* retourne 16 nouveaux vecteurs de partage obtenus en additionnant les vecteurs de partage associés à la clé de tour avec ceux associés au résultat intermédiaire :

$$\begin{array}{|c|c|c|c|} \hline v_{k_0} & v_{k_4} & v_{k_8} & v_{k_{12}} \\ \hline v_{k_1} & v_{k_5} & v_{k_9} & v_{k_{13}} \\ \hline v_{k_2} & v_{k_6} & v_{k_{10}} & v_{k_{14}} \\ \hline v_{k_3} & v_{k_7} & v_{k_{11}} & v_{k_{15}} \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline v_{x_0} & v_{x_4} & v_{x_8} & v_{x_{12}} \\ \hline v_{x_1} & v_{x_5} & v_{x_9} & v_{x_{13}} \\ \hline v_{x_2} & v_{x_6} & v_{x_{10}} & v_{x_{14}} \\ \hline v_{x_3} & v_{x_7} & v_{x_{11}} & v_{x_{15}} \\ \hline \end{array} \xrightarrow{\text{AddRoundKey}} \begin{array}{|c|c|c|c|} \hline v_{x_0} + v_{k_0} & v_{x_4} + v_{k_4} & v_{x_8} + v_{k_8} & v_{x_{12}} + v_{k_{12}} \\ \hline v_{x_1} + v_{k_1} & v_{x_5} + v_{k_5} & v_{x_9} + v_{k_9} & v_{x_{13}} + v_{k_{13}} \\ \hline v_{x_2} + v_{k_2} & v_{x_6} + v_{k_6} & v_{x_{10}} + v_{k_{10}} & v_{x_{14}} + v_{k_{14}} \\ \hline v_{x_3} + v_{k_3} & v_{x_7} + v_{k_7} & v_{x_{11}} + v_{k_{11}} & v_{x_{15}} + v_{k_{15}} \\ \hline \end{array} \tag{2.37}$$

Ces trois transformations étant des transformations stables sur $E$, d'après la remarque 2.4.1, l'utilisation d'un schéma de partage de secret à seuil est un bon choix pour implanter ces transformations de façon sécurisée tout en minimisant le coût imputé par l'ajout de cette contre-mesure. Pour cette raison, les principaux schémas étudiés dans

la littérature pour implanter ces transformations sont le schéma de partage de secret basique [RP10, CPRR13, GPQ11a, GPQ11b], et le schéma de partage de secret de Shamir [GM11, PR11a].

À présent considérons de tels schémas pour implanter les autres transformations de l'AES, *i.e.,* les transformations *KeyExpansion* et *SubBytes*.
La transformation *KeyExpansion* nécessite :

- des ou-exclusif,
- la fonction *SubWord()* qui applique la transformation *SubBytes* à 4 octets,
- la fonction *RotWord()* qui est une rotation cyclique retournant sur 4 octets d'entrée $(a, b, c, d)$, la sortie $(b, c, d, a)$,
- la fonction *Ron*(), qui pour tout $n$ retourne les 4 octets suivants : $(2^n, 0, 0, 0)$.

Une description plus détaillée de la transformation *KeyExpansion* peut être trouvée dans [FIP01]. On peut remarquer que cette transformation requiert des fonctions similaires à celles décrites par les autres transformations. Elle peut donc être implantée de façon sécurisée en appliquant les solutions proposées pour ces transformations. Il nous reste donc à décrire l'implantation sécurisée de la transformation *SubBytes*.

Lorsqu'aucune sécurité n'est requise, cette dernière transformation s'effectue efficacement en appliquant une boite-S indépendamment sur chacun des octets du résultat intermédiaire. Cependant lorsque l'on souhaite effectuer cette transformation de façon sécurisée, cette solution devient coûteuse en terme de consommation mémoire car elle demande de stocker autant de tables pré-calculées que de nombres de parts du schéma de partage de secret associé à $E$. Pour remédier à ce problème, une autre solution consiste à effectuer la transformation *SubBytes* en appliquant successivement une inversion sur $\mathbb{F}_{256}$ et une transformation affine [DR02].
L'inversion sur $\mathbb{F}_{256}$ peut être décrite comme une élévation à la puissance 254, nécessitant des élévations au carré et des multiplications sur $\mathbb{F}_{256}$ :

$$\mathsf{Inverse}[X] = X^{254} \mod 256 \ . \tag{2.38}$$

Pour effectuer la multiplication entre deux éléments $a, b \in \mathbb{F}_{256}$, la première approche pourrait consister à utiliser une table pré-calculée pour obtenir le résultat : $\mathsf{mult}[a][b] = ab$. Cependant, une telle table requiert de stocker 65.536 octets, ce qui est trop coûteux en terme de consommation mémoire. Une méthode plus efficace couramment appliquée consiste à prendre en compte deux tables pré-calculées, $\mathsf{log}[.]$ et $\mathsf{alog}[.]$ contenant chacune 255 octets et satisfaisant :

$$\begin{array}{ll} \mathsf{log}[\alpha^i] & = i \\ \mathsf{alog}[i] & = \alpha^i \end{array} \tag{2.39}$$

pour $1 \leqslant i \leqslant 255$ et $\alpha$ un générateur de $\mathbb{F}_{256}^*$ [DR02].
Si $a$ et $b$ sont des éléments non nuls de $\mathbb{F}_{256}$, alors le résultat de la multiplication entre $a$ et $b$ s'obtient en calculant :

$$ab = \mathsf{alog}[(\mathsf{log}[a] + \mathsf{log}[b]) \pmod{255}] \ . \tag{2.40}$$

Une alternative à cette méthode est de considérer le corps $\mathbb{F}_{256}$ comme une extension quadratique de $\mathbb{F}_{2^4}$ [RBJ+01, SMTM01]. Dans ce cas, les opérations requises lors de l'inversion sont définies sur $\mathbb{F}_{2^4}$ et ainsi les multiplications entre deux éléments peuvent être effectuées à l'aide d'une unique table pré-calculée de 256 éléments, ce qui est plus efficace que l'utilisation des tables log, alog.
Une autre alternative pour implanter la transformation *SubBytes* consiste à représenter les boîtes-S sous forme polynômiale, puis à effectuer les puissances requises à l'aide des classes cyclotomiques [3] [CGP+12a]. La transformation *SubBytes* devient alors :

$$\begin{aligned} SubBytes[X] &= \{0x05\}X^{254} + \{0x09\}X^{253} + \{0xF9\}X^{251} + \{0x25\}X^{247} \\ &+ \{0xF4\}X^{239} + X^{223} + \{0xB5\}X^{195} + \{0x8F\}X^{127} + \{0x63\} , \end{aligned} \tag{2.41}$$

où chacune des puissances requises par cette égalité appartient à une même classe cyclotomique. En particulier, d'après la table 1 donnée dans [CGP+12a], cette solution nécessite 4 procédures de multiplication sur des vecteurs de partage défini sur $\mathbb{F}_{256}$.

Dans les sous-sections suivantes, on présente les principales solutions génériques proposées dans la littérature utilisant des schémas de partage de secret pour implanter la transformation *SubBytes* de façon sécurisée. De plus, le coût de chacune de ces solutions est comparé dans le chapitre 4 (cf. sous-section 4.4).
Dans la suite, on suppose que les transformations linéaires sont effectuées comme décrites précédemment en utilisant un schéma de partage de secret linéaire à seuil requérant seulement $t+1$ parts. Par conséquent, on suppose que la transformation *SubBytes* prend en entrée un résultat intermédiaire constitué de 16 vecteurs de partage de longueur $t+1$ et retourne 16 nouveaux vecteurs de partage :

$$\begin{array}{|c|c|c|c|} \hline v_{x_0} & v_{x_4} & v_{x_8} & v_{x_{12}} \\ \hline v_{x_1} & v_{x_5} & v_{x_9} & v_{x_{13}} \\ \hline v_{x_2} & v_{x_6} & v_{x_{10}} & v_{x_{14}} \\ \hline v_{x_3} & v_{x_7} & v_{x_{11}} & v_{x_{15}} \\ \hline \end{array} \xrightarrow{\text{SubBytes}} \begin{array}{|c|c|c|c|} \hline v_{SubBytes(x_0)} & v_{SubBytes(x_4)} & v_{SubBytes(x_8)} & v_{SubBytes(x_{12})} \\ \hline v_{SubBytes(x_1)} & v_{SubBytes(x_5)} & v_{SubBytes(x_9)} & v_{SubBytes(x_{13})} \\ \hline v_{SubBytes(x_2)} & v_{SubBytes(x_6)} & v_{SubBytes(x_{10})} & v_{SubBytes(x_{14})} \\ \hline v_{SubBytes(x_3)} & v_{SubBytes(x_7)} & v_{SubBytes(x_{11})} & v_{SubBytes(x_{15})} \\ \hline \end{array} \tag{2.42}$$

### 2.4.2.1 Utilisation du schéma de partage de secret basique

Dans cette sous-section, on présente les solutions proposées dans [RP10, CPRR13] utilisant le schéma de partage de secret basique. Considérons l'ensemble $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \in E_s \subseteq (\mathbb{F}_{256})^{t+1}\}$ associé au schéma de partage de secret basique. Le résultat en entrée de la transformation *SubBytes* est alors constitué de 16 vecteurs de partage associés à ce schéma. Pour effectuer cette transformation, on présente d'abord la solution proposée dans [RP10] qui consiste à appliquer indépendamment sur chacun des vecteurs

3. Cette solution peut être appliquée pour toute boîte-S.

de partage une procédure d'inversion, puis à appliquer la transformation affine.

Commençons par présenter l'implantation de la transformation la plus simple : celle de la transformation affine sur un vecteur de partage. Cette transformation peut être appliquée pour tout $X \in \mathbb{F}_{256}$ comme :

$$\mathsf{TransAff}[X] = \varphi_A(X) + b \ , \tag{2.43}$$

où $\varphi_A$ est une fonction linéaire sur $\mathbb{F}_2$ et $b$ est l'octet $\{\mathsf{0x63}\}$.
En déterminant à partir d'un vecteur de partage $v_s = (c_1, \ldots, c_{t+1}) \in E_s$, le vecteur suivant :

$$v_{s'} = (\varphi_A(c_1) + b, \varphi_A(c_2), \ldots, \varphi_A(c_{t+1})) \ , \tag{2.44}$$

la procédure de reconstruction appliquée à $v_{s'}$ retourne dans ce cas :

$$\mathsf{reconstruction}(v_{s'}) = \left(\sum_{i=1}^{t+1} \varphi_A(c_i)\right) + b = \varphi_A\left(\sum_{i=1}^{t+1} c_i\right) + b = \varphi_A(s) + b \ . \tag{2.45}$$

La transformation affine peut alors être appliquée comme une transformation stable. Par conséquent, la transformation *SubBytes* décrite dans [RP10] consiste à appliquer la procédure d'inversion donnée par l'algorithme 7 sur chacun des vecteurs de partage du résultat intermédiaire, puis à appliquer la transformation affine comme décrite par la relation (2.44).

L'inversion d'un élément de $\mathbb{F}_{256}$ effectuée comme une exponentiation à la puissance 254 requiert une succession d'élévations au carré et de multiplications. On peut remarquer que l'application d'élévation au carré sur un vecteur de partage associé au schéma de partage de secret basique défini sur $\mathbb{F}_{256}$ est une transformation stable. En effet, pour tout vecteur de partage $v_s = (c_1, \ldots, c_{t+1}) \in E_s$, $\mathsf{reconstruction}(v_s) = \sum_{i=1}^{t+1} c_i = s$, ce qui implique comme $\mathbb{F}_{256}$ est de caractéristique 2 :

$$s^2 = \left(\sum_{i=1}^{t+1} c_i\right)^2 = \sum_{i=1}^{t+1} c_i^2 \ . \tag{2.46}$$

D'où $(c_1^2, \ldots, c_{t+1}^2) \in E_{s^2}$. La procédure d'élévation au carré sur un vecteur de partage peut donc être effectuée comme une transformation stable qui requiert seulement $t+1$ élévations au carré.
La multiplication entre deux vecteurs de partage n'est pas quant à elle une transformation stable, *i.e.*, pour tout $v_s = (c_1, \ldots, c_{t+1}) \in E_s$ et $v_{s'} = (c'_1, \ldots, c'_{t+1}) \in E_{s'}$, tels que $v_s \neq v_{s'}$ :

$$(c_1 c'_1, \ldots, c_{t+1} c'_{t+1}) \notin E_{ss'} \ . \tag{2.47}$$

De ce fait, une procédure de multiplication sécurisée entre deux vecteurs de partage associés au schéma de partage de secret basique est requise. Une première solution a été proposée par Yuval Ishai, Amit Sahai and David Wagner dans [ISW03] pour des implantations *hardware*. Ils ont montré la manière d'effectuer de façon sécurisée une multiplication entre deux vecteurs de partage définis sur $\mathbb{F}_2$. Ensuite Matthieu Rivain et Emmanuel Prouff ont adapté cette solution en proposant une procédure de multiplication entre vecteurs de partage défini sur $\mathbb{F}_{2^n}$, avec $n \in \mathbb{N}^*$ [RP10]. Cette procédure, décrite par l'algorithme 6, nécessite $(t+1)^2$ multiplications, $2t(t+1)$ additions et $t(t+1)/2$ générations d'éléments uniformément distribués sur $\mathbb{F}_{2^n}$.

**Algorithme 6** Procédure de multiplication [RP10, Algorithme 1]

ENTRÉES: $E = \{(s, v_s) : s \in \mathbb{F}_{2^n}, v_s \in E_s \subseteq (\mathbb{F}_{2^n})^{t+1}\}$ un ensemble associé à un schéma de partage de secret basique de paramètre de sécurité $t$.
$(x_1, \ldots, x_{t+1}) \in E_s$ et $(y_1, \ldots, y_{t+1}) \in E_{s'}$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{ss'}$

**Fonction :** $\mathsf{SecMult}_n\left((x_1, \ldots, x_{t+1}), (y_1, \ldots, y_{t+1})\right)$

1. **Pour** $i = 1$ *à* $t+1$ **faire:**
2. **Pour** $j = i+1$ *à* $t+1$ **faire:**
3. Générer $r_{i,j} \in_R \mathbb{F}_{2^n}$
4. $r_{j,i} \leftarrow (r_{i,j} + x_i y_j) + x_j y_i$
5. **Pour** $i = 1$ *à* $t+1$ **faire:**
6. $z_i \leftarrow x_i y_i$
7. **Pour** $j = i+1$ *à* $t+1$, $j \neq i$ **faire:**
8. $z_i \leftarrow z_i + r_{i,j}$
9. **Retourner** $(z_1, \ldots, z_{t+1})$

La procédure d'inversion sur un vecteur de partage peut être effectuée à partir des procédures d'élévation au carré et de multiplication décrites précédemment. Comme la procédure de multiplication est plus coûteuse que celle d'élévation au carré, les auteurs de [RP10] ont explicité une exponentiation à la puissance 254 minimisant le nombre de multiplications. Leur proposition requiert 4 multiplications et 7 élévations au carré sur $\mathbb{F}_{256}$ :

$$\mathsf{Inverse}[X] = X^{254} = \left[(X^2X)^4(X^2X)\right]^{16}(X^2X)^4X^2 \ , \tag{2.48}$$

qui peut s'effectuer en calculant successivement :

$$X \xrightarrow{C} X^2 \xrightarrow{M} X^3 \xrightarrow{2C} X^{12} \xrightarrow{M} X^{15} \xrightarrow{4C} X^{240} \xrightarrow{M} X^{252} \xrightarrow{M} X^{254} \tag{2.49}$$

où $C$, $M$, $2C$ et $4C$ signifient respectivement carré, multiplication, 2 carrés et 4 carrés.

Pour garantir la sécurité de l'exponentiation à la puissance 254, chaque procédure requise doit être résistante aux attaques d'ordre $t$ et notamment la procédure de multiplication. Cependant, comme explicité dans [RP10], cette procédure de multiplication peut être prouvée sûre seulement si les vecteurs de partage en entrée sont $t$-indépendants (*i.e.,* tous les $t$-uplets du premier vecteur doivent être indépendants de tous les $t$-uplets du second). De ce fait, pour pouvoir l'appliquer entre les vecteurs de partage associés à $X^2$ et $X$ ou à $X^3$ et $(X^3)^4$, une procédure dite de rafraichissement doit alors être mise en place pour rendre chacun des vecteurs $t$-indépendant. Sous cette condition, l'exponentiation à la puissance 254, *i.e.,* l'inversion, peut être effectuée de manière sécurisée comme décrite par l'algorithme 7, où $\mathsf{Refresh}(.)$ réfère à une procédure de rafraîchissement. Pour effectuer cette procédure, la solution proposée dans [RP10] consiste à effectuer une addition entre l'un des vecteurs de partage et un vecteur de partage de 0.

**Algorithme 7** Procédure d'inversion de Rivain et Prouff [RP10, Algorithme 3]

ENTRÉES: $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \in E_s \subseteq (\mathbb{F}_{256})^{t+1}\}$ un ensemble associé à un schéma de partage de secret basique de paramètre de sécurité $t$.
$(x_1, \ldots, x_{t+1}) \in E_s$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{s^{254}}$

1. $(y_1, \ldots, y_{t+1}) \leftarrow (x_1^2, \ldots, x_{t+1}^2)$
2. $\mathsf{Refresh}(y_1, \ldots, y_{t+1})$
3. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}_8\left((y_1, \ldots, y_{t+1}), (x_1, \ldots, x_{t+1})\right)$
4. $(w_1, \ldots, w_{t+1}) \leftarrow (z_1^4, \ldots, z_{t+1}^4)$
5. $\mathsf{Refresh}(w_1, \ldots, w_{t+1})$
6. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}_8\left((z_1, \ldots, z_{t+1}), (w_1, \ldots, w_{t+1})\right)$
7. $(z_1, \ldots, z_{t+1}) \leftarrow (z_1^{16}, \ldots, z_{t+1}^{16})$
8. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}_8\left((z_1, \ldots, z_{t+1}), (w_1, \ldots, w_{t+1})\right)$
9. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}_8\left((z_1, \ldots, z_{t+1}), (y_1, \ldots, y_{t+1})\right)$
10. **Retourner** $(z_1, \ldots, z_{t+1})$

On peut noter que deux alternatives à la procédure d'inversion décrite par l'algorithme 7 ont été proposées dans [KHL11, CGP+12a]. Celle proposée dans [KHL11] est une adaptation de cette méthode et de celle décrite dans [RBJ+01, SMTM01] qui considère le corps $\mathbb{F}_{256}$ comme une extension quadratique de $\mathbb{F}_{16}$. Dans ce cas, la procédure d'inversion requiert 5 applications de la procédure de multiplication. Cependant, comme la multiplication entre deux éléments de $\mathbb{F}_{16}$ peut être effectuée à l'aide d'une table précalculée, leur solution reste moins coûteuse en termes de temps d'exécution comparée à la solution initiale. La seconde alternative, décrite dans [CGP+12a], propose d'implanter la transformation *SubBytes* comme définie par la relation 2.41 à l'aide de la procédure

de multiplication donnée par l'algorithme 6. D'après la table 1 donnée dans [CGP+12a], on peut voir que leur solution nécessite 4 procédures de multiplication sécurisées comme pour la solution [RP10].

Ces différentes méthodes permettant d'effectuer la transformation *SuBbytes* utilisent chacune aux moins une fois la procédure de multiplication sécurisée décrite par l'algorithme 6 précédée d'une procédure de rafraichissement. Cependant, Jean-Sébastien Coron *et al.* ont récemment explicité dans [CPRR13] une faille de sécurité lorsque cette procédure de multiplication est précédée de la fonction de rafraîchissement consistant simplement à effectuer une addition entre le vecteur de partage à rafraîchir et un vecteur de partage de zéro. En effet, ils expliquent que dans ce cas, les vecteurs de partage en entrée de la procédure de multiplication ne sont pas $t$-indépendants.
Par exemple, explicitons cette faille pour $t = 2$. Considérons les vecteurs de partage $v_s = (x_1, x_2, x_3)$ et $v_{s^2} = (y_1, y_2, y_3)$ obtenus à la fin de l'étape 2 de l'algorithme 7. Comme la fonction de rafraichissement consiste à additionner le vecteur avec un vecteur de zéro, $v_{s^2}$ se réécrit :

$$v_{s^2} = (y_1, y_2, y_3) = (x_1^2 + r_1, x_2^2 + r_2, x_3^2 + r_1 + r_2) \ , \tag{2.50}$$

avec $r_1, r_2 \in_R \mathbb{F}_q$.

Supposons l'attaquant capable d'observer les consommations associées aux variables suivantes :

- $HW(S^2 + X_1^2 + X_2^2 + R_1) + B_1$ qui correspond à la manipulation de $X_3^2 + R_1 = S^2 + X_1^2 + X_2^2 + R_1$ à un moment donné lors de l'exécution de la fonction de rafraichissement,

- $HW(X_2 \times (X_1^2 + R_1)) + B_2$ qui correspond à la manipulation d'une donnée intermédiaire (étape 4) de la procédure de multiplication de l'étape 3.

On peut voir que la variable $S^2 + X_1^2 + X_2^2 + R_1$ manipule la variable sensible $S^2$ et les mêmes masques $X_1, X_2, R_1$ que la variable $X_2 \times (X_1^2 + R_1)$. De cette remarque, les auteurs de [CPRR13] ont montré que l'observation de ces variables permet de retrouver de l'information sur $S$. Pour ce faire, ils ont explicité la quantité d'information pouvant être retrouvée sur $S$ comparé à des attaques d'ordre 2 et d'ordre 3 classiques, *i.e.,* des attaques ciblant respectivement les points de fuite associés à $\mathrm{w_H}(S + X_1) + B_1$ et $\mathrm{w_H}(X_1) + B_2$ ; et les points de fuite associés à $\mathrm{w_H}(S + X_1 + X_2) + B_1$, $\mathrm{w_H}(X_1) + B_2$ et $\mathrm{w_H}(X_2) + B_3$.

Pour remédier à cette faille de sécurité, les auteurs de [CPRR13] proposent une solution impliquant une fonction $h : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ satisfaisant pour tout $x \in \mathbb{F}_{2^n}$ :

$$h(x) = xg(x) \ , \tag{2.51}$$

où $g$ est une fonction $\mathbb{F}_2$-linéaire.
Ainsi pour effectuer les procédures de multiplication entre les vecteurs de partage associés

à $X$ et $X^2$ et à $X^3$ et $X^{12}$ requises par l'algorithme 7, ils donnent un algorithme sécurisé de la procédure d'évaluation $h$ appliqué sur un vecteur de partage, où $g$ correspond à une élévation soit à la puissance 2, soit à la puissance 4 (cf. [CPRR13, Algorithme 3]). Par ailleurs, ils proposent d'utiliser des tables pré-calculées pour retourner le résultat de la fonction $h$ appliquée sur un élément de $\mathbb{F}_{256}$. Bien que leur solution demande de générer plus de nombres aléatoires, elle reste plus efficace que la solution précédente consistant à effectuer une procédure de rafraichissement et une procédure de multiplication. Ainsi ils suggèrent de remplacer les étapes 2 à 3, et 5 à 6 de la procédure d'inversion (Alg. 7) par cette solution. La procédure d'inversion ainsi obtenue est donnée par l'algorithme 8, où les procédures $\mathsf{EvalPuissance}_3$ et $\mathsf{EvalPuissance}_5$ référent respectivement à l'algorithme sécurisé de la procédure d'évaluation $h$ appliqué sur un vecteur de partage donnée par l'algorithme 5 dans [CPRR13]), où $g$ correspond à une élévation à la puissance 2, respectivement à la puissance 4. De plus, la table 2.1 indique le coût des procédures requises par cet algorithme. Cependant, on peut noter qu'aucune preuve de sécurité sur l'inversion entière n'est prouvée.

**Algorithme 8** Procédure d'inversion de Coron *et al.* [CPRR13]

ENTRÉES: $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \in E_s \subseteq (\mathbb{F}_{256})^{t+1}\}$ un ensemble associé à un schéma de partage de secret basique de paramètre de sécurité $t$.
$(x_1, \ldots, x_{t+1}) \in E_s$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{s^{254}}$

1. $(y_1, \ldots, y_{t+1}) \leftarrow (x_1^2, \ldots, x_{t+1}^2)$ *(vecteur de partage de $E_{s^2}$)*
2. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{EvalPuissance}_3(x_1, \ldots, x_{t+1})$ *(vecteur de partage de $E_{s^3}$)*
3. $(w_1, \ldots, w_{t+1}) \leftarrow (z_1^4, \ldots, z_{t+1}^4)$ *(vecteur de partage de $E_{s^{12}}$)*
4. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{EvalPuissance}_5(z_1, \ldots, z_{t+1})$ *(vecteur de partage de $E_{s^{15}}$)*
5. $(z_1, \ldots, z_{t+1}) \leftarrow (z_1^{16}, \ldots, z_{t+1}^{16})$ *(vecteur de partage de $E_{s^{240}}$)*
6. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}_8((z_1, \ldots, z_{t+1}), (w_1, \ldots, w_{t+1}))$ *(vecteur de partage de $E_{s^{252}}$)*
7. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}_8((z_1, \ldots, z_{t+1}), (y_1, \ldots, y_{t+1}))$ *(vecteur de partage de $E_{s^{254}}$)*
8. **Retourner** $(z_1, \ldots, z_{t+1})$

#### 2.4.2.2 Utilisation du schéma de partage de secret multiplicatif

Comme dans la sous-section précédente, on suppose que toutes les transformations linéaires de l'AES sont effectuées en utilisant le schéma de partage de secret basique défini sur $\mathbb{F}_{256}$ ; de même pour la transformation affine afin qu'elle puisse s'effectuer comme une transformation stable (cf relation (2.44)). Dans ce cas, les 16 vecteurs de partage en entrée et en sortie de l'inversion requise par la transformation *SubBytes* doivent être associés au

| | aléa | add | mult | table |
|---|---|---|---|---|
| $\mathsf{SecMult}_8$ (Alg. 6) | $(t^2+t)/2$ | $2t^2+2t$ | $t^2+2t+1$ | - |
| $\mathsf{EvalPuissance}_3$ ([CPRR13, Alg. 5]) | $t^2+t$ | $5t^2+4t-1$ | - | $2t^2+3t+1$ |
| $\mathsf{EvalPuissance}_5$ ([CPRR13, Alg. 5]) | $t^2+t$ | $5t^2+4t-1$ | - | $2t^2+3t+1$ |

*Le symbole* ***aléa*** *désigne le nombre d'éléments générés uniformément sur* $\mathbb{F}_{256}$ *;* ***add*** *et* ***mult*** *indiquent respectivement le nombre d'additions et de multiplications entre 2 éléments de* $\mathbb{F}_{256}$ *et* ***table*** *indique le nombre d'accès à une table pré-calculée (soit à la table retournant l'élévation à la puissance 3 d'un élément de* $\mathbb{F}_{256}$ *ou soit à la puissance 5).*

TABLE 2.1 – Coût des procédures résistantes aux attaques d'ordre $t$ requises par l'algorithme 8

schéma de partage de secret basique associé à l'ensemble $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \subseteq \mathbb{F}_{256}^{t+1}\}$.

La différence entre la solution présentée dans cette sous-section et celle décrite dans la sous-section 2.4.2.1 porte sur la procédure d'inversion requise par la transformation *SubBytes*. Cette dernière est décrite ici en appliquant un schéma de partage de secret multiplicatif défini sur $\mathbb{F}_{256}^*$. Pour éviter toute confusion, on note l'ensemble associé à ce schéma $Em = \{(s, v_s) : s \in \mathbb{F}_{256}^*, v_s \subseteq (\mathbb{F}_{256}^*)^{t+1}\}$.
Dans un tel schéma, la procédure de multiplication entre deux vecteurs de partage est une transformation stable :

$$\forall (s, c_1, \ldots, c_{t+1}), (s', c'_1, \ldots, c'_{t+1}) \in Em \Rightarrow (c_1 c'_1, \ldots, c_{t+1} c'_{t+1}) \in Em_{ss'} \ . \tag{2.52}$$

Par conséquent, la procédure d'inversion d'un secret $s \in \mathbb{F}_{256}^*$ sur un vecteur de partage est aussi une transformation stable :

$$\forall (s, c_1, \ldots, c_{t+1}) \in Em \Rightarrow (\mathsf{Inverse}[c_1], \ldots, \mathsf{Inverse}[c_{t+1}]) \in Em_{\mathsf{Inverse}[s]} \ , \tag{2.53}$$

avec $\mathsf{Inverse}[.]$ une table pré-calculée retournant l'inverse d'un élément de $\mathbb{F}_{256}^*$ donné en entrée.

Cependant, le schéma de partage de secret initialement considéré est le schéma de partage de secret basique défini sur $\mathbb{F}_{256}$ et non le schéma multiplicatif défini sur $\mathbb{F}_{256}^*$. De ce fait, des méthodes de conversion pour passer d'un schéma de partage de secret à un autre sont nécessaires. Dans cette sous-section, on présente succinctement la méthode proposée dans [GPQ11b] par Laurie Genelle, Emmanuel Prouff et Michaël Quisquater pour y remédier. Cette solution s'inspire de leurs précédents articles [GPQ10, GPQ11a]. Les procédures de conversion proposées dans [GPQ11b], appelées AMtoMM et MMtoAM

vérifient :

$$\begin{array}{rcl} \mathsf{AMtoMM}(x_1,\ldots,x_t,x+\sum_{i=1}^{t}x_i) & \rightarrow & (z_1,\ldots,z_t,x\times\prod_{i=1}^{t}z_i) \\ \mathsf{MMtoAM}(z_1,\ldots,z_t,x\times\prod_{i=1}^{t}z_i) & \rightarrow & (x_1,\ldots,x_t,x+\sum_{i=1}^{t}x_i) \end{array} \tag{2.54}$$

avec $x, x_1, z_1, \ldots, x_t, z_t \in \mathbb{F}_{256}^*$.
La procédure AMtoMM consiste à transformer successivement chacune des coordonnées additives $x_i$, pour $i = 1, \ldots, t$, en coordonnées multiplicatives. Pour ce faire, cette procédure considère les ensembles : $S_{MV} = \{x_{t+1} = x + \sum_{i=1}^{t} x_i\}$, $S_{AM} = \{x_1, \ldots, x_t\}$ et $S_{MM} = \varnothing$, puis effectue successivement pour $i = 1, \ldots, t$, le traitement suivant :

1. multiplier l'élément de $S_{MV}$ et chaque élément de $S_{AM}$ par $z_i \in_R \mathbb{F}_{256}^*$,
2. insérer $z_i$ dans l'ensemble $S_{MM}$,
3. supprimer le premier élément de $S_{AM}$ et l'additionner à l'élément de $S_{MV}$.

Inversement, la procédure MMtoAM transforme successivement chacune des coordonnées multiplicatives $z_i$, pour $i = 1, \ldots, t$, en coordonnées additives. En supposant les ensembles précédents initialisés tels que $S_{MV} = \{z_{t+1} = x \times \prod_{i=1}^{t} z_i\}$, $S_{AM} = \varnothing$ et $S_{MM} = \{z_1, \ldots, z_t\}$, cette procédure effectue successivement pour $i = 1, \ldots, t$, le traitement suivant :

1. additionner l'élément de $S_{MV}$ avec $x_i \in_R \mathbb{F}_{256}^*$,
2. insérer $r_i$ dans l'ensemble $S_{AM}$,
3. supprimer $z_i$ de $S_{MM}$, puis multiplier par $z_i^{-1}$ l'élément de $S_{MV}$ et chacun des éléments de $S_{AM}$.

Les algorithmes relatifs à ces deux procédures de conversion sont données dans [GPQ11b] par les algorithmes 1 et 2.

Afin de pouvoir appliquer ces deux procédures, les éléments considérés (*i.e.,* $x$, $x_1$, $z_1$, ..., $x_t$ et $z_t$) doivent être définis sur $\mathbb{F}_{256}^*$. Pour que cette condition soit remplie, les auteurs proposent d'appliquer (avant la procédure de conversion AMtoMM) une procédure, appelée SecureDirac, permettant d'obtenir des vecteurs définis sur $(\mathbb{F}_{256}^*)^{t+1}$ à partir des vecteurs de partage basique définis sur $\mathbb{F}_{256}^{t+1}$. Cette dernière permet notamment de gérer le cas où le secret partagé vaut 0. Elle utilise la fonction de Dirac, notée $\delta_0$, définie pour tout $x \in \mathbb{F}_{256}$ par :

$$\delta_0(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{sinon} \end{cases} \tag{2.55}$$

L'implantation de cette procédure sur un composant ayant un processeur de $l$ bits (généralement $l = 8$), s'applique sur $l$ vecteurs de partage de sorte à manipuler $t + 1$ matrices binaires de dimension $l \times l$ où la matrice $i$, pour $i = 1 \ldots, t + 1$, est composée des $i$-èmes parts des $l$ vecteurs. En supposant $l = 8$, cette procédure s'applique sur les 8 premiers vecteurs de partage $v_{x_0}, \ldots, v_{x_7}$ puis indépendamment sur les 8 derniers : $v_{x_8}, \ldots, v_{x_{15}}$

décrits dans la relation (2.42). L'application de cette procédure sur 8 vecteurs de partage retourne alors 8 nouveaux vecteurs de longueur $t+1$ dont les éléments sont tous définis sur $\mathbb{F}_{256}^*$. De plus, cette procédure garde en mémoire $t+1$ vecteurs binaires : $D_1, \ldots, D_{t+1}$ de dimension $l$ permettant de se souvenir des modifications apportées sur les 8 vecteurs initialement considérés.

À ce stade, la procédure de conversion AMtoMM peut alors être appliquée et ainsi la procédure d'inversion décrite comme une transformation stable peut être effectuée indépendamment sur chacune des coordonnées des 8 vecteurs de partage. Une fois ce traitement effectué, la procédure conversion MMtoAM peut être appliquée sur ces 8 vecteurs. Les 8 vecteurs additifs ainsi obtenus sont alors ajustés à l'aide du vecteur $(D_1, \ldots, D_{t+1})$ de sorte que la procédure d'inversion retourne bien 8 vecteurs de partage associés aux 8 secrets attendus. Cette procédure d'inversion est décrite par l'algorithme 9 qui s'applique sur $l$ vecteurs de partage directement. À titre d'indication, le coût des différentes procédures appelées dans cette procédure d'inversion est précisé dans la table 2.2. Plus de détails sur la procédure SecureDirac peuvent être trouvés dans [GPQ11a].

| | aléa | add | and | mult |
|---|---|---|---|---|
| SecureDirac ([GPQ11a, Alg. 4]) | $7t(t+1)$ bits | $(28t+16)(t+1)$ | $14t(t+1)^2$ | - |
| AMtoMM ([GPQ11b, Alg. 1]) | $8t(t+1)$ octets | $16t^2$ | - | $8t(t+3)$ |
| MMtoAM ([GPQ11b, Alg. 2]) | $24t(t+1)$ octets | $16t(t+2)$ | - | $8t(t+3)$ |

*Le symbole* ***aléa*** *désigne le nombre d'éléments générés uniformément sur* $\mathbb{F}_{256}$ *ou* $\mathbb{F}_{256}^*$ *;* ***and*** *désigne l'opération de multiplication bit à bit ;* ***add*** *et* ***mult*** *indiquent respectivement le nombre d'additions et de multiplications entre 2 éléments de* $\mathbb{F}_{256}$.

TABLE 2.2 – Coût des procédures résistantes aux attaques d'ordre $t$ requises par l'algorithme 9

**Notation 2.4.4.** *Dans l'algorithme 9, les éléments en gras représentent $l$ éléments de $\mathbb{F}_{256}$ ou sur $\mathbb{F}_{256}^*$, par exemple, pour $i = 1, \ldots, t+1$ :*

$$\mathbf{x_i} = \begin{pmatrix} x_i^{[1]} \\ \vdots \\ x_i^{[l]} \end{pmatrix}, \tag{2.56}$$

*avec $x_i^{[1]}, \ldots, x_i^{[l]} \in \mathbb{F}_{256}$ ou $\mathbb{F}_{256}^*$.*

**Algorithme 9** Procédure d'inversion de Genelle, Prouff et Quisquater [GPQ11b]

ENTRÉES: $E = \{(s, v_s) : s \in S, (c_1, \ldots, c_{t+1}) \in E_s \subseteq (\mathbb{F}_{256})^{t+1}\}$ un ensemble associé à un schéma de partage de secret basique de paramètre de sécurité $t$.
$(\mathbf{x}_1, \ldots, \mathbf{x}_{t+1})$ : $l$ vecteurs de partage tels que pour tout $j = 1, \ldots, l$ : $(s^{[j]}, x_1^{[j]}, \ldots, x_{t+1}^{[j]}) \in E$
SORTIE: $(\mathbf{z}_1, \ldots, \mathbf{z}_{t+1})$ : $l$ vecteurs de partage tels que pour tout $j = 1, \ldots, l$ :
$((s^{[j]})^{-1}, z_1^{[j]}, \ldots, z_{t+1}^{[j]}) \in E$ (cf. notation 2.4.4)

1. $[(D_1, \ldots, D_{t+1}), (\mathbf{y}_1, \ldots, \mathbf{y}_{t+1})] \leftarrow \mathsf{SecureDirac}(\mathbf{x}_1, \ldots, \mathbf{x}_{t+1})$
2. $(\mathbf{y}_1, \ldots, \mathbf{y}_{t+1}) \leftarrow \mathsf{AMtoMM}(\mathbf{y}_1, \ldots, \mathbf{y}_{t+1})$
3. **Pour** $j = 1$ *à* $l$ **faire:**
4. $\quad (y_1^{[j]}, \ldots, y_{t+1}^{[j]}) \leftarrow \left(\mathsf{Inverse}[y_1^{[j]}], \ldots, \mathsf{Inverse}[y_{t+1}^{[j]}]\right)$
5. $(\mathbf{x}_1, \ldots, \mathbf{x}_{t+1}) \leftarrow \mathsf{MMtoAM}(\mathbf{y}_1, \ldots, \mathbf{y}_{t+1})$
6. $(\mathbf{z}_1, \ldots, \mathbf{z}_{t+1}) \leftarrow (\mathbf{x}_1 + D_1, \ldots, \mathbf{x}_{t+1} + D_{t+1})$
7. **Retourner** $(\mathbf{z}_1, \ldots, \mathbf{z}_{t+1})$

#### 2.4.2.3 Utilisation du schéma de partage de secret de Shamir

Dans cette sous-section, on présente les solutions proposées dans [GM11, PR11b] utilisant le schéma de partage de secret de Shamir. Considérons l'ensemble $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \in E_s \subseteq (\mathbb{F}_{256})^n\}$ associé à un tel schéma de paramètre de sécurité $t$, avec $t < n$. Les transformations linéaires sur $\mathbb{F}_{256}$ de l'AES seront effectuées comme décrites en introduction de la sous-section 2.4.2, en considérant le schéma de partage de secret de Shamir. Les résultats temporaires en entrée et en sortie de la transformation *SubBytes* sont donc chacun constitué de 16 vecteurs de partage associés à ce schéma. Il reste à présenter l'implantation de la transformation *SubBytes*. Cette transformation consiste à appliquer, indépendamment sur chacun des vecteurs de partage, une procédure d'inversion partant de la relation (2.48), puis à effectuer la transformation affine représentée sous forme polynômiale.

Pour effectuer la procédure d'inversion, les auteurs de [GM11, PR11b] se sont basés sur la procédure de multiplication introduite dans le contexte du calcul multi-parties par

les articles [BOGW88, CCD88]. Rappelons cette procédure.
Considérons l'ensemble $E$ associé au schéma de partage de secret de Shamir avec $n = 2t + 1$ parts, où $t$ est le paramètre de sécurité. Chaque vecteur de partage d'un tel schéma est constitué de l'évaluation d'un polynôme de degré $t$ en $2t + 1$ points non nuls distincts : $x_1, \ldots, x_{2t+1}$. Soient $(s, v_s), (s', v_{s'}) \in E$, avec $v_s = (f(x_1), \ldots, f(x_{2t+1}))$ et $v_{s'} = (f'(x_1), \ldots, f'(x_{2t+1}))$, où $f$ et $f'$ sont deux polynômes de degré $t$. En multipliant ces vecteurs entre eux, coordonnée par coordonnée, on obtient un nouveau vecteur dont les coordonnées correspondent à l'évaluation d'un troisième polynôme :

$$\begin{array}{rcl}(s, v_s) * (s', v_{s'}) & = & (ss', f(x_1)f(x_1), \ldots, f(x_{2t+1})f'(x_{2t+1})) \\ & = & (ss', g(x_1), \ldots, g(x_{2t+1}))\end{array} \tag{2.57}$$

Cependant, ce vecteur n'est pas associé à un schéma de partage de secret de Shamir de paramètre de sécurité $t$, mais à schéma de partage de secret de Shamir de paramètre de sécurité $2t$. En effet, ce vecteur obtenu est constitué de l'évaluation d'un polynôme $g$ de degré $2t$ en $2t + 1$ points. Pour conserver les propriétés du schéma de partage de secret initial, notamment afin de pouvoir reconstruire le secret partagé à partir de $t + 1$ parts seulement, une méthode permettant de réduire le degré du polynôme de $2t$ à $t$ est alors primordiale. Dans le cas contraire, une succession de $p$ multiplications requerrait $pt + 1$ parts.

Considérons un schéma de partage de secret de Shamir entre $2t+1$ parts de paramètre de sécurité $2t$ et notons $\widehat{\lambda} = (\widehat{\lambda}_1, \ldots, \widehat{\lambda}_{2t+1})$ un vecteur défini tel que $\widehat{\lambda}_j = \prod_{j=1, j\neq i}^{2t+1} \frac{-x_j}{x_i - x_j}$, pour $j = 1, \ldots, 2t+1$ (*i.e.,* des polynômes de Lagrange évalués en 0). La solution proposée dans [BOGW88, CCD88] pour réduire le degré d'un polynôme partagé consiste dans un premier temps à multiplier par $\widehat{\lambda}_j$ chacune des coordonnées $j$ du vecteur de partage résultat pour $j = 1, \ldots, 2t + 1$. Ensuite elle requiert d'appliquer la procédure de partage fournie avec $E$ sur chacune des coordonnées $c_j c'_j \widehat{\lambda}_j$ (pour $j = 1, \ldots, 2t + 1$). Enfin, en additionnant entre eux les $2t + 1$ nouveaux vecteurs de partage de $c_j c'_j \widehat{\lambda}_j$, coordonnée par coordonnée, on obtient un vecteur de partage constitué de l'évaluation d'un polynôme de degré $t$ en $2t + 1$ points non nuls distincts. Par construction, le secret partagé par un tel vecteur est bien ce que l'on voulait : $\sum_{i=1}^{2t+1} c_j c'_j \widehat{\lambda}_j = ss'$.

Cependant, cette solution nécessite de considérer $n = 2t + 1$ parts, au lieu de $t + 1$ comme attendu. En appliquant cette solution dans le cas de l'AES, cet inconvénient se révélerait coûteux lors des transformations linéaires qui seraient alors effectuées avec plus de parts que nécessaire. Pour réduire ce coût, Louis Goubin et Ange Martinelli dans [GM11] ont proposé deux améliorations de la procédure de multiplication.
La première est une adaptation de la méthode présentée précédemment. Cette solution considère seulement $n = t + 1$ parts durant les applications linéaires et génère à la volée les $t$ parts manquantes afin de pouvoir appliquer la procédure de multiplication avec les $2t + 1$ parts requises (cf. Alg. 10).

**Algorithme 10** Procédure de multiplication de Goubin et Martinelli [GM11, Adaptation de l'algorithme 2]

ENTRÉES: $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \in E_s \subseteq (\mathbb{F}_{256})^{t+1}\}$ un ensemble associé à un schéma de partage de secret de Shamir de paramètre de sécurité $t$.
$2t+1$ éléments publics distincts non-nuls : $x_1, \ldots, x_{2t+1}$
Les éléments $\beta_j(x_i)(= \prod_{k\neq j, k=1}^{t+1} \frac{x_i - x_k}{x_j - x_k})$ pré-calculés pour $1 \leqslant j \leqslant t+1$ et $t+2 \leqslant i \leqslant 2t+1$
Les éléments $\widehat{\lambda}_i(= \prod_{j\neq i, j=1}^{2t+1} \frac{-x_j}{x_i - x_j})$ pré-calculés pour $1 \leqslant i \leqslant 2t+1$
$(c_1, \ldots, c_{t+1}) \in E_s$ et $(c'_1, \ldots, c'_{t+1}) \in E_{s'}$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{ss'}$

*Génération à la volée des t parts supplémentaires :*

1. $(c_1, \ldots, c_{t+1}, c_{t+2}, \ldots, c_{2t+1}) \leftarrow \left(c_1, \ldots, c_{t+1}, \sum_{j=1}^{t+1} c_j\beta_j(x_{t+2}), \ldots, \sum_{j=1}^{t+1} c_j\beta_j(x_{2t+1})\right)$
2. $\left(c'_1, \ldots, c'_{t+1}, c'_{t+2}, \ldots, c'_{2t+1}\right) \leftarrow \left(c'_1, \ldots, c'_{t+1}, \sum_{j=1}^{t+1} c'_j\beta_j(x_{t+2}), \ldots, \sum_{j=1}^{t+1} c'_j\beta_j(x_{2t+1})\right)$

*Procédure de multiplication :*

3. $(w_1, \ldots, w_{2t+1}) \leftarrow \left(c_1c'_1\widehat{\lambda}_1, \ldots c_{2t+1}c'_{2t+1}\widehat{\lambda}_{2t+1}\right)$
4. **Pour** $i = 1$ *à* $2t+1$ **faire:**
   $v_{w_i} \leftarrow \mathsf{partage}(w_i)$
5. $(z_1, \ldots, z_{t+1}) \leftarrow \sum_{i=1}^{2t+1} v_{w_i}$
6. **Retourner** $(z_1, \ldots, z_{t+1})$

Il est à noter que dans la solution donnée par l'algorithme 10, les deux premières étapes peuvent s'effectuer de manière résistante aux attaques d'ordre $t$. En effet, supposons par exemple, un attaquant pouvant observer $t$ valeurs durant l'exécution d'un des calculs de $c_{t+1+i}$ pour $i = 1, \ldots, t$ effectué comme :

$$\begin{array}{rcl} c_{t+1+i} & \leftarrow & c_1\beta_1(x_{t+1+i}) \\ c_{t+1+i} & \leftarrow & c_{t+1+i} + c_2\beta_2(x_{t+1+i})) \\ \vdots & \vdots & \vdots \\ c_{t+1+i} & \leftarrow & c_{t+1+i} + c_{t+1}\beta_{t+1}(x_{t+1+i})) \end{array} \tag{2.58}$$

Durant cette exécution, un attaquant observant $t$ valeurs intermédiaires ne pourra pas obtenir plus d'information que s'il détenait directement $t$ coordonnées. En effet, comme chaque $t$-uplet de valeurs intermédiaires est uniformément distribué sur $\mathbb{F}_{256}^t$, ils sont donc chacun indépendant du secret partagé par le vecteur $(c_1, \ldots, c_{t+1})$. Par cette remarque, il est direct de voir que les deux premières étapes peuvent s'effectuer de manière sécurisée.

Afin de confirmer l'efficacité de leur proposition, la table 3.1 compare le coût de l'algorithme 10 avec celui de la méthode dite standard décrite dans [BOGW88, CCD88]. De plus, la table 3.2 est une application numérique de la table précédente pour $t = 1, \ldots, 6$. D'après cette table, on peut constater que l'algorithme 10 est bien plus efficace que la

procédure standard [BOGW88, CCD88] dès $t \geqslant 2$.

La seconde amélioration proposée dans [GM11] est nettement plus efficace que la précédente solution. En effet, à partir de deux vecteurs de partage $(c_1, \ldots, c_{t+1}) \in E_s$ et $(c'_1, \ldots, c'_{t+1}) \in E_{s'}$, cette dernière consiste simplement à déterminer un nouveau vecteur de partage $(z_1, \ldots, z_{t+1})$ satisfaisant pour $i \in \{1, \ldots, t+1\}$ :

$$z_i \leftarrow \sum_{j=1}^{t+1} \sum_{k=1}^{t+1} (c_j c'_k + c_k c'_j) \beta_{j,k}(x_i) \ , \tag{2.59}$$

où les éléments $\beta_{j,k}(x_i)$ sont pré-calculés et définis pour tout $j, k = 1 \ldots, t+1$ par :

$$\begin{aligned} \beta_{j,k}(x_i) &= \mathsf{Tronc}(\beta_j(x)\beta_k(x)) \\ &= \mathsf{Tronc}(\alpha_{2t} x^{2t} + \ldots + \alpha_t x^t + \ldots + \alpha_1 x + \alpha_0) \\ &= \alpha_t x^t + \ldots + \alpha_1 x + \alpha_0 \end{aligned}$$

Cependant il est montré dans [CPR12], que cette seconde solution contient une faiblesse permettant théoriquement de révoquer la sécurité de leur proposition en appliquant une attaque de premier ordre quelque soit le paramètre de sécurité $t$.

Pour appliquer la procédure d'inversion sur un vecteur de partage défini sur $\mathbb{F}_{256}$ à partir de la relation (2.48), une procédure d'élévation à la puissance $2^k$, avec $k > 1$ est requise. Comme $\mathbb{F}_{256}$ est de caractéristique 2, cette dernière est moins coûteuse qu'une procédure de multiplication car elle peut être effectuée en manipulant seulement $t+1$ parts. En effet, on peut remarquer que pour tout polynôme $f$ de degré $t$, on a :

$$(f(X))^{2^k} = \left(s + a_1 X + \ldots + a_t X^t\right)^{2^k} = s^{2^k} + a_1^2 X^{2^k} + \ldots + a_t^2 X^{2^k t} \ . \tag{2.60}$$

En posant $Y = X^{2^k}$, le polynôme précédent devient un polynôme de degré $t$. Considérons le vecteur de partage constitué des évaluations du polynôme $f$ en les points $x_1, \ldots, x_{t+1}$ : $v_s = (f(x_1), \ldots, f(x_{t+1}))$. Le vecteur de partage suivant $(f(x_1)^{2^k}, \ldots, f(x_{t+1})^{2^k})$ est donc un vecteur de partage de $s^{2^k}$. Cependant $(s^{2^k}, f(x_1)^{2^k}, \ldots, f(x_{t+1})^{2^k}) \notin E$, car il s'agit de l'évaluation du polynôme $f$ en les points $(x_1^{2^k}, \ldots, x_{t+1}^{2^k})$ et non pas en $(x_1, \ldots, x_{t+1})$.
Pour conserver les propriétés du schéma de partage de secret, *i.e.,* afin d'obtenir un vecteur de partage constitué des évaluations d'un polynôme en les points $x_1, \ldots, x_{t+1}$, les auteurs de [GM11] proposent d'appliquer la procédure de partage sur les coordonnées $f(x_j)^{2^k} \beta_j(0)^{2^k}$. Un vecteur de partage $(y_1, \ldots, y_{t+1})$ de $E_{s^{2^k}}$ est alors obtenu en additionnant entre eux les $t+1$ vecteurs de partage obtenus. Ce dernier est bien un vecteur associé à $s^{2^k}$, car la reconstruction donne

$$\mathsf{reconstruction}(y_1, \ldots, y_{t+1}) = \sum_{i=1}^{t+1} y_i \beta_i(0) = \sum_{j=1}^{t+1} f(x_j)^{2^k} \beta_j(0)^{2^k} = s^{2^k} \ . \tag{2.61}$$

De plus, dans la version longue de [PR11a], Emmanuel Prouff et Thomas Roche proposent une seconde solution pour effectuer une élévation de puissance $2^k$. Leur méthode consiste à choisir des points $x_1, \ldots, x_{t+1}$ tels que l'ensemble de ces points soient stables par la fonction de Frobenius $x \mapsto x^2$. Dans ce cas, le traitement requérant $t+1$ procédures de partage ainsi que la somme des vecteurs de partage obtenus entre eux, peut être remplacé par une simple permutation des parts. Plus précisément, considérons des points $x_1, \ldots, x_{t+1}$ deux à deux distincts satisfaisant :

$$\forall i \in \{1, \ldots, t+1\}, \exists j \neq i \in \{1, \ldots, t+1\} \text{ tel que } x_i^2 = x_j \ . \tag{2.62}$$

Pour tout $i \in \{1, \ldots, t+1\}$, on :

$$\beta_i(0)^2 = \prod_{u=1, u\neq i}^{t+1} \frac{x_u^2}{x_i^2 - x_u^2} = \prod_{u=1, u\neq j}^{t+1} \frac{x_u}{x_j - x_u} = \beta_j(0) \ . \tag{2.63}$$

De plus, considérons un vecteur de partage $(c_1, \ldots, c_{t+1}) \in E_s$ associé à un polynôme $f(x) = s + a_1 x + \ldots + a_t x^t$. Par définition, on a :

$$(\mathsf{reconstruction}(c_1, \ldots, c_{t+1}))^2 = \left(\sum_{i=1}^{t+1} c_i \beta_i(0)\right)^2 = \sum_{i=1}^{t+1} c_i^2 \beta_i(0)^2 = \sum_{i=1}^{t+1} f(x_i)^2 \beta_i(0)^2 = s^2 \ . \tag{2.64}$$

Par ailleurs, d'après la relation (2.62), on a

$$f(x_i)^2 = g(x_i^2) = g(x_j) \ , \tag{2.65}$$

avec $g(x) = s^2 + a_1^2 x + \ldots + a_t^2 x^t$.
D'où :

$$s^2 = \sum_{i=1}^{t+1} f(x_i)^2 \beta_i(0)^2 = \sum_{j=1}^{t+1} g(x_j) \beta_j(0) \ . \tag{2.66}$$

L'élévation au carré d'un secret nécessite donc $t+1$ carrés et une permutation des parts.

La procédure d'inversion sur les vecteurs de partage associés à un résultat intermédiaire de l'AES s'effectue de façon sécurisée en appliquant l'algorithme 11, où :

- la procédure de multiplication, notée $\mathsf{SecMult}$, réfère à l'algorithme 10,
- et les procédures d'élévation à la puissance 2,4, et 16 sont effectuées à l'aide d'une transformation stable et d'une permutation de parts. Dans l'algorithme 11, cette procédure est notée $\mathsf{Puissance2^m}$, avec $2^m = 2, 4$ ou 16.

**Algorithme 11** Procédure d'inversion de Goubin et Martinelli [GM11]

ENTRÉES: $E = \{(s, v_s) : s \in \mathbb{F}_{256}, v_s \in E_s \subseteq (\mathbb{F}_{256})^{t+1}\}$ un ensemble associé à un schéma de partage de secret de Shamir de paramètre de sécurité $t$.
$(x_1, \ldots, x_{t+1}) \in E_s$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{s^{254}}$

1. $(y_1, \ldots, y_{t+1}) \leftarrow \mathsf{Puissance2}(x_1, \ldots, x_{t+1})$
2. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}\,((y_1, \ldots, y_{t+1}), (x_1, \ldots, x_{t+1}))$
3. $(w_1, \ldots, w_{t+1}) \leftarrow \mathsf{Puissance4}(z_1, \ldots, z_{t+1})$
4. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}\,((z_1, \ldots, z_{t+1}), (w_1, \ldots, w_{t+1}))$
5. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{Puissance16}(z_1, \ldots, z_{t+1})$
6. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}\,((z_1, \ldots, z_{t+1}), (w_1, \ldots, w_{t+1}))$
7. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{SecMult}\,((z_1, \ldots, z_{t+1}), (y_1, \ldots, y_{t+1}))$
8. **Retourner** $(z_1, \ldots, z_{t+1})$

Enfin pour appliquer la transformation affine à chaque vecteur de partage associé au résultat intermédiaire, une solution efficace décrite dans [MR02] consiste à considérer cette transformation sous sa forme polynômiale :

$$\begin{aligned} \mathsf{TransAff}[X] \;=\; & \{\mathsf{0x05}\}X^{128} + \{\mathsf{0x09}\}X^{64} + \{\mathsf{0xF9}\}X^{32} + \{\mathsf{0x25}\}X^{16} \\ & + \{\mathsf{0xF4}\}X^{8} + X^{4} + \{\mathsf{0xB5}\}X^{2} + \{\mathsf{0x8F}\}X + \{\mathsf{0x63}\}\ . \end{aligned} \tag{2.67}$$

Ainsi cette dernière transformation appliquée à un vecteur de partage requiert $7(t+1)$ élévations au carré (avec permutation des parts), 7 multiplications de vecteur de partage avec un scalaire et 8 additions de vecteurs de partage.

## 2.5 Adéquation du modèle de fuite et des contre-mesures

Dans cette section, les résultats présentés ont fait l'objet d'une publication à Cosade 2012 [CGP+12b].

**Notation 2.5.1.** *Pour plus de clarté, on notera dans la suite en lettre majuscule (par exemple $X$, $Z$, ...) les variables uniformément distribuées sur un ensemble donné, et en lettre minuscule (par exemple $x$,$z$, ...) les valeurs prises par ces variables.*

### 2.5.1 Poids de Hamming vs. distance de Hamming

Pour garantir la sécurité d'une implantation cryptographique contre les attaques d'ordre $t$, une approche courante est de prouver formellement la sécurité du schéma en

considérant un modèle de fuite. Hormis de très rares exceptions, les preuves de sécurité étudiées sont généralement menées dans le modèle de fuite lié au poids de Hamming (cf. définition 1.5.5). C'est en particulier le cas des contre-mesures décrites dans la sous-section 2.4.2. Cependant en pratique, le modèle de fuite de certains composants est lié à la distance de Hamming (cf. définition 1.5.4). À partir de cette observation, on peut se demander si une contre-mesure prouvée sûre dans le modèle de fuite lié au poids de Hamming reste sûre dans le modèle lié à la distance de Hamming. Par une implantation naïve, il est facile de rendre une contre-mesure prouvée sûre dans le modèle lié au poids de Hamming, vulnérable dans le modèle lié à la distance de Hamming. Porter une contre-mesure d'un modèle à un autre est un exercice qui s'avère délicat. En particulier, on présente deux exemples dans les deux sous-sections suivantes.

#### 2.5.1.1 Attaque triviale d'ordre 1

Par un exemple naïf d'implantation, une contre-mesure basée sur le schéma de partage de secret basique (*i.e.,* le masquage booléen) étudiée dans le modèle lié au poids de Hamming peut devenir inefficace lorsqu'on l'applique dans un modèle de fuite lié à la distance de Hamming. En effet, considérons les variables suivantes $Z$, $T$ et $Z+T$ définies sur $\mathbb{F}_q$ (un corps fini de caractéristique 2), où $Z$ est une variable sensible et $T$ une variable uniformément distribué sur $\mathbb{F}_q$. Supposons que l'implantation effectue l'écriture de $Z+T$ dans un registre contenant $T$. Dans ce cas, la consommation produite après cette écriture dépend du poids de Hamming associée à la variable sensible :

$$\mathrm{dist}(Z+T,T) = \mathrm{w_H}(Z)\ . \tag{2.68}$$

Ainsi l'effet de la contre-mesure de masquage est annulé rendant une attaque d'ordre 1 possible.

#### 2.5.1.2 Attaque d'ordre 1 sur une contre-mesure résistante aux attaques d'ordre 2

Dans cette sous-section, on considère l'implantation d'une boîte-S résistante contre les attaques d'ordre 2 dans le modèle de fuite lié au poids de Hamming décrite dans [RDP08] (cf. Algo. 12). Il s'agit d'une adaptation de la solution décrite par l'algorithme 5 dans le chapitre 1.

**Algorithme 12** Implantation d'une boîte-S résistante aux attaques d'ordre 2, définie sur un corps fini $\mathbb{F}_{2^n}$ [RDP08]

ENTRÉES: Soient $\tilde{x} = x + t_1 + t_2 \in \mathbb{F}_{2^n}$ une valeur masquée, $t_1$, $t_2$ deux masques d'entrée, $s_1, s_2$ deux masques de sortie, $F[.]$ une table pré-calculée telle que $F[x] = \mathsf{sbox}(x)$, pour tout $x \in \mathbb{F}_{256}$ Soient $b$ un bit généré aléatoirement et $compare(x, y)$ une fonction de comparaison sécurisée retournant $b$ si $x = y$ et $\bar{b}$ sinon. Notons $\mathrm{R}_0$ et $\mathrm{R}_1$ deux registres distincts
SORTIE: $\mathsf{sbox}(x) + s_1 + s_2$

1. $b \in_R \mathbb{F}_2$
2. **Pour** $a = 0$ *à* $255$
3. $\quad cmp \leftarrow compare(t_1 + a, t_2)$
4. $\quad \mathrm{R}_{cmp} \leftarrow (F[\tilde{x} + a] + s_1) + s_2$
5. **Retourner** $\mathrm{R}_b$

Supposons que l'algorithme 12 est implanté sur un composant dont le modèle de fuite est lié à la distance de Hamming avec un bruit de moyenne nulle. Supposons de plus que ce composant possède un langage assembleur standard dans lequel un registre $\mathrm{R}_A$ est utilisé comme un accumulateur et que chacun des registres utilisées (*i.e.,* $\mathrm{R}_A$, $\mathrm{R}_0$ et $\mathrm{R}_1$) sont initialisés à zéro. À partir de ces suppositions, on peut considérer la quatrième étape de l'algorithme 12 implantée de la manière suivante :

$$\begin{array}{lll} 4.1 & \mathrm{R}_A & \leftarrow \tilde{x} + a \\ 4.2 & \mathrm{R}_A & \leftarrow F(\mathrm{R}_A) \\ 4.3 & \mathrm{R}_A & \leftarrow \mathrm{R}_A + s_1 \\ 4.4 & \mathrm{R}_A & \leftarrow \mathrm{R}_A + s_2 \\ 4.5 & \mathrm{R}_{cmp} & \leftarrow \mathrm{R}_A \end{array} \tag{2.69}$$

Durant cette exécution, où $\tilde{X} = X + T_1 + T_2$, le contenu initial du registre $\mathrm{R}_{cmp}$, noté $R$, satisfait les équations suivantes en fonction de la valeur prise par l'indice de la boucle $a$, de $T_1$ et de $T_2$ :

$$R = \begin{cases} 0 & \text{si } a = 0 \\ 0 & \text{si } a = 1 \text{ et } T_1 + T_2 = 0 \\ 0 & \text{si } a > 0 \text{ et } T_1 + T_2 = a \\ F(\tilde{X} + (a-2)) + S_1 + S_2 & \text{si } a > 1 \text{ et } T_1 + T_2 = (a-1) \\ F(\tilde{X} + (a-1)) + S_1 + S_2 & \text{sinon} \end{cases} \tag{2.70}$$

Dans la suite, on va montrer que la distribution de la variable $R$, définie par (2.70), apporte de l'information sur la variable sensible $X$. Pour ce faire, on distingue deux cas, le premier lorsque $\mathrm{R}_A = \mathrm{R}_{cmp}$ et le second lorsque $\mathrm{R}_A \neq \mathrm{R}_{cmp}$.

**Premier cas : $\mathbf{R}_A = \mathbf{R}_{cmp}$**

Lorsque le registre $\mathrm{R}_{cmp}$ est l'accumulateur (*i.e.,* $\mathrm{R}_A = \mathrm{R}_{cmp}$), l'étape 4.5 décrite par l'implantation détaillée (2.69) est inutile et on peut supposer que le registre $\mathrm{R}_{cmp}$ fuit à chaque étape (en particulier à l'étape 4.1). Comme on suppose que le modèle de fuite du composant est lié à la distance de Hamming, la variable liée aux fuites de consommation associées à l'étape 4.1 de l'implantation détaillée (2.69), notée $Y$, se modélise par :

$$Y = \mathrm{w_H}(R + \tilde{X} + a) + B \ , \tag{2.71}$$

où $R$ réfère à l'état initial de $\mathrm{R}_{cmp}$ avant l'écriture de $\tilde{X} + a$.
À partir des relations (2.70) et (2.71), on en déduit :

$$Y = \begin{cases} \mathrm{w_H}(\tilde{X}) + B & \text{si } a = 0 \\ \mathrm{w_H}(X+1) + B & \text{si } a = 1 \text{ et } T_1 + T_2 = 0 \\ \mathrm{w_H}(X) + B & \text{si } a > 0 \text{ et } T_1 + T_2 = a \\ \mathrm{w_H}(F(\tilde{X} + (a-2)) + S_1 + S_2 + \tilde{X} + a) + B & \text{si } a > 1 \text{ et } T_1 + T_2 = (a-1) \\ \mathrm{w_H}(F(\tilde{X} + (a-1)) + S_1 + S_2 + \tilde{X} + a) + B & \text{sinon} \end{cases} \tag{2.72}$$

Comme $\tilde{X} = X + T_1 + T_2$, avec $T_1, T_2$ des variables indépendantes de $X$ et uniformément distribuées sur $\mathbb{F}_{256}$, la variable $Y$ n'apporte alors aucune information sur $X$ lorsque $a = 0$. On choisit alors d'ignorer ce cas. Ainsi, on a :

$$Y = \begin{cases} \mathrm{w_H}(X) + B & \text{si } T_1 + T_2 = a \\ \mathrm{w_H}(X+1) + B & \text{si } T_1 + T_2 = 0 \ \text{ et } \ a = 1 \\ \mathrm{w_H}(Z) + B & \text{sinon} \end{cases} \tag{2.73}$$

avec $Z$ une variable indépendante de $X$ uniformément distribuée sur $\mathbb{F}_{2^n}$.
À partir de la relation (2.73), on peut montrer que la variable $Y$ dépend de $X$. En effet, l'espérance de $Y$ sachant les valeurs prises par $X$ satisfait :

$$E[Y \mid X = x] = \begin{cases} \frac{1}{2^n} \times (\mathrm{w_H}(x) + \mathrm{w_H}(x+1)) + \frac{2^n-2}{2^n} \times E[\mathrm{w_H}(Z)] & \text{if } a = 1 \\ \frac{1}{2^n} \times \mathrm{w_H}(x) + \frac{2^n-1}{2^n} \times E[\mathrm{w_H}(Z)] & \text{if } a > 1 \end{cases} \tag{2.74}$$

Comme $Z$ est uniformément distribuée sur $\mathbb{F}_{2^n}$, on a :

$$E[Y \mid X = x] = \begin{cases} \frac{1}{2^n} \times (\mathrm{w_H}(x) + \mathrm{w_H}(x+1)) + \frac{n \times (2^n-2)}{2^{n+1}} & \text{if } a = 1 \\ \frac{1}{2^n} \times \mathrm{w_H}(x) + \frac{n \times (2^n-1)}{2^{n+1}} & \text{if } a > 1 \end{cases} \tag{2.75}$$

Lorsque $a > 1$, l'espérance décrite par la relation (2.75) est une fonction affine de $\mathrm{w_H}(x)$ et une fonction affine de $\mathrm{w_H}(x) + \mathrm{w_H}(x+1)$ lorsque $a = 1$. Dans les deux cas, cette

espérance révèle de l'information sur $X$. Un attaquant peut alors cibler le second tour de l'algorithme 12 (*i.e.,* $a = 1$) et récupérer les fuites de consommation relatives à la variable $Y$ définie par la relation (2.71). Comme $X$ est une variable sensible (*i.e.,* correspond à l'addition bit à bit entre une petite partie de la sous-clé et une partie du message clair pouvant être choisie par l'attaquant), on peut appliquer une attaque statistique pour expliciter la dépendance donnée par la relation (2.75) et ainsi retrouver la partir de la clé attaquée.

**Second cas : $\mathbf{R}_A \neq \mathbf{R}_{cmp}$**

Dans ce second cas, on suppose que l'accumulateur $\mathrm{R}_A$ est différent du registre $\mathrm{R}_{cmp}$. Le modèle de fuite étant lié à la distance de Hamming, la fuite produite à l'étape 4.5 de l'implantation donnée par (2.69) dépend donc du contenu du registre $\mathrm{R}_{cmp}$, noté $r$ et du contenu du registre $\mathrm{R}_A$. La variable relative aux fuites de consommation pouvant être produites à cette étape peut être modélisée par :

$$Y = \mathrm{w_H}(R + F(X + T + a) + S) + B \ , \tag{2.76}$$

où $T = T_1 + T_2$ and $S = S_1 + S_2$. À partir de la relation 2.70, la relation (2.76) peut être réécrite en fonction de $a$ et des valeurs pouvant être prises par $T$ :

$$Y = \begin{cases} \mathrm{w_H}(\ F(X+T) + S\ ) + B & \text{si } a = 0 \\ \mathrm{w_H}(\ F(X) + S\ ) + B & \text{si } a = 1 \text{ et } T = (a-1) \\ \mathrm{w_H}(\ F(X) + S\ ) + B & \text{si } a > 0 \text{ et } T = a \\ \mathrm{w_H}(\ D_{a+(a-2)}F(X + (a-2) + (a-1)\ ) + B & \text{si } a > 1 \text{ et } T = (a-1) \\ \mathrm{w_H}(\ D_{a+(a-1)}F(X + (a-1) + T)\ ) + B & \text{sinon} \end{cases} \tag{2.77}$$

où $D_aF$ réfère à la dérivée de $F$ par rapport à $a \in \mathbb{F}_{2^n}$, qui est définie pour tout $x \in \mathbb{F}_{2^n}$ telle que :

$$D_aF(x) = F(x) + F(x + a) \ . \tag{2.78}$$

Dans les trois premiers cas donnés par la relation (2.77), la présence de $S$ implique que la variable $Y$ est indépendante de $X$. En effet, dans ces cas $Y$ est de la forme $\mathrm{w_H}(Z)+B$ où $Z$ est une variable uniformément distribuée sur $\mathbb{F}_{2^n}$ et indépendante de $X$. Dans les derniers cas, $S$ n'apparaît pas. Par conséquent, on peut vérifier que la variable $Y$ dépend de $X$. En effet, grâce à la loi des probabilités totales, pour tout $x \in \mathbb{F}_{2^n}$ et $a = 1$, l'espérance de $Y$ sachant les valeurs prises par la variable $X$ satisfait :

$$E[Y|X = x] = \frac{2\mu}{2^n} + \frac{1}{2^n} \sum_{t=2}^{2^n-1} \mathrm{w_H}(D_aF(x + t)), \tag{2.79}$$

où $\mu = E[\mathrm{w_H}(U)] = n/2$ avec $U$ une variable uniformément distribuée sur $\mathbb{F}_{2^n}$.
Lorsque $a > 1$, cette espérance satisfait :

$$E[Y|X = x] = \frac{\mu}{2^n} + \frac{1}{2^n}\,\mathrm{w_H}(D_{a+(a-2)}F(x + (a-2) + (a-1)))$$

$$+ \frac{1}{2^n} \sum_{t=0, t\neq a,(a-1)}^{2^n-1} \mathrm{w_H}(D_{a+(a-1)}F(x + (a-1) + t)). \tag{2.80}$$

D'un point de vue algébrique, la somme explicitée dans (2.79) (respectivement celle explicitée dans (2.80)) peut être vue comme étant la moyenne des valeurs prises par $D_aF(x+t)$ (respectivement par $D_{a+(a-1)}F(x + (a-1) + t)$) sur le coset $x + \{t, t \in [2, 2^n - 1]\}$ (respectivement $x + \{t, t \in [0, 2^n - 1]\backslash\{a-1, a\}\}$). Or comme ces cosets ne sont pas tous égaux en fonction d'un $x$ donné, leur moyenne sera probablement différente en fonction des valeurs $x$. À partir des relations (2.79) et (2.80), on peut alors en déduire que la fuite révèle de l'information sur $X$. Par conséquent, des observations de la variable $Y$ peuvent être utilisées pour effectuer une attaque d'ordre 1 ciblant la variable sensible $X$.

### 2.5.2 Adaptation de contre-mesure : méthode intuitive

Comme illustré par les exemples présentés dans la sous-section 2.5.1, une attention particulière doit être portée lors de l'implantation d'une contre-mesure résistante dans le modèle lié au poids de Hamming sur un composant dont modèle de fuite est lié à la distance de Hamming. Une idée naturelle pour y parvenir consiste à effacer le registre utilisé avant chaque nouvelle écriture. Ainsi, la distance de Hamming entre le précédent contenu du registre $R$ et la valeur écrite $X$ (*i.e.,* $\mathrm{dist}(X, R)$) sera remplacée par la séquence suivante :

$$\begin{cases} \mathrm{dist}(0, R) &= \mathrm{w_H}(R) \\ \mathrm{dist}(X, 0) &= \mathrm{w_H}(X) \end{cases} \tag{2.81}$$

On peut noter que cette technique est couramment utilisée en pratique à la fois au niveau *hardware* et *software*.

Contrairement au modèle de fuite lié au poids de Hamming, ce modèle prend en compte une fuite supplémentaire liée au contenu de la mémoire avant écriture. Comme cette fuite supplémentaire est supposée correspondre à la manipulation d'une variable qui a déjà été manipulée, on peut supposer que la fuite liée à $\mathrm{w_H}(R)$ a déjà été prise en compte dans la preuve de sécurité. Par conséquent, on peut supposer que cette solution d'implantation peut permettre d'étendre une preuve de sécurité établie dans le modèle de fuite lié au poids de Hamming à une preuve de sécurité dans le modèle de fuite lié à la distance de Hamming.

Cependant, cette solution n'est plus suffisante dès lors que l'hypothèse suivante devient caduque : *les fuites supplémentaires sont supposées déjà prises en compte dans la preuve de sécurité*. Par exemple, reprenons l'exemple de la contre-mesure résistante à l'ordre 2 décrite par l'algorithme 12 donné dans la sous-section 2.5.1.2. On va montrer que cette solution d'effacement permet dans ce cas de mettre en place une attaque d'ordre 2.
Supposons que le modèle de fuite est lié à la distance de Hamming et que le contenu des registres $\mathrm{R}_b$ et $\mathrm{R}_{\overline{b}}$ est initialement égal à zéro, avec $b \in_R \mathbb{F}_2$. Dans le but de conserver la preuve de sécurité donnée dans le modèle lié au poids de Hamming, on propose d'effacer le registre utilisé avant une nouvelle écriture. Sous ces hypothèses, la quatrième étape de l'algorithme 12 peut être implantée de la façon suivante :

$$\begin{array}{lll} 4.1 & \mathrm{R}_{cmp} & \leftarrow 0 \\ 4.2 & \mathrm{R}_{cmp} & \leftarrow F(\tilde{x}+a)+s_1+s_2 \end{array} \tag{2.82}$$

Comme précédemment, on suppose que le contenu du registre $\mathrm{R}_{cmp}$ avant l'étape 4.1 vaut $R$ (cf. relation (2.70)). De plus, on suppose que le contenu du registre $\mathrm{R}_{cmp}$ est mis à zéro avant l'écriture $Z = F(\tilde{X}+a)+S_1+S_2$ à l'étape 4.2. La variable relative à la fuite de consommation définie par la relation (2.71) est alors remplacée par la séquence suivante :

$$\begin{cases} \mathrm{dist}(R,0)+B_1 = \mathrm{w_H}(R)+B_1 \text{ (étape 4.1)} \\ \mathrm{dist}(0,Z)+B_2 = \mathrm{w_H}(Z)+B_2 \text{ (étape 4.2)} \end{cases} \tag{2.83}$$

Ce modèle n'est pas totalement équivalent au modèle lié au poids de Hamming car le registre $\mathrm{R}_{cmp}$ fuit lorsqu'il est effacé. Pour effectuer une attaque d'ordre 2, on propose d'exploiter les variables relatives aux fuites d'information, notées $Y_1$ et $Y_2$, produites lors de chaque exécution de l'algorithme 12 implanté avec l'étape (2.82). La première variable est associée aux fuites pouvant être produites lors de la manipulation de $\tilde{X}$ :

$$Y_1 = \mathrm{w_H}(\tilde{X}) + B_0 \ . \tag{2.84}$$

La seconde est quant à elle associée aux fuites pouvant être produites à l'étape 4.1 de l'implantation détaillée (2.82) :

$$Y_2 = \mathrm{w_H}(R) + B_1 \ . \tag{2.85}$$

À partir des relations (2.70) et (2.85), on en déduit :

$$Y_2 = \begin{cases} \mathrm{w_H}(0)+B_1 & \text{si } a=0 \\ \mathrm{w_H}(0)+B_1 & \text{si } a=1 \text{ et } T_1+T_2=0 \\ \mathrm{w_H}(0)+B_1 & \text{si } a>0 \text{ et } T_1+T_2=a \\ \mathrm{w_H}(F(\tilde{X}+(a-2))+S_1+S_2)+B_1 & \text{si } a>1 \text{ et } T_1+T_2=(a-1) \\ \mathrm{w_H}(F(\tilde{X}+(a-1))+S_1+S_2)+B_1 & \text{sinon} \end{cases} \tag{2.86}$$

ce qui implique :

$$Y_2 = \begin{cases} \mathrm{w_H}(0) + B_1 & \text{si } a = 0 \\ \mathrm{w_H}(0) + B_1 & \text{si } a = 1 \text{ et } T_1 + T_2 = 0 \text{ ou } 1 \\ \mathrm{w_H}(Z) + B_1 & \text{si } a = 1 \text{ et } T_1 + T_2 \neq 0 \text{ ou } 1 \\ \mathrm{w_H}(0) + B_1 & \text{si } a > 1 \text{ et } T_1 + T_2 = a \\ \mathrm{w_H}(Z) + B_1 & \text{si } a > 1 \text{ et } T_1 + T_2 \neq a \end{cases} \tag{2.87}$$

avec $Z$ une variable indépendante de $X$ et uniformément distribuée sur $\mathbb{F}_{2^n}$.
À partir de la relation (2.87), on peut voir que la variable $Y_2$ est indépendante de $T_1 + T_2$ lorsque $a = 0$. Pour cette raison, on propose d'étudier l'espérance de $Y_2$ seulement lorsque $a > 0$ :

$$E(Y_2) = \begin{cases} \mathrm{w_H}(0) & \text{si } a = 1 \text{ et } T_1 + T_2 = 0 \text{ ou } 1 \\ \mathrm{w_H}(Z) & \text{si } a = 1 \text{ et } T_1 + T_2 \neq 0 \text{ ou } 1 \\ \mathrm{w_H}(0) & \text{si } a > 1 \text{ et } T_1 + T_2 = a \\ \mathrm{w_H}(Z) & \text{si } a > 1 \text{ et } T_1 + T_2 \neq a \end{cases} \tag{2.88}$$

ce qui donne, comme $Z$ est uniformément distribuée sur $\mathbb{F}_{2^n}$ :

$$E(Y_2) = \begin{cases} \mathrm{w_H}(0) & \text{si } a = 1 \text{ et } T_1 + T_2 = 0 \text{ ou } 1 \\ \mathrm{w_H}(0) & \text{si } a > 1 \text{ et } T_1 + T_2 = a \\ \frac{n}{2} & \text{sinon} \end{cases} \tag{2.89}$$

La variable $Y_1$ dépend quant elle de $X + T_1 + T_2$. Par conséquent, on en déduit que la paire $(Y_1, Y_2)$ dépend statistiquement de la variable sensible $X$. De plus, on peut voir d'après la relation (2.89) que la fuite sur $T_1 + T_2$ est maximale lorsque $a = 1$. Un attaquant peut alors cibler le second tour de l'algorithme 12 (*i.e.,* $a = 1$), pour récupérer à chaque exécution, la paire de consommation de fuite correspondant à l'observation de $(Y_1, Y_2)$. Ensuite, à partir de ces mesures de consommation, il va pouvoir exploiter l'information sur $X$ en effectuant une attaque d'ordre 2.

Pour illustrer cette attaque d'ordre 2, on l'a simulée en considérant le modèle de fuite sans bruit et en ciblant la variable sensible $X = M + k^*$, où $M$ correspond à 8 bits du message clair connu par l'attaquant et $k^*$ correspond à 8 bits de la clé secrète (fixe). Pour ce faire, les fuites associées à $Y_1$ et $Y_2$ (produites au tour $a = 1$) ont été combinées à l'aide de la fonction de combinaison appliquant la multiplication normalisée entre deux points de fuite [PRB09]. Ensuite, en appliquant une CPA, la valeur secrète $k^*$ a été retrouvée avec un taux de succès de 99% à l'aide de moins de 200.000 courbes. La figure 2.5.2 représente la convergence de la valeur de corrélation maximale pour les différentes hypothèses de sous-clé en fonction du nombre de courbes de consommation simulées. Chaque courbe correspond à une hypothèse de sous-clé $k$. En particulier, la courbe noire correspond à la bonne hypothèse $k^*$.

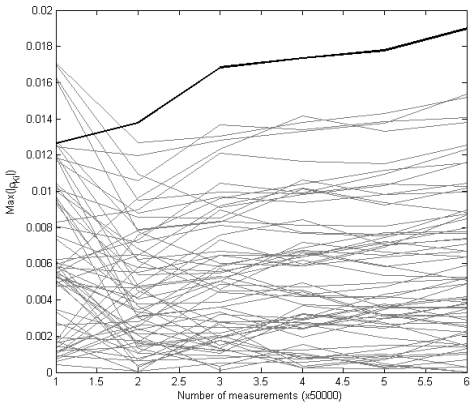



FIGURE 2.1 – Courbes de convergence résultant d'une CPA effectuée à partir de courbes de consommation simulées sans bruit

### 2.5.3 Autre adaptation de contre-mesure

L'attaque d'ordre 2 présentée dans la sous-section 2.5.2 montre que l'effacement des registres avant une nouvelle écriture n'est pas toujours suffisant pour porter une preuve de sécurité d'un modèle lié au poids de Hamming vers celui lié à la distance de Hamming. Des solutions complémentaires doivent être apportées en fonction de l'implantation à protéger. Notamment pour rendre cette attaque inefficace, une solution plus pertinente pourrait par exemple consister à remplacer le contenu du registre $\mathrm{R}_{cmp}$ avant chaque nouvelle écriture par la valeur d'une variable uniformément distribuée sur $\mathbb{F}_{2^n}$. En effet, dans ce cas l'espérance de la variable $Y_2$ vaut $\mathrm{w_H}(Z)$, avec $Z$ une variable uniformément distribuée sur $\mathbb{F}_{2^n}$, et donc l'attaquant ne peut plus mettre en évidence de dépendance avec la variable sensible $X$.

Bien que cette solution est pertinente pour l'algorithme 12, il semble périlleux d'affirmer que cette solution peut être généralisée pour porter n'importe quelle contre-mesure résistante aux attaques d'ordre $t$ dans le modèle liée au poids de Hamming vers une contre-mesure sécurisée dans le modèle liée à la distance de Hamming. L'élaboration et la preuve de sécurité d'une méthode générique reste donc un sujet de recherche ouvert.

## 2.6 Conclusion

Dans ce chapitre, après avoir introduit les attaques d'ordre supérieur, on a présenté les notions associées aux schémas de partage de secret, puis on a exposé leur utilité dans le monde de l'embarqué. Plus précisément, on a explicité le fait que n'importe quel schéma de partage de secret de paramètre de sécurité $t$ peut être appliqué dans le contexte des attaques par canaux cachés pour protéger un cryptosystème des attaques d'ordre $t$. En particulier, on a présenté des exemples d'utilisation de schémas de partage de secret permettant d'implémenter un AES résistant aux attaques d'ordre $t$. Par ces exemples, on a souligné l'intérêt des schémas de partage linéaire à seuil pour implanter les transformations dites linéaires de l'AES. En ce qui concerne les transformations non stables, telles que la transformation *SubBytes*, ce sujet de recherche suscite toujours beaucoup d'intérêt, malgré les nombreuses solutions proposées.

Dans le but de proposer de nouvelles méthodes pour implanter les transformations de l'AES et en particulier la transformation *SubBytes*, on s'intéresse dans le chapitre suivant à une construction de schémas de partage de secret peu connue dans le contexte des attaques par canaux cachés. Cette dernière est basée sur l'utilisation des codes correcteurs d'erreurs, initialement présentée dans [Mas93]. En appliquant de tels schémas de partage de secret dans le contexte des attaques par canaux cachés, on propose ensuite dans le chapitre 4, de nouvelles méthodes pour implanter différentes transformations résistantes aux attaques d'ordre $t$ dans le modèle de fuite lié au poids de Hamming.

# Chapitre 3

# Schémas de partage de secret et codes linéaires

## Sommaire

## 3.1 Introduction

Dans ce chapitre, on s'intéresse à une construction de schémas de partage de secret peu connue dans le domaine des attaques par canaux cachés, à savoir celle explicitée par James L. Massey dans [Mas93] dans les années 1990. Cette construction relie les paramètres d'un schéma de partage de secret linéaire à ceux d'un code linéaire permettant ainsi de tirer profit des résultats de la théorie des codes. Plus précisément, après avoir rappelé les notions nécessaires sur la théorie des codes, on explicitera cette construction qui assure une protection contre les attaques d'ordre supérieur. On présentera ensuite la procédure de multiplication fournie par les codes linéaires. Cette procédure a été présentée par Ronald Cramer *et al.* par exemple dans [CC06, CCCX09] dans le contexte du calcul multi-parties. Il s'agit d'une généralisation, en terme de codes linéaires, de la procédure de multiplication initialement décrite pour le schéma de partage de secret de Shamir [BOGW88, CCD88](cf. sous-section 2.4.2.3). En considérant le contexte des attaques par canaux cachés, on proposera une amélioration de cette procédure sécurisée que l'on appliquera au schéma de partage de secret de Shamir afin de discuter de son efficacité. Enfin, on étudiera la procédure d'élévation à la puissance $p$, lorsque le corps de base est de caractéristique $p$ .

## 3.2 Généralités sur les codes linéaires

Dans cette section, on rappelle des notions fondamentales de la théorie des codes. De plus amples informations dans ce domaine peuvent être retrouvées par exemple dans l'ouvrage [MS77]. Par défaut, un code sera toujours supposé linéaire.

### 3.2.1 Définitions

Soit $\mathbb{F}_q$ le corps fini à $q$ éléments. Un code linéaire $\mathcal{C}$ sur $\mathbb{F}_q$ de longueur $n$ et dimension $k$ est un sous-espace vectoriel de $\mathbb{F}_q^n$ de dimension $k$. Un tel code contient $q^k$ vecteurs de longueur $n$ sur $\mathbb{F}_q$, appelés **mots de code**. Ces derniers peuvent être entièrement déterminés par une matrice $G$ appelée **matrice génératrice** dont les lignes constituent une base de code. À tout vecteur $(r_1, \ldots, r_k) \in \mathbb{F}_q^k$ est associé un mot de code $c = (c_1, \ldots, c_n)$ obtenu en calculant :

$$(c_1, \ldots, c_n) \leftarrow (r_1, \ldots, r_k) \cdot G \ . \tag{3.1}$$

**Définition 3.2.1** (Matrice sous forme systématique)**.** *Une matrice génératrice d'un code linéaire dont la sous-matrice extraite des k premières colonnes constitue la matrice identité de dimension k est dite sous forme systématique.*

**Notation 3.2.2.** *On notera dans la suite $\mathcal{I}_k$, la matrice identité de dimension $k$.*

**Définition 3.2.3** (Codes équivalents)**.** *Deux codes linéaires $\mathcal{C}$ et $\mathcal{C}'$ sont équivalents si $\mathcal{C}'$ peut être obtenu à partir de $\mathcal{C}$ en appliquant une permutation sur les coordonnées.*

À équivalence près, tout code linéaire peut être défini par une matrice génératrice sous forme systématique.

**Définition 3.2.4** (Distance de Hamming)**.** *La distance de Hamming entre deux vecteurs $c = (c_1, \ldots, c_n)$ et $c' = (c'_1, \ldots, c'_n)$ de $\mathbb{F}_q^n$, notée* $\mathrm{dist}(c, c')$*, est le nombre de coordonnées où les deux vecteurs diffèrent :*

$$\mathrm{dist}(c, c') = \mathrm{card}(\{i : i \in \{1, \ldots, n\}, c_i \neq c'_i\}) \ . \tag{3.2}$$

**Définition 3.2.5** (Poids de Hamming)**.** *Le poids de Hamming d'un vecteur $c = (c_1, \ldots, c_n)$ de $\mathbb{F}_q^n$, noté* $\mathrm{w_H}(c)$*, est le nombre de coordonnées $c_i$ différentes de zéro :*

$$\mathrm{w_H}(c) = \mathrm{card}(\{i : i \in \{1, \ldots, n\}, c_i \neq 0\}) \ . \tag{3.3}$$

*On a* $\mathrm{dist}(c, c') = \mathrm{w_H}(c - c')$ *pour tout mot de code $c, c' \in \mathcal{C}$.*

**Définition 3.2.6** (Distance minimale)**.** *La distance minimale, notée $d$, d'un code linéaire $\mathcal{C}$ est égale à :*

$$d = \min\{\mathrm{dist}(c, c') : c, c' \in \mathcal{C}, c \neq c'\} \ . \tag{3.4}$$

*Comme $\mathcal{C}$ est un code linéaire, c'est aussi le plus petit poids d'un mot non nul de $\mathcal{C}$ :*

$$d = \min\{\mathrm{w_H}(c) : c \in \mathcal{C}, c \neq (0, \ldots, 0)\} \ . \tag{3.5}$$

**Définition 3.2.7** (Codes isomorphes)**.** *Deux codes linéaires $\mathcal{C}$ et $\mathcal{C}'$ sont isomorphes si et seulement s'il existe une application bijective $\varphi : \mathcal{C} \to \mathcal{C}'$ qui conserve la linéarité entre deux mots de codes et la distance minimale des mots,* i.e.*, pour $c, c' \in \mathcal{C}$ :*

$$\begin{array}{l} \varphi(c) + \varphi(c') = \varphi(c + c') \\ \mathrm{w_H}(c) = \mathrm{w_H}(\varphi(c)) \end{array} \tag{3.6}$$

*Dans ce cas, on notera : $\mathcal{C} \simeq \mathcal{C}'$.*

Un code linéaire $\mathcal{C}$ de longueur $n$, de dimension $k$ et de distance minimale $d$ défini sur $\mathbb{F}_q$ est dit de paramètres $[n, k, d]_q$ (ou simplement de paramètres $[n, k]_q$ lorsque la distance minimale n'est pas précisée). Lorsque $q = 2$, on dira que le code $\mathcal{C}$ est binaire de paramètres $[n, k, d]$ (ou simplement $[n, k]$).

**Propriété 3.2.8.** (Borne de Singleton, [MS77, p.33]) *Soit $\mathcal{C}$ un code linéaire de paramètres $[n, k, d]_q$. La distance minimale $d$ du code $\mathcal{C}$ vérifie la majoration suivante :*

$$d \leqslant n - k + 1 \ . \tag{3.7}$$

*Si cette borne est atteinte, le code est dit MDS (pour "Maximum Distance Separable").*

### 3.2.2 Dualité et codes auto-duaux

**Définition/Propriété 3.2.9** (Code dual)**.** ([MS77, p.26]) *Le dual $\mathcal{C}^\perp$ d'un code $\mathcal{C}$ de paramètres $[n,k]_q$ est l'ensemble*

$$\mathcal{C}^\perp = \left\{ h \in \mathbb{F}_q^n \ : \langle h, c \rangle = 0 \ , \forall c \in \mathcal{C} \right\} \ , \tag{3.8}$$

*où $\langle h, c \rangle$ désigne le produit scalaire entre $h = (h_1, \ldots, h_n)$ et $c = (c_1, \ldots, c_n)$ :*

$$\langle h, c \rangle = \sum_{i=1}^{n} h_i c_i \ . \tag{3.9}$$

*Le dual $\mathcal{C}^\perp$ est un code de paramètres $[n, n-k]_q$.*

**Notation 3.2.10.** *Dans la suite, on notera $d^\perp$ la distance minimale de $\mathcal{C}^\perp$.*

**Définition 3.2.11** (Code auto-dual)**.** *Si $\mathcal{C} = \mathcal{C}^\perp$, le code $\mathcal{C}$ est dit auto-dual.*

**Définition 3.2.12** (Code faiblement auto-dual)**.** *Si $\mathcal{C} \subsetneq \mathcal{C}^\perp$ le code $\mathcal{C}$ est dit faiblement auto-dual.*

Les codes auto-duaux sont bien connus dans la littérature. Par exemple dans [MS77, p.626], une liste de codes binaires auto-duaux ayant une distance minimale inférieure ou égale à 20 et de longueur inférieure à 104 est présentée.

**Définition 3.2.13** (Matrice de contrôle)**.** *Soient $\mathcal{C}$ un code linéaire de paramètres $[n,k]_q$ et $\mathcal{C}^\perp$ son dual. On appelle matrice de contrôle de $\mathcal{C}$ une matrice génératrice de $\mathcal{C}^\perp$ que l'on notera $H$. Lorsque la matrice génératrice de $\mathcal{C}$ est sous forme systématique : $G = (\mathcal{I}_k | P)$, alors $H = (P^T | \mathcal{I}_{n-k})$, où $P^T$ désigne la transposée de $P$, est une matrice de contrôle de $\mathcal{C}$. Par construction, cette matrice vérifie :*

$$c \in \mathcal{C} \Leftrightarrow H \cdot c^T = (0, \ldots, 0)^T \tag{3.10}$$

**Propriété 3.2.14.** ([MS77, p.33]) *Soit $\mathcal{C}$ un code linéaire de longueur $n$ de matrice de contrôle $H$. Si $\delta$ désigne le nombre maximal de colonnes de $H$ linéairement indépendantes, alors la distance minimale de $\mathcal{C}$ vaut $\delta + 1$.*

**Propriété 3.2.15.** ([MS77, p.318]) *Si $\mathcal{C}$ est un code linéaire MDS, alors $\mathcal{C}^\perp$ l'est aussi.*

### 3.2.3 Construction de code linéaire

Dans cette sous-section, on décrit une propriété et une définition présentant chacune une construction de code linéaire que l'on considèrera à la fois dans ce chapitre et dans le chapitre 4. La première construction requiert le lemme suivant :

**Lemme 3.2.16.** *Considérons $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Soient $L$ un sous espace-vectoriel de $\mathbb{F}_p^n$ de dimension $k$ et $V$ le sous espace-vectoriel de $\mathbb{F}_q^n$ engendré par une base de $L$. La dimension de $V$ vaut $k$.*

*Démonstration.* Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$ de degré $m$. Soient $L$ un sous espace-vectoriel de $\mathbb{F}_p^n$ de dimension $k$ et $V$ le sous espace-vectoriel de $\mathbb{F}_q^n$ engendré par une base de $L$ notée $B = (b_1, \ldots, b_k)$.
Par construction, $B$ engendre $V$, $B$ est donc une famille génératrice de $V$. Il reste donc à montrer que $B$ est une famille libre sur $\mathbb{F}_q$. Pour ce faire, considérons $\lambda_1, \ldots, \lambda_k$ des éléments de $\mathbb{F}_q$ vérifiant la relation suivante :

$$\sum_{i=1}^{k} \lambda_i b_i = 0 \ . \tag{3.11}$$

$\mathbb{F}_q$ étant un $\mathbb{F}_p$-espace vectoriel de dimension $m$, chaque élément de $\mathbb{F}_q$ peut s'écrire par définition comme une combinaison linéaire sur $\mathbb{F}_p$. En considérant une base $(v_1, \ldots, v_m)$ de $\mathbb{F}_q$ sur $\mathbb{F}_p$, on peut alors écrire, pour tous les $\lambda_i \in \mathbb{F}_q$ avec $i = 1, \ldots, k$ :

$$\lambda_i = \sum_{j=1}^{m} u_{i,j} v_j \ , \tag{3.12}$$

avec les $u_{i,j} \in \mathbb{F}_p$ pour $i = 1, \ldots, k$ et $j = 1, \ldots, m$.
En réécrivant la somme donnée par la relation (3.11) à l'aide de la relation précédente, on obtient alors :

$$\sum_{i=1}^{k} \left( \sum_{j=1}^{m} u_{i,j} v_j \right) b_i = 0 \ . \tag{3.13}$$

En notant $b_i = (b_{i,1}, \ldots, b_{i,n}) \in \mathbb{F}_p^n$, pour $i = 1, \ldots, k$, la relation (3.13) devient pour tout $l = 1, \ldots, n$ :

$$\textstyle\sum_{i=1}^{k} \left( \sum_{j=1}^{m} u_{i,j} v_j \right) b_{i,l} = 0 \ , \tag{3.14}$$

$$\Leftrightarrow \quad \textstyle\sum_{j=1}^{m} \left( \sum_{i=1}^{k} u_{i,j} b_{i,l} \right) v_j = 0 \ . \tag{3.15}$$

Comme $(v_1, \ldots, v_m)$ est une base de $\mathbb{F}_q$ sur $\mathbb{F}_p$, on a donc pour $l = 1, \ldots, n$ et $j = 1, \ldots, m$ :

$$\sum_{i=1}^{k} u_{i,j} b_{i,l} = 0 \text{ dans } \mathbb{F}_p \ . \tag{3.16}$$

On en déduit pour tout $j = 1, \ldots, m$ :

$$\sum_{i=1}^{k} u_{i,j} b_i = 0 \text{ dans } \mathbb{F}_p^n \ . \tag{3.17}$$

La famille $B = (b_1, \ldots, b_k)$ étant libre sur $\mathbb{F}_p$, on a alors pour tout $i = 1, \ldots, k$ et $j = 1, \ldots, m : u_{i,j} = 0$. Ce qui implique que pour tout $i = 1, \ldots, k : \lambda_i = 0$. Par conséquent, la famille $B = (b_1, \ldots, b_k)$ est libre sur $\mathbb{F}_q$.

□

**Propriété 3.2.17.** *Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Soit $\mathcal{C}$ un code linéaire de paramètres $[n, k, d]_p$ de matrice génératrice $G$. Le code $\mathcal{C}'$ défini sur $\mathbb{F}_q$, engendré par $G$, est un code linéaire de paramètres $[n, k, d]_q$. Les mots de code de $\mathcal{C}'$ sont construits à partir de $G$ en considérant des éléments dans $\mathbb{F}_q$,* i.e.,

$$\mathcal{C}' = \{(r_1, \ldots, r_k) \cdot G : r_i \in \mathbb{F}_q, i \in \{1, \ldots, k\}\} \ . \tag{3.18}$$

*De façon similaire, on retrouve la même propriété entre les codes duaux de $\mathcal{C}$ et de $\mathcal{C}'$.*

*Démonstration.* Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Considérons $\mathcal{C}$ un code linéaire de paramètres $[n, k, d]_p$ de matrice génératrice $G$ et de code dual $\mathcal{C}^\perp$. Soit $\mathcal{C}'$ le code linéaire défini sur $\mathbb{F}_q$ engendré par $G$. Par construction, $G$ est une matrice génératrice de $\mathcal{C}'$, il est alors évident que la longueur de $\mathcal{C}'$ est égale à $n$. De plus, par le lemme 3.2.16, on déduit que la dimension de $\mathcal{C}'$ est égale à celle de $\mathcal{C}$, *i.e.,* à $k$.

Par ailleurs, comme $G$ est une matrice génératrice de $\mathcal{C}$ et de $\mathcal{C}'$ alors on peut en déduire qu'une matrice de contrôle de $\mathcal{C}'$ est aussi une matrice de contrôle de $\mathcal{C}$. En effet, à équivalence près, on peut supposer que la matrice génératrice de $\mathcal{C}$ est sous forme systématique : $G = (\mathcal{I}_k | P)$. Dans ce cas, il est direct d'en déduire la matrice de contrôle associée : $H = (P^T | \mathcal{I}_{n-k})$. Ainsi, la matrice $H$ est par construction aussi une matrice de contrôle de $\mathcal{C}'$. Par le même raisonnement que précédemment, on peut donc en déduire que le code dual de $\mathcal{C}'$ est de même longueur et de même dimension que $\mathcal{C}^\perp$.

Enfin, par la propriété 3.2.14, on en déduit que la distance minimale de $\mathcal{C}'$ vaut donc $d$ et par un raisonnement similaire, on peut aussi en déduire que la distance minimale du code dual de $\mathcal{C}'$ est égale à celle de $\mathcal{C}^\perp$.

□

**Définition 3.2.18.** *Soit $\mathcal{C}$ un code linéaire de paramètres $[n, k]_q$. Le code étendu de $\mathcal{C}$, noté $\overline{\mathcal{C}}$ est le code de paramètres $[n + 1, k]_q$ défini par :*

$$\overline{\mathcal{C}} = \left\{ \left( c_1, \ldots, c_n, \sum_{i=1}^{n} -c_i \right) : (c_1, \ldots, c_n) \in \mathcal{C} \right\} \ . \tag{3.19}$$

## 3.2.4 Quelques exemples de codes linéaires

### 3.2.4.1 Code de parité

Le code de parité binaire de longueur $n$ est un code linéaire de paramètres $[n, n-1]$. Ce code est défini de sorte que la somme des coordonnées de chaque mot de code soit

nulle. La matrice génératrice sous forme systématique d'un tel code est :

$$G = \left( \begin{array}{c|c} & 1 \\ \mathcal{I}_{n-1} & \vdots \\ & 1 \end{array} \right) . \tag{3.20}$$

La matrice de contrôle $H$ correspondante se réduit au vecteur tout à un de longueur $n$ :

$$H = (1, \ldots, 1) = \mathbf{1}_n \ . \tag{3.21}$$

Par la propriété 3.2.14, on déduit que la distance minimale d'un code de parité vaut 2. De plus, d'après la matrice de contrôle $H$, on en déduit directement que le code dual de $\mathcal{C}$ est de paramètres $[n, 1, n]$.

**Remarque 3.2.19.** *Dans la suite, le code de parité de paramètres $[n, n-1, 2]_q$ désignera le code défini sur $\mathbb{F}_q$ de caractéristique 2 engendré par la matrice $G$ comme décrit par la propriété 3.2.17.*

#### 3.2.4.2 Code de Hamming

Le code de Hamming binaire de longueur $n = 2^m - 1$, avec $m \geqslant 2$ peut être construit à partir d'une matrice de contrôle $H$ dont l'ensemble des colonnes est égal à l'ensemble de tous les vecteurs non nuls possibles de $\mathbb{F}_2^m$. Un tel code est de paramètres $[2^m - 1, 2^m - m - 1]$. De plus, comme la somme de deux colonnes de $H$ est par construction égale à une troisième colonne de $H$, on en déduit par la propriété 3.2.14 que la distance minimale d'un code de Hamming est toujours égale à 3 quelque soit la valeur du paramètre $m$.

**Exemple 3.2.20.** *Soit $\mathcal{C}$ un code de Hamming binaire de paramètres $[7, 4, 3]$ (*i.e., *$m = 3$). Une matrice de contrôle $H$ d'un tel code est*

$$H = (P^T | \mathcal{I}_3) = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{3.22}$$

*et la matrice génératrice associée est égale à :*

$$G = (\mathcal{I}_4 | P) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} . \tag{3.23}$$

*En considérant le code étendu de $\mathcal{C}$, appelé code de Hamming étendu, on peut déterminer une matrice génératrice $\overline{G}$ et une matrice de contrôle $\overline{H}$ de $\overline{\mathcal{C}}$ telles que :*

$$\overline{G} = \overline{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} . \tag{3.24}$$

*Le code de Hamming étendu est donc un code auto-dual.*
*De plus par définition, tout mot de code* $c = (c_1, \ldots, c_7) \in \mathcal{C}$ *est associé à un mot de code* $x = (c_1, \ldots, c_7, \sum_{i=1}^{7} c_i) \in \overline{\mathcal{C}}$ *tel que :*

$$\mathrm{w_H}(x) = \begin{cases} \mathrm{w_H}(c) & \textit{si } \mathrm{w_H}(c) = 0 \bmod 2 \\ \mathrm{w_H}(c) + 1 & \textit{sinon} \end{cases} \tag{3.25}$$

*D'après la distance minimale du code de Hamming* $\mathcal{C}$*, on en déduit alors que la distance minimale de* $\overline{\mathcal{C}}$ *vaut 4. Par conséquent, le code de Hamming étendu est de paramètres* $[8, 4, 4]$.

#### 3.2.4.3 Code de Reed-Solomon

Un code de Reed-Solomon est un code MDS de paramètres $[n, k]_q$ avec $n \leqslant q$. Il est construit en considérant l'ensemble des vecteurs de $\mathbb{F}_q^n$ de la forme :

$$(f(x_1), \ldots, f(x_n)) \ , \tag{3.26}$$

où $x_1, \ldots, x_n$ sont des éléments fixes et distincts de $\mathbb{F}_q$ et où $f(X)$ décrit l'ensemble des polynômes de degré $\leqslant k - 1$.
Une matrice génératrice $G$ d'un tel code est donnée par :

$$G = \begin{pmatrix} 1 & \ldots & 1 \\ x_1 & \ldots & x_n \\ x_1^2 & \ldots & x_n^2 \\ \vdots & \vdots & \vdots \\ x_1^{k-1} & \ldots & x_n^{k-1} \end{pmatrix} . \tag{3.27}$$

## 3.3 Schémas de partage de secret et codes linéaires

### 3.3.1 Description de la construction

Comme explicité par James L. Massey dans [Mas93], une équivalence entre les schémas de partage de secret linéaires et les codes linéaires existe. Dans cette sous-section, on rappelle cette équivalence et on explicite la construction d'un schéma de partage de secret établie à partir d'un code linéaire.

Dans un premier temps, considérons un schéma de partage de secret linéaire et idéal entre $n$ participants défini sur le corps $\mathbb{F}_q$ à partir des procédures de partage et de reconstruction décrites dans le chapitre 2 (cf. définition 2.3.1). Montrons qu'à partir d'un tel schéma, on peut construire un code linéaire.

**Notation 3.3.1.** *Comme dans le chapitre 2, $E_s$ désigne l'ensemble des vecteurs de partage d'un secret $s \in S$ pouvant être construit à partir d'une procédure de partage donnée, où $S$ est un ensemble non vide.*

Comme décrit par la définition 2.3.1, un schéma de partage de secret linéaire défini sur $\mathbb{F}_q$ est associé à l'ensemble

$$E = \{(s, v_s) : s \in S, v_s \in E_s\} \subset \mathbb{F}_q^{n+1} \ . \tag{3.28}$$

Par construction, un tel schéma est stable sur $\mathbb{F}_q$ par l'addition de deux éléments et par la multiplication par un scalaire. L'ensemble $E$ est donc un sous-espace vectoriel de $\mathbb{F}_q^{n+1}$. Par définition d'un code linéaire de longueur $n+1$ défini sur $\mathbb{F}_q$, on en déduit qu'à partir d'un schéma de partage de secret linéaire sur $\mathbb{F}_q$, on peut construire un code linéaire dont les mots de code sont les éléments de $E$.

Réciproquement, montrons qu'un code linéaire $\mathcal{C}$ satisfaisant une condition technique peu restrictive donnée par le théorème 3.3.3 permet de définir un schéma de partage de secret. Avant de donner ce théorème, on décrit le lemme suivant qui servira à sa démonstration.

**Lemme 3.3.2.** *Si $\mathcal{C}$ est un code linéaire de paramètres $[n+1, k]_q$, on a l'assertion suivante :*

$$(1, 0, \ldots, 0) \notin \mathcal{C} \ \Rightarrow \ \mathcal{C}^\perp \nsubseteq \{(0, h_1, \ldots, h_n) : h_1, \ldots, h_n \in \mathbb{F}_q\} \ . \tag{3.29}$$

*Démonstration.* Ce lemme se vérifie aisément par un raisonnement par contraposée. En effet, si $\mathcal{C}^\perp \subseteq \{(0, h_1, \ldots, h_n)\}$, on a alors pour tout mot de code $h \in \mathcal{C}^\perp$ :

$$\langle (1, 0, \ldots, 0), h \rangle = 0 \ , \tag{3.30}$$

ce qui impliquerait que $(1, 0, \ldots, 0) \in \mathcal{C}$.

□

**Théoreme 3.3.3.** *Tout code linéaire $\mathcal{C}$ de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$, où $d^\perp$ est la distance minimale de $\mathcal{C}^\perp$, permet de définir un schéma de partage de secret linéaire entre $n$ participants avec $S = \mathbb{F}_q$.*
*De plus, lorsque la matrice génératrice $G$ de $\mathcal{C}$ est sous forme systématique, les algorithmes 13 et 14 permettent de définir respectivement les procédures de partage et de reconstruction de ce schéma de partage de secret. En particulier, la procédure de reconstruction associé à $\mathcal{C}$ utilise un vecteur $\lambda = (\lambda_1, \ldots, \lambda_n)$, appelé **vecteur de reconstruction**, vérifiant : $(1, -\lambda_1, \ldots, -\lambda_n) \in \mathcal{C}^\perp$.*

**Remarque 3.3.4.** *Par la construction donnée dans l'algorithme 13, chaque mot de code de $\mathcal{C}$ est constitué d'une coordonnée secrète $s \in S$ et $n$ coordonnées formant un vecteur de partage de $s$. Le choix de la coordonnée secrète étant arbitraire, on supposera dans la suite la première coordonnée de chaque mot de code comme secrète.*

**Notations 3.3.5.** *Dans la suite, on pourra noter* $\mathsf{partage}_{\mathcal{C}}(s)$ *et* $\mathsf{reconstruction}_{\mathcal{C}}(v_s)$ *pour préciser respectivement que la procédure de partage appliquée à* $s$ *et celle de reconstruction appliquée à* $v_s$ *sont définies à partir du code linéaire* $\mathcal{C}$*.*

*De plus, on notera* $E_s(\mathcal{C})$ *au lieu de noter simplement* $E_s$*, l'ensemble des vecteurs de partage possibles obtenu en appliquant une procédure de partage à un secret* $s \in S$ *décrite à partir du code* $\mathcal{C}$*.*

**Algorithme 13** Procédure de partage

ENTRÉES: Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$
Considérons une matrice génératrice $G$ de $\mathcal{C}$ sous forme systématique et $s \in S$ un secret
SORTIE: $v_s$ un vecteur de partage appartenant à $E_s(\mathcal{C})$

**Fonction :** $\mathsf{partage}(s)$

1. **Pour** $i = 1$ *à* $k-1$ **faire:**
   Générer $r_i \in_R \mathbb{F}_q$
2. $(s, c_1, \ldots, c_n) \leftarrow (s, r_1, \ldots, r_{k-1}) \cdot G$
3. **Retourner** $v_s \leftarrow (c_1, \ldots, c_n)$

**Algorithme 14** Procédure de reconstruction

ENTRÉES: Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$
Soient $v_s \in E_s(\mathcal{C})$ un vecteur de partage et $\lambda$ un vecteur de reconstruction associé à $\mathcal{C}$
SORTIE: Le secret $s \in S$

**Fonction :** $\mathsf{reconstruction}\,(v_s)$

1. $s \leftarrow \langle v_s, \lambda \rangle$
2. **Retourner** $s$

*Démonstration du théorème 3.3.3.* Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$. Considérons $G$ une matrice génératrice de $\mathcal{C}$ sous forme systématique. Montrons qu'un tel code permet de déterminer un schéma de partage de secret linéaire défini sur $\mathbb{F}_q$.

Par hypothèse, comme $d, d^\perp \geqslant 2$, on a alors $(1, 0, \ldots, 0) \notin \mathcal{C} \cup \mathcal{C}^\perp$. En appliquant le lemme 3.3.2 à $\mathcal{C}^\perp$, on en déduit que $\mathcal{C} \nsubseteq \{(0, c_1, \ldots, c_n) : c_1, \ldots, c_n \in \mathbb{F}_q\}$. Par conséquent, il existe un mot $c = (1, c_1, \ldots, c_n)$. Ainsi pour tout secret $s \in S = \mathbb{F}_q$, il existe alors un mot de code $c = (c_0, c_1, \ldots, c_n) \in \mathcal{C}$ tel que $c_0 = s$.
De même, en appliquant le lemme 3.3.2 à $\mathcal{C}$, on en déduit que $\mathcal{C}^\perp \nsubseteq \{(0, h_1, \ldots, h_n) :$

$h_1, \ldots, h_n \in \mathbb{F}_q\}$. Il existe alors un mot $h = (1, h_1, \ldots, h_n) \in \mathcal{C}^\perp$. Ainsi, pour tout mot de code $c = (s, c_1, \ldots, c_n) \in \mathcal{C}$, on a :

$$\langle (s, c_1, \ldots, c_n), (1, h_1, \ldots, h_n) \rangle = 0 \quad \Longrightarrow \quad s = \sum_{i=1}^{n} c_i(-h_i) \ . \tag{3.31}$$

On peut alors choisir pour vecteur de reconstruction le vecteur suivant : $\lambda = (-h_1, \ldots, -h_n)$. Ainsi le code $\mathcal{C}$ permet de construire un schéma de partage de secret entre $n$ participants sur $\mathbb{F}_q$, pour lequel on a bien pour tout $s \in S = \mathbb{F}_q$ :

$$s = \mathsf{reconstruction}\,(\mathsf{partage}(s)) \ . \tag{3.32}$$

□

**Remarque 3.3.6.** *D'après le théorème 3.3.3 et les notations précédentes, on peut remarquer l'équivalence suivante :*

$$(s, c_1, \ldots, c_n) \in \mathcal{C} \Leftrightarrow (c_1, \ldots, c_n) \in E_s(\mathcal{C}) \ . \tag{3.33}$$

**Propriété 3.3.7.** *Considérons le schéma de partage de secret construit à partir d'un code linéaire $\mathcal{C}$ de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$. La procédure de partage décrite par l'algorithme 13 appliquée à un secret $s \in S$ retourne un vecteur de partage uniformément distribué sur $E_s(\mathcal{C})$.*

*Démonstration.* Considérons le schéma de partage de secret construit à partir d'un code linéaire $\mathcal{C}$ de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$. Montrons que la procédure de partage appliquée à un secret $s \in S$ (Alg. 13) retourne un vecteur de partage uniformément distribué sur $E_s(\mathcal{C})$. Pour ce faire, montrons que pour tout secret $s \in S$, l'application suivante est bijective :

$$\begin{array}{rccc} \varphi_s : & \mathbb{F}_q^{k-1} & \longrightarrow & E_s(\mathcal{C}) \\ & (r_1, \ldots, r_{k-1}) & \longmapsto & (c_1, \ldots, c_n) \end{array} \tag{3.34}$$

où $(c_1, \ldots, c_n)$ est extrait du vecteur $(s, c_1, \ldots, c_n)$ obtenu en calculant $(s, r_1, \ldots, r_{k-1}) \cdot G$, avec $G$ une matrice génératrice de $\mathcal{C}$ sous forme systématique.

Par définition : $\varphi_s(\mathbb{F}_q^{k-1}) = E_s(\mathcal{C})$, l'application $\varphi_s$ est donc surjective. Il reste à montrer que $\varphi_s$ est injective. L'application $\varphi_s$ étant linéaire, déterminons son noyau. Considérons un vecteur $(r_1, \ldots, r_{k-1}) \in \mathbb{F}_q^{k-1}$ et supposons que :

$$\varphi_s(r_1, \ldots, r_{k-1}) = (s, r_1, \ldots, r_{k-1}) \cdot G = (0, \ldots, 0) \ . \tag{3.35}$$

Comme la matrice $G$ est sous forme systématique (*i.e.,* $G = [\mathcal{I}_k | P]$), par identification, on en déduit alors que :

$$\left\{ \begin{array}{rcl} r_1 & = & 0 \\ & \vdots & \\ r_{k-1} & = & 0 \end{array} \right. \tag{3.36}$$

D'où $\mathrm{Ker}(\varphi_s) = (0, \ldots, 0)$, et donc $\varphi_s$ est injective.
Par conséquent, l'application $\varphi_s$ est bijective, ce qui assure que pour tout $s \in S$ et pour tout $r_1, \ldots, r_{k-1} \in_R \mathbb{F}_q : \varphi_s(r_1, \ldots, r_{k-1}) \in_R E_s(\mathcal{C})$.

□

Comme explicité dans le chapitre 2, les schémas de partage de secret sont couramment utilisés dans le monde l'embarqué afin de protéger les implantations cryptographiques des attaques d'ordre $t$, où $t \geqslant 0$ est le **paramètre de sécurité**. Un second paramètre que l'on peut considérer est le **paramètre de reconstruction** noté $r$. Comme introduit dans le chapitre 2, ce paramètre indique le nombre minimum de parts quelconques permettant de reconstruire le secret partagé par un vecteur de partage. En considérant un schéma de partage de secret linéaire construit à partir d'un code linéaire $\mathcal{C}$, le théorème suivant montre que ces deux paramètres dépendent des distances minimales $d$ et $d^\perp$.

**Propriété 3.3.8.** ([CCG+07]) *Un schéma de partage de secret construit à partir d'un code linéaire $\mathcal{C}$ de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$, a pour*

1. *paramètre de sécurité $t \geqslant d^\perp - 2$,*
2. *paramètre de reconstruction $r = n - d + 2$.*

*Démonstration.* Soit $\mathcal{C}$ un code de paramètres $[n+1, k, d]_q$ tel que $d, d^\perp \geqslant 2$.

1. Posons $t' = d^\perp - 2$ et montrons que $t'$ est bien le paramètre de sécurité, *i.e.,* $t = t'$. Soit $s \in S$ un secret et choisissons $t'$ indices distincts $(i_1, \ldots, i_{t'}) \subset \{1, \ldots, n\}$. Montrons que la connaissance de $t'$ coordonnées d'un vecteur de partage $v_s \in E_s(\mathcal{C})$ n'apporte aucune information sur $s$. Pour ce faire, considérons le code linéaire $\mathcal{C}' = \{(s, c_{i_1}, \ldots, c_{i_{t'}}) : (s, c_1, \ldots, c_n) \in \mathcal{C}\}$ et montrons que $\mathcal{C}' = \mathbb{F}_q^{t'+1}$. Par l'absurde, supposons que $\mathcal{C}' \neq \mathbb{F}_q^{t'+1}$, ce qui équivaut à $\mathcal{C}'^\perp$ soit différent de $\{0\}$. Dans ce cas, il existe alors un mot de code $h' = (h'_0, h'_1, \ldots, h'_{t'}) \in \mathcal{C}'^\perp$ avec les $h'_i$ non tous nuls pour $i \in \{0, 1, \ldots, t'\}$. Considérons le mot de code $h \in \mathcal{C}^\perp$ construit à partir de $h'$ tel que $h_0 = h'_0$, $h_{i_j} = h'_j$ pour $i_j \in \{i_1, \ldots, i_{t'}\}$ et $h_i = 0$ sinon, *i.e.,*

$$h = (h'_0, 0, \ldots, 0, h'_1, 0, \ldots, 0, h'_{t'}, 0, \ldots, 0) \ . \tag{3.37}$$

   Ceci implique que $\mathrm{w_H}(h) \leqslant t' + 1 = d^\perp - 1$, ce qui est exclu. On en déduit que $\mathcal{C}' = \mathbb{F}_q^{t'+1}$. Par conséquent la connaissance de $t'$ coordonnées au maximum d'un vecteur de partage $v_s \in E_s(\mathcal{C})$ n'apporte aucune information sur $s$, donc $t = t'$.

2. Posons $r' = n - d + 2$ et montrons que le paramètre de reconstruction $r$ est égal à $r'$. Choisissons $r'$ indices distincts $(i_1, \ldots, i_{r'}) \subset \{1, \ldots, n\}$ et considérons le code linéaire $\mathcal{C}' = \{(s, c_{i_1}, \ldots, c_{i_{r'}}) : (s, c_1, \ldots, c_n) \in \mathcal{C}\}$. Montrons qu'à partir d'un tel code, on peut déterminer un schéma de partage de secret. D'après la preuve du théorème 3.3.3, il suffit de montrer que :

$$(1, 0, \ldots, 0) \notin \mathcal{C}' \cup \mathcal{C}'^\perp \ . \tag{3.38}$$

Si $(1, 0, \ldots, 0) \in \mathcal{C}'$, il existe un mot de code $c \in \mathcal{C}$ tel que $c_{i_j} = 0$ pour $j \in \{1, \ldots, r'\}$. Le poids de Hamming d'un tel mot est alors majoré par :

$$\mathrm{w_H}(c) \leqslant n - (n - d + 2) + 1 = d - 1 \ , \tag{3.39}$$

ce qui est exclu. Par conséquent, on en déduit que $(1, 0, \ldots, 0) \notin \mathcal{C}'$.
Si $(1, 0, \ldots, 0) \in \mathcal{C}'^{\perp}$, alors on peut construire un mot de code $h$ de $\mathcal{C}^{\perp}$ tel que $h = (1, 0, \ldots, 0)$ ce qui est exclu car $d^{\perp} \geqslant 2$. Donc $(1, 0, \ldots, 0) \notin \mathcal{C}'^{\perp}$.
Par conséquent, on a $(1, 0, \ldots, 0) \notin \mathcal{C}' \cup \mathcal{C}'^{\perp}$, et donc au minimum $r' = n - d + 2$ coordonnées quelconques d'un vecteur de partage permettent de reconstruire le secret associé : $r = r'$.

□

**Lemme 3.3.9.** *Soit $\mathcal{C}$ un code linéaire de paramètres $[n + 1, k, d]_q$ tel que $d, d^{\perp} \geqslant 2$. Le code $\mathcal{C}$ est MDS si et seulement si le schéma de partage de secret construit à partir de $\mathcal{C}$ est un schéma de partage de secret linéaire à seuil.*

*Démonstration.* Soit $\mathcal{C}$ un code linéaire de paramètres $[n + 1, k, d]_q$ de dual $\mathcal{C}^{\perp}$ tel que $d, d^{\perp} \geqslant 2$. Par la borne de Singleton, on a :

$$\begin{cases} d & \leqslant & (n+1) - k + 1 = n - k + 2 \\ d^{\perp} & \leqslant & (n+1) - (n+1-k) + 1 = k + 1 \end{cases} \tag{3.40}$$

Supposons que $\mathcal{C}$ soit un code MDS. D'après la propriété 3.2.15, $\mathcal{C}^{\perp}$ est aussi un code MDS. Ainsi $d$ et $d^{\perp}$ atteignent la borne de Singleton. Par définition des paramètres $t$ et $r$, on en déduit :

$$\begin{cases} t & = & d^{\perp} - 2 = k - 1 \\ r & = & n - d + 2 = k \end{cases} \tag{3.41}$$

D'où la relation $r = t + 1$ est vérifiée.
Inversement, supposons que le schéma de partage de secret soit à seuil. Du système (3.40) et de la définition des paramètres $t$ et $r$, on en déduit que $t \leqslant k - 1$ et $r \geqslant k$. Or comme le schéma est supposé à seuil, on en déduit que $r = t + 1 = k + 1$. Par conséquent, le système (3.41) est également vérifié, et donc le code $\mathcal{C}$ est MDS.

□

### 3.3.2 Opérations linéaires

Un schéma de partage de secret déterminé à partir d'un code linéaire est par construction linéaire. Par conséquent, les **transformations stables** telles que les additions de deux secrets et les multiplications d'un secret par un scalaire en utilisant des vecteurs de partage, peuvent s'effectuer efficacement comme pour n'importe quel schéma de partage de secret linéaire.

### 3.3.3 Exemples

Dans cette sous-section, on applique la construction de schémas de partage de secret linéaires à l'aide de codes linéaires, afin de décrire les exemples introduits dans le chapitre 2, à savoir le schéma de partage de secret basique et celui de Shamir définis sur $\mathbb{F}_q$. Ces schémas étant à seuil, les codes associés sont donc des codes MDS. En prenant des codes MDS de longueur $n+1$ et de dimension $k$, le système (3.41) vérifie donc $r = t+1 = k$. Pour déterminer des schémas résistants aux attaques d'ordre $t \geqslant 0$, on considère alors des codes associés au schéma de partage de secret basique et à celui de Shamir de dimension $t+1$.

#### 3.3.3.1 Schéma de partage de secret basique

Le schéma de partage de secret basique défini sur $\mathbb{F}_q$, où $q$ est de caractéristique 2, peut être construit à partir du code de parité $\mathcal{C}$ de paramètres $[t+2, t+1, 2]_q$. En effet, un tel code vérifie bien les conditions attendues par le théorème 3.3.3 sur $d$ et $d^\perp$, pour $t \geqslant 0$ :

$$\begin{cases} d &= 2 \\ d^\perp &= t+2 \end{cases} \tag{3.42}$$

De plus, pour chaque secret $s \in S$, un vecteur de partage $v_s = (c_1, \ldots, c_{t+1}) \in E_s(\mathcal{C})$ est obtenu en générant les éléments $r_1, \ldots, r_t \in_R \mathbb{F}_q$, puis en calculant :

$$(s, r_1, \ldots, r_t) \cdot G = (s, v_s) \ ,$$

où $G$ est la matrice génératrice décrite par la matrice (3.20) en prenant $n = t+2$.
La procédure de reconstruction (Alg. 14) est quant à elle déterminée en prenant $\lambda = \mathbf{1}_{t+1}$ car la matrice de contrôle $H = \mathbf{1}_{t+2}$ (cf. sous-section 3.2.4.1). Ainsi le secret $s$ est reconstruit en calculant :

$$s = \sum_{i=1}^{t+1} c_i \ . \tag{3.43}$$

#### 3.3.3.2 Schéma de partage de secret de Shamir

Quelques années après l'introduction du schéma de partage de secret de Shamir dans [Sha79], Robert J. McEliece et Dilip V. Sarwate ont fait remarquer dans [MS81] que le schéma de partage de secret de Shamir peut être construit à partir d'un code de Reed-Solomon de paramètres $[n+1, t+1]_q$, avec $n \leqslant q$ et $n \geqslant t+1$, où $t \geqslant 0$ est le paramètre de sécurité. Un tel code étant un code MDS, on a alors :

$$\begin{cases} d &= n+1-(t+1)+1 = n-t+1 \\ d^\perp &= t+2 \end{cases} \Rightarrow \begin{cases} d &\geqslant 2 \\ d^\perp &\geqslant 2 \end{cases} \tag{3.44}$$

comme attendu par le théorème 3.3.3.
La procédure de partage utilisant la matrice génératrice suivante :

$$G = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_0 & x_1 & \dots & x_n \\ x_0^2 & x_1^2 & \dots & x_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_0^t & x_1^t & \dots & x_n^t \end{pmatrix}, \tag{3.45}$$

où $x_0 = 0$ et $x_i \neq x_j$ pour $i \neq j \in \{0, 1, \dots, n\}$, revient à évaluer un polynôme de degré $t$ aux points $x_1, \dots, x_n$. La procédure de reconstruction s'effectue quant à elle en considérant le vecteur de reconstruction $\lambda$ défini à partir des polynômes de Lagrange :

$$\lambda_i = \begin{cases} \beta_i(0), \text{ si } i \in Q \\ 0, \text{sinon} \end{cases} \tag{3.46}$$

où $Q$ est un groupe qualifié (cf. sous-section 2.3.2.3).

## 3.4 Procédure de multiplication sécurisée

En considérant des codes linéaires, il est facile d'effectuer efficacement des opérations linéaires en utilisant des vecteurs de partage. Cependant qu'en est-il des opérations non linéaires ? En particulier, on s'intéresse à la multiplication entre deux secrets sur des vecteurs de partage. Cette procédure a été initialement décrite dans [BOGW88, CCD88, GRR98] pour le schéma de partage de secret de Shamir. Elle a ensuite été généralisée pour n'importe quel schéma de partage de secret ayant une propriété dite *multiplicative* dans [CDM00]. Enfin, cette procédure a été plus récemment décrite en termes de codes correcteurs par exemple dans [CC06, CCCX09].

Dans cette section, on présente la généralisation de la procédure de multiplication fournie par les codes linéaires, que l'on nomme **procédure de multiplication standard**. Puis, on propose une amélioration de cette procédure dans le contexte des attaques d'ordre $t$. Enfin, on évalue son efficacité en considérant le schéma de partage de secret de Shamir.

### 3.4.1 Description de la procédure de multiplication standard

Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Considérons un code linéaire $\mathcal{C}$ de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$. Comme vu précédemment, un tel code permet de construire un schéma de partage de secret de paramètre de sécurité $t = d^\perp - 2$.
À partir de deux mots de code de $\mathcal{C}$ : $c = (s, c_1, \dots, c_n)$ et $c' = (s', c'_1, \dots, c'_n)$, une manière

naturelle d'avoir un partage de $ss'$ est d'effectuer le **produit de Schur** entre ces deux vecteurs défini par :

$$c * c' = (s, c_1, \ldots, c_n) * (s', c'_1, \ldots, c'_n) \quad = \quad (ss', c_1c'_1, \ldots, c_nc'_n) \ . \tag{3.47}$$

Cependant, le mot de code résultant de ce produit n'est pas un mot de code de $\mathcal{C}$ et oblige à considérer le code $\widehat{\mathcal{C}}$ défini par :

$$\widehat{\mathcal{C}} = \{c * c' \ : c, c' \in \mathcal{C}\} \ . \tag{3.48}$$

**Notations 3.4.1.** *Dans la suite, la distance minimale de $\widehat{\mathcal{C}}$ est notée $\widehat{d}$. De plus $\widehat{\mathcal{C}}^\perp$ désigne le code dual de $\widehat{\mathcal{C}}$ et $\widehat{d}^\perp$ sa distance minimale.*

Considérons deux vecteurs de partage $v_s \in E_s(\mathcal{C})$ et $v_{s'} \in E_{s'}(\mathcal{C})$ où $s, s' \in S$. Le produit de Schur entre ces deux vecteurs ne conserve pas les propriétés du schéma de partage de secret associé à $\mathcal{C}$ dans le sens où un vecteur de reconstruction $\lambda$ associé au code $\mathcal{C}$, ne permet pas de reconstituer le secret $ss'$. Cependant, si le code $\widehat{\mathcal{C}}$ vérifie $\widehat{d}, \widehat{d}^\perp \geqslant 2$, alors ce code permet de définir un schéma de partage de secret. Dans ce cas, on peut noter :

$$v_{ss'} = v_s * v_{s'} = (c_1c'_1, \ldots, c_nc'_n) \in E_{ss'}(\widehat{\mathcal{C}}) \ . \tag{3.49}$$

De plus, il existe alors un vecteur de reconstruction $\widehat{\lambda} = (\widehat{\lambda}_1, \ldots, \widehat{\lambda}_n)$ associé à $\widehat{\mathcal{C}}$. Par conséquent le secret partagé par $v_{ss'}$ pourra être reconstruit :

$$ss' = \sum_{i=1}^{n} \widehat{\lambda}_i c_i c'_i. \tag{3.50}$$

Le lemme suivant indique la condition sur $\widehat{\mathcal{C}}$ permettant de construire un schéma de partage de secret à l'issue du produit de Schur entre deux vecteurs de partage de $\mathcal{C}$ résistant aux attaques d'ordre $t$, où $t$ est le paramètre de sécurité relatif à $\mathcal{C}$.

**Lemme 3.4.2** (Propriété de multiplication des codes linéaires)**.** *Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$ et de paramètre de sécurité $t = d^\perp - 2$. Le code $\widehat{\mathcal{C}}$ permet de construire un schéma de partage de secret qui conserve le paramètre de sécurité relatif à $\mathcal{C}$ si $\widehat{d} \geqslant 2$ et $\widehat{d}^\perp \geqslant d^\perp$.*

*Démonstration.* Considérons un schéma de partage de secret construit à partir d'un code linéaire $\mathcal{C}$ de paramètres $[n+1, k, d]_q$ de dual $\mathcal{C}^\perp$ tel que $d, d^\perp \geqslant 2$. De plus, supposons que le code $\widehat{\mathcal{C}}$ vérifie $\widehat{d} \geqslant 2$ et $\widehat{d}^\perp \geqslant d^\perp$. On peut noter que comme $\widehat{d} \geqslant 2$ alors $\widehat{\mathcal{C}}^\perp$ est non vide. D'après le théorème 3.3.3, on déduit que $\widehat{\mathcal{C}}$ permet de construire un schéma de partage. De plus comme $\widehat{d}^\perp \geqslant d^\perp$, il s'ensuit que $t = d^\perp - 2 \leqslant \widehat{d}^\perp - 2$. Par conséquent le schéma de partage associé à $\widehat{\mathcal{C}}$ est résistant aux attaques d'ordre $t$.

□

À ce stade, une méthode permettant de *transcoder* le vecteur de partage $v_{ss'} \in E_{ss'}(\widehat{\mathcal{C}})$ en un vecteur de partage $v_{ss'} \in E_{ss'}(\mathcal{C})$ est primordiale afin de conserver les propriétés du schéma de partage de secret relatif à $\mathcal{C}$. Notamment si l'on souhaite effectuer plusieurs multiplications successives ou bien si l'on souhaite reconstituer le secret à partir de la procédure de reconstruction en utilisant le vecteur de reconstruction $\lambda$ (Alg. 14). Une telle méthode est appelée **procédure de transcodage**. Dans le cas général, cette méthode permet d'obtenir un vecteur de partage associé à un code $\mathcal{C}$ à partir d'un vecteur de partage associé à un autre code $\mathcal{C}'$ en appliquant la procédure décrite par l'algorithme 15.

**Algorithme 15** Procédure de transcodage

ENTRÉES: Soient $\mathcal{C}$ et $\mathcal{C}'$ deux codes linéaires de paramètres respectifs $[n+1, k, d]_q$ et $[n'+1, k', d']_q$ vérifiant $d, d^\perp, d', d'^\perp \geqslant 2$, où $d'^\perp$ est la distance minimale duale de $\mathcal{C}'$
Soient $(w_1, \ldots, w_{n'}) \in E_s(\mathcal{C}')$ et $\lambda'$ un vecteur de reconstruction associé à $\mathcal{C}'$
SORTIE: $(z_1, \ldots, z_n) \in E_s(\mathcal{C})$

**Fonction :** $\mathsf{transcodage}_{(\mathcal{C}' \to \mathcal{C})}(\lambda', (w_1, \ldots, w_{n'}))$

1. *Déterminer $n'$ nouveaux vecteurs de partage associés à $\mathcal{C}$*

$$\begin{array}{ccc} (w_{1,1}, \ldots, w_{1,n}) \in E_{\lambda'_1 w_1}(\mathcal{C}) & \leftarrow & \mathsf{partage}_{\mathcal{C}}(\lambda'_1 w_1) \\ \vdots & \vdots & \vdots \\ (w_{n',1}, \ldots, w_{n',n}) \in E_{\lambda'_{n'} w_{n'}}(\mathcal{C}) & \leftarrow & \mathsf{partage}_{\mathcal{C}}(\lambda'_{n'} w_{n'}) \end{array}$$

2. **Pour** $j = 1$ *à* $n$ **faire:**
3. $\quad z_j \leftarrow \sum_{i=1}^{n'} w_{i,j}$
4. **Retourner** $(z_1, \ldots, z_n)$

**Théoreme 3.4.3** (Procédure de transcodage sécurisée)**.** *Soient $\mathcal{C}$ et $\mathcal{C}'$ deux codes linéaires de paramètres respectifs $[n+1, k, d]_q$ et $[n'+1, k', d']_q$ vérifiant $d, d^\perp, d', d'^\perp \geqslant 2$, où $d'^\perp$ est la distance minimale duale de $\mathcal{C}'$. La procédure de transcodage de $\mathcal{C}'$ vers $\mathcal{C}$ s'applique à un vecteur de partage de $E_s(\mathcal{C}')$ et retourne un vecteur de partage de $E_s(\mathcal{C})$. De plus, la procédure décrite par l'algorithme 15 est résistante aux attaques d'ordre $t$, lorsque $d'^\perp - 2 \geqslant t$ et $d^\perp - 2 \geqslant t$.*

*Démonstration.* Soient $\mathcal{C}$ et $\mathcal{C}'$ deux codes linéaires de paramètres respectifs $[n+1, k, d]_q$ et $[n'+1, k', d']_q$ vérifiant $d, d^\perp, d', d'^\perp \geqslant 2$. Soit $(w_1, \ldots, w_{n'}) \in E_s(\mathcal{C}')$. Considérons un paramètre de sécurité $t$, vérifiant $d^\perp - 2 \geqslant t$ et $d'^\perp - 2 \geqslant t$.
Par hypothèse, on peut supposer le vecteur de reconstruction $\lambda'$ associé à $\mathcal{C}'$ public. L'observation de $t$ valeurs intermédiaires durant la manipulation des vecteurs $(w_1, \ldots, w_{n'})$ et

$(\lambda'_1 w_1, \ldots, \lambda'_{n'} w_{n'})$ n'apporte pas d'information sur le secret $s$. Ensuite, la procédure de partage associé à $\mathcal{C}$ est appliquée pour chaque élément $\lambda'_i w_i$ pour $i \in \{1, \ldots, n'\}$ :

$$(\lambda'_i w_i, w_{i,1}, \ldots, w_{i,n}) \leftarrow (\lambda'_i w_i, r_1, \ldots, r_{k-1}) \cdot G \ , \tag{3.51}$$

où $G$ est une matrice génératrice de $\mathcal{C}$ et $r_1, \ldots, r_{k-1} \in_R \mathbb{F}_q$. D'après la propriété 3.3.7, on sait que chaque vecteur de partage de $s$ issu de la procédure de partage est uniformément distribué sur $E_s(\mathcal{C})$. De plus, comme on suppose $d^\perp - 2 \geqslant t$, il est alors facile de voir qu'un attaquant observant $t$ valeurs intermédiaires durant cette étape n'apprendra pas plus d'information sur le secret $s$ qu'en observant directement $t$ valeurs intermédiaires du vecteurs $(w_1, \ldots, w_{n'})$. Enfin, les vecteurs $(w_{i,1}, \ldots, w_{i,n})$ pour $i \in \{1, \ldots, n'\}$ obtenus à l'issue des $n'$ partages correspondent à des mots de code commençant par $\lambda'_i w_i$. Leur somme donne donc un mot de $\mathcal{C}$ commençant par $s$ qui conserve le paramètre de sécurité $t$. La procédure de transcodage décrite par l'algorithme 15 est donc résistante aux attaques d'ordre $t$.

□

Dans le cas de la multiplication entre deux vecteurs de partage, en supposant $\widehat{d} \geqslant 2$ et $\widehat{d}^\perp \geqslant d^\perp$, la procédure de transcodage sécurisée peut alors être appliquée sur un vecteur de partage de $ss'$ associé à $\widehat{\mathcal{C}}$ afin d'obtenir un nouveau vecteur de partage de $ss'$ associé à $\mathcal{C}$. De plus, comme les conditions attendues par la procédure sécurisée de transcodage de $\widehat{\mathcal{C}}$ vers $\mathcal{C}$ sont remplies, on peut alors en déduire le théorème suivant :

**Théoreme 3.4.4** (Procédure de multiplication sécurisée)**.** *Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$ vérifiant $d, d^\perp \geqslant 2$. Supposons que $\widehat{d} \geqslant 2$ et $\widehat{d}^\perp \geqslant d^\perp$. La procédure de multiplication entre deux vecteurs de partage décrite par l'algorithme 16 est résistante aux attaques d'ordre $t = d^\perp - 2$.*

---

**Algorithme 16** Procédure de multiplication standard entre deux vecteurs de partage

---

ENTRÉES: Soient $\mathcal{C}$ et $\widehat{\mathcal{C}}$ deux codes linéaires de paramètres respectifs $[n+1, k, d]_q$ et $[n+1, \widehat{k}, \widehat{d}]_q$ vérifiant $d, d^\perp \geqslant 2$, $\widehat{d} \geqslant 2$ et $\widehat{d}^\perp \geqslant d^\perp$
Soient $(c_1, \ldots, c_n) \in E_s(\mathcal{C})$, $(c'_1, \ldots, c'_n) \in E_{s'}(\mathcal{C})$ et $\widehat{\lambda}$ un vecteur de reconstruction associé à $\widehat{\mathcal{C}}$
SORTIE: $(z_1, \ldots, z_n) \in E_{ss'}(\mathcal{C})$

---

**Fonction :** $\mathsf{SecMult}((c_1, \ldots, c_n), (c'_1, \ldots, c'_n))$

1. $(w_1, \ldots, w_n) \leftarrow (c_1 c'_1, \ldots, c_n c'_n)$
2. $(z_1, \ldots, z_n) \leftarrow \mathsf{transcodage}_{(\widehat{\mathcal{C}} \to \mathcal{C})}(\widehat{\lambda}, (w_1, \ldots, w_n))$
3. **Retourner** $(z_1, \ldots, z_n)$

---

### 3.4.2 Amélioration de la procédure de multiplication

La procédure de multiplication sécurisée décrite par l'algorithme 16 a été initialement introduite dans le contexte du calcul multi-parties, où le modèle d'attaquant est différent de celui considéré dans le contexte des attaques par canaux cachés (cf. chapitre 2). De ce fait, en analysant la procédure de multiplication sécurisée dans le contexte des attaques par canaux cachés, on va montrer que cette procédure peut être améliorée.

**Lemme 3.4.5.** *Soient $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$ et $s \in \mathbb{F}_q$ un secret. Supposons que le code $\widehat{\mathcal{C}}$ vérifie $\widehat{d} \geqslant 2$ et $\widehat{d}^\perp \geqslant d^\perp$. Notons $e = \widehat{t} - t$ l'entier défini à partir de $t$ et de $\widehat{t}$, respectivement les paramètres de sécurité fournis par $\mathcal{C}$ et $\widehat{\mathcal{C}}$. Si $e$ est strictement positif, alors additionner $e+1$ parts d'un vecteur de partage $v_s = (w_1, \ldots, w_n) \in_R E_s(\widehat{\mathcal{C}})$ entre elles, n'apporte aucune information sur le secret $s$ partagé par le vecteur $v_s$. De même additionner $e+1$ parts du vecteur $(\widehat{\lambda}_1 w_1, \ldots, \widehat{\lambda}_n w_n)$ entre elles, où $\widehat{\lambda}$ est un vecteur de reconstruction associé à $\widehat{\mathcal{C}}$, n'apporte aucune information sur $s$.*

*Démonstration.* Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$, avec $d, d^\perp \geqslant 2$ et posons $t = d^\perp - 2$. Considérons un vecteur de partage de $s \in \mathbb{F}_q$ uniformément distribué sur $E_s(\widehat{\mathcal{C}})$ : $v_s = (w_1, \ldots, w_n) \in_R E_s(\widehat{\mathcal{C}})$. Par les propriétés de $\widehat{\mathcal{C}}$, si $e = (\widehat{d}^\perp - 2) - t > 0$, alors l'observation de $t+e$ valeurs intermédiaires n'apporte alors aucune information sur le secret $s$. Or le niveau de sécurité attendu reste quant lui égal à $t$, *i.e.,* un attaquant ne peut seulement observer $t$ valeurs intermédiaires. Ainsi en regroupant $e+1$ parts entre elles (par exemple les $e+1$ premières), on obtient un vecteur de longueur $n-e$ pour lequel l'observation de $t$ parts parmi les parts $\left(\sum_{i=1}^{e+1} w_i, w_{e+2}, \ldots, w_n\right)$ ne révèle pas d'information sur $s$.
De même, comme le vecteur de reconstruction $\widehat{\lambda}$ est public, l'observation de $t+e$ parts parmi les parts $\widehat{\lambda}_1 w_1, \ldots, \widehat{\lambda}_n w_n$ n'apporte aucune information sur le secret partagé $s$. On en déduit qu'en regroupant $e+1$ parts entre elles, l'observation de $t$ parts parmi les parts $\left(\sum_{i=1}^{e+1} \widehat{\lambda}_i w_i, \widehat{\lambda}_{e+2} w_{e+2}, \ldots, \widehat{\lambda}_n w_n\right)$ ne révèle pas d'information sur $s$.
Par conséquent, $e+1$ parts d'un vecteur de partage $v_{ss'} \in E_{ss'}(\widehat{\mathcal{C}})$ peuvent être additionnées entre elles tout en conservant le paramètre de sécurité attendu $t$ du schéma fourni par le code $\mathcal{C}$.

□

Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp, \widehat{d} \geqslant 2$ et $\widehat{d^\perp} > d^\perp$. Afin d'améliorer la procédure de multiplication décrite par l'algorithme 16, on propose d'appliquer le lemme 3.4.5 avant d'effectuer la procédure de transcodage. Cependant, ce lemme s'applique sur un vecteur de partage associé au code $\widehat{\mathcal{C}}$ qui doit être uniformément distribué. Pour assurer cette condition, on propose de tirer profit du lemme suivant sur

le vecteur de partage associé à $\widehat{\mathcal{C}}$ résultant du produit de Schur entre deux vecteurs de partage associé au code $\mathcal{C}$.

**Lemme 3.4.6.** *Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$. L'addition entre un vecteur de partage $v_s \in E_s(\mathcal{C})$ et un vecteur issu de la procédure de partage appliquée à 0 retourne un vecteur uniformément distribué sur $E_s(\mathcal{C})$.*

*Démonstration.* Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$. Pour prouver le lemme précédent, il suffit de montrer la bijectivité de l'application affine suivante, pour tout $s \in S = \mathbb{F}_q$ et tout vecteur $v_s = (c_1, \ldots, c_n) \in E_s(\mathcal{C})$ :

$$\begin{array}{rccc} \varphi_{vs} : & E_0(\mathcal{C}) & \longrightarrow & E_s(\mathcal{C}) \\ & v_0 & \longmapsto & v_s + v_0 \end{array} \tag{3.52}$$

La bijectivité de cette application se déduit directement en exhibant l'application réciproque de $\varphi_{vs}$ :

$$\begin{array}{rccc} (\varphi_{vs})^{-1} : & E_s(\mathcal{C}) & \longrightarrow & E_0(\mathcal{C}) \\ & y & \longmapsto & y - v_s \end{array} \tag{3.53}$$

Par conséquent, pour tout $s \in \mathbb{F}_q$ et pour tout vecteur $v_s \in E_s(\mathcal{C})$, l'application $\varphi_{vs}$ est bijective, ce qui assure que pour tout vecteur $v_0 \in_R E_0(\mathcal{C}) : (v_0 + v_s) \in_R E_s(\mathcal{C})$.

□

Par les lemmes 3.4.5 et 3.4.6, on peut alors en déduire le théorème suivant :

**Théoreme 3.4.7** (Procédure de multiplication sécurisée améliorée)**.** *Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$. Supposons que $\widehat{d} \geqslant 2$, $\widehat{d}^\perp > d^\perp$. La procédure de multiplication entre deux vecteurs de partage décrite par l'algorithme 17 est résistante aux attaques d'ordre $t = d^\perp - 2$.*

**Remarque 3.4.8.** *Le choix des parts à additionner entre elles est arbitraire. Par souci de clarté, on additionnera toujours les $e+1$ premières parts dans la suite.*

**Notation 3.4.9.** *Dans l'algorithme 17, afin de pouvoir appliquer la procédure de transcodage sur le vecteur $(\sum_{i=1}^{e+1} \widehat{\lambda}_i w_i, \widehat{\lambda}_{e+2} w_{e+2}, \ldots, \widehat{\lambda}_n w_n)$, on dira que ce vecteur est associé à un schéma de partage de secret construit à partir du code issu de $\widehat{\mathcal{C}}$ de longueur $n-e+1$. Dans la suite, on notera ce code $\widehat{\mathcal{C}}^\star$. De plus, par construction, on peut prendre comme vecteur de reconstruction de $\widehat{\mathcal{C}}^\star$ le vecteur $\mathbf{1}_{n-e}$.*

**Algorithme 17** Amélioration de la procédure de multiplication standard

ENTRÉES: Soient $\mathcal{C}$ et $\widehat{\mathcal{C}}$ deux codes linéaires de paramètres respectifs $[n+1,k,d]_q$ et $[n+1,\widehat{k},\widehat{d}]_q$ vérifiant $d, d^{\perp} \geqslant 2$, $\widehat{d} \geqslant 2$ et $\widehat{d}^{\perp} \geqslant d^{\perp}$. Notons $t$ et $\widehat{t}$ les paramètres de sécurité respectifs de $\mathcal{C}$ et $\widehat{\mathcal{C}}$ vérifiant : $e = \widehat{t} - t > 0$ et $\widehat{\mathcal{C}}^{\star}$ le code de longueur $n - e + 1$ déduit de $\widehat{\mathcal{C}}$ (cf. notation 3.4.9)
Soient $(c_1, \ldots, c_n) \in E_s(\mathcal{C})$, $(c'_1, \ldots, c'_n) \in E_{s'}(\mathcal{C})$ et $\widehat{\lambda}$ un vecteur de reconstruction associé à $\widehat{\mathcal{C}}$
SORTIE: $(z_1, \ldots, z_n) \in E_{ss'}(\mathcal{C})$

**Fonction :** $\mathsf{SecMult2}((c_1, \ldots, c_n), (c'_1, \ldots, c'_n))$
1. $(y_1, \ldots, y_n) \leftarrow (c_1 c'_1, \ldots, c_n c'_n)$
2. $(w_1, \ldots, w_n) \leftarrow (y_1, \ldots, y_n) + \mathsf{partage}_{\widehat{\mathcal{C}}}(0)$
3. $(w_{e+1}, \ldots, w_n) \leftarrow \left(\sum_{i=1}^{e+1} \widehat{\lambda}_i w_i, \widehat{\lambda}_{e+2} w_{e+2}, \ldots, \widehat{\lambda}_n w_n\right)$
4. $(z_1, \ldots, z_n) \leftarrow \mathsf{transcodage}_{(\widehat{\mathcal{C}}^{\star} \mapsto \mathcal{C})}(\mathbf{1}_{n-e}, (w_{e+1}, \ldots, w_n))$
5. **Retourner** $(z_1, \ldots, z_n)$

### 3.4.3 Applications

Dans cette sous-section, on étudie l'application de la procédure de multiplication fournie par les codes linéaires et son amélioration lorsque l'on considère les exemples de schémas de partage de secret cités dans la sous-section 3.3.3.

#### 3.4.3.1 Schéma de partage de secret basique

**Lemme 3.4.10.** *Soit $\mathbb{F}_q$ le corps fini à $q$ éléments de caractéristique $p$. Si $\mathcal{C}$ est le code de parité de paramètres $[t+2, t+1]_q$ (avec $t \geqslant 1$), alors $\widehat{\mathcal{C}}$ ne permet pas de construire un schéma de partage de secret linéaire comme décrit par le théorème 3.3.3.*

*Démonstration.* Soit $\mathcal{C}$ le code de parité de paramètres $[t+2, t+1, 2]_q$. Pour tout $i = 0, \ldots, t+1$, notons $e_i$ le vecteur de la base canonique $\mathbb{F}_q^{t+2}$ ayant un 1 en position $i$. Montrons que pour tout $i$ : $e_i \in \widehat{\mathcal{C}}$.
D'après la matrice génératrice de $\mathcal{C}$ (cf. matrice (3.20)), il est facile de voir que pour tout $i, j = 0, \ldots, t+1$, les vecteurs $e_{i,j} = e_i + e_j$ sont des mots de code de $\mathcal{C}$. Par construction, on a $e_i = e_{i,j} * e_{i,k}$, pour tout $i, j, k = 0, \ldots, t+1$ tels que $i \neq j \neq k$. Donc $e_i \in \widehat{\mathcal{C}}$, pour tout $i = 0, \ldots, t+1$. D'où $\widehat{\mathcal{C}} = \mathbb{F}_q^{n+1}$. Par conséquent le code $\widehat{\mathcal{C}}$ ne permet de construire un schéma de partage de secret comme décrit par le théorème 3.3.3.

□

D'après le lemme 3.4.10, on sait que le code $\widehat{\mathcal{C}}$ associé au schéma de partage de secret basique construit à partir du code de parité $\mathcal{C}$ de paramètres $[t+2, t+1]_q$ (avec $t \geqslant 1$) ne permet pas de construire un schéma de partage de secret linéaire comme décrit par le théorème 3.3.3. Par conséquent la procédure de multiplication fournie par la structure des codes linéaires ne peut pas être appliquée. Néanmoins d'autres méthodes de multiplication

ont été proposées dans la littérature n'utilisant pas les propriétés des codes linéaires (cf. chapitre 2).

#### 3.4.3.2 Schéma de partage de secret de Shamir

**Lemme 3.4.11.** *Soit $\mathbb{F}_q$ le corps fini à $q$ éléments de caractéristique $p$. Considérons un schéma de partage de secret de paramètre de sécurité $t \geqslant 0$ construit à partir du code de Reed-Solomon $\mathcal{C}$ de paramètres $[n+1, t+1]_q$ avec $n \leqslant q$ et $d, d^\perp \geqslant 2$. Lorsque $n \geqslant 2t+1$, alors le code $\widehat{\mathcal{C}}$ est un code de Reed-Solomon de paramètres $[n+1, 2t+1]_q$ qui permet de construire un schéma de partage de secret de paramètre de sécurité $\widehat{t} = 2t$.*

*Démonstration.* Soit $\mathcal{C}$ le code de Reed-Solomon de paramètres $[n+1, t+1]_q$ avec $n \leqslant q$ ; $n \geqslant 2t+1$ et $d, d^\perp \geqslant 2$. Considérons $x_0, \ldots, x_n$ des éléments de $\mathbb{F}_q$ tous distincts deux à deux. Soient deux mots de code $(f(x_0), \ldots, f(x_n)) \in \mathcal{C}$ et $(g(x_0), \ldots, g(x_n)) \in \mathcal{C}$, avec $f$ et $g$ deux polynômes de degré $t$. Le produit $fg$ retourne un polynôme de degré $2t$ :

$$f(x)g(x) = \sum_{i=0}^{2t} c_i x^i \ . \tag{3.54}$$

On a alors par construction de $\widehat{\mathcal{C}}$ que $(f(x_0)g(x_0), \ldots, f(x_n)g(x_n)) \in \widehat{\mathcal{C}}$ correspond à l'évaluation du polynôme $fg$ de degré $2t$ en les $n+1 \geqslant 2t+1$ points : $x_0, \ldots, x_n$. Le code $\widehat{\mathcal{C}}$ est donc inclus dans le code de Reed-Solomon de paramètres $[n+1, 2t+1]_q$.
De plus, on a l'inclusion inverse. En effet, chaque vecteur de base $(x_0^i, \ldots, x_n^i)$ du code de Reed-Solomon de paramètres $[n+1, 2t+1]_q$, pour $i = 0, \ldots, 2t$, s'écrit comme le produit de Schur de deux vecteurs de base de $\mathcal{C}$ :

$$\begin{array}{rcl} (x_0^i, \ldots, x_n^i) & = & \underbrace{(1, \ldots, 1)}_{\in \mathcal{C}} \ * \ \underbrace{(x_0^i, \ldots, x_n^i)}_{\in \mathcal{C}} \ , \text{ si } i \leqslant t \\ (x_0^i, \ldots, x_n^i) & = & \underbrace{(x_0^t, \ldots, x_n^t)}_{\in \mathcal{C}} \ * \ \underbrace{(x_0^{i-t}, \ldots, x_n^{i-t})}_{\in \mathcal{C}} \ , \text{ si } i > t \end{array} \tag{3.55}$$

Par conséquent, le code $\widehat{\mathcal{C}}$ est donc un code de Reed-Solomon de paramètres $[n+1, 2t+1]_q$ qui permet de construire un schéma de partage de secret de paramètre de sécurité $\widehat{t} = \widehat{d}^\perp - 2 = 2t$.

□

Soit $\mathcal{C}$ le code de Reed-Solomon de paramètres $[n+1, t+1]_q$ avec $n \leqslant q$. Lorsque $n \geqslant 2t+1$, le code $\mathcal{C}$ permet de déterminer un schéma de partage de secret de paramètre de sécurité $t$ comme défini par le théorème 3.3.3. De plus, d'après le lemme précédent, on peut en déduire que $\widehat{\mathcal{C}}$ permet quant à lui de déterminer un schéma de partage de secret de paramètre de sécurité $2t$. Les procédures de multiplication sécurisées décrites par les

algorithmes 16 et 17 sont donc applicables.

Par ailleurs, l'application de la procédure de multiplication standard décrite par l'algorithme 16 est une généralisation de celle décrite initialement dans [BOGW88, CCD88] pour le schéma de partage de secret de Shamir. De plus, on peut également écrire en termes de code correcteurs l'amélioration proposée par Louis Goubin et Ange Martinelli dans [GM11] rappelée dans le chapitre 2 par l'algorihme 10. Ce qui donne l'algorithme 18.

**Algorithme 18** Multiplication de Goubin et Martinelli [GM11, Adaptation de l'algorithme 2]

ENTRÉES: Soient $\mathcal{C}$ et $\widehat{\mathcal{C}}$ les codes de Reed-Solomon de paramètres respectifs $[2t+2, t+1]_q$ et $[2t+2, 2t+1]_q$ ; $\mathcal{C}'$ le code de Reed-Solomon de paramètres $[t+2, t+1]_q$ obtenu en tronquant $\mathcal{C}$
Soient $t+1$ éléments publics distincts non nuls : $x_{t+2}, \ldots, x_{2t+1}$, $\beta_j(x_i)$ des éléments pré-calculés pour $1 \leqslant j \leqslant t+1$ et $t+2 \leqslant i \leqslant 2t+1$
Soient $(c_1, \ldots, c_{t+1}) \in E_s(\mathcal{C}')$, $(c'_1, \ldots, c'_{t+1}) \in E_{s'}(\mathcal{C}')$ et $\widehat{\lambda}$ un vecteur de reconstruction de $\widehat{\mathcal{C}}$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{ss'}(\mathcal{C}')$

*Génération à la volée des t parts supplémentaires :*

1. $(c_1, \ldots, c_{t+1}, c_{t+2}, \ldots, c_{2t+1}) \leftarrow \left(c_1, \ldots, c_{t+1}, \sum_{j=1}^{t+1} c_j\beta_j(x_{t+2}), \ldots, \sum_{j=1}^{t+1} c_j\beta_j(x_{2t+1})\right)$
2. $\left(c'_1, \ldots, c'_{t+1}, c'_{t+2}, \ldots, c'_{2t+1}\right) \leftarrow \left(c'_1, \ldots, c'_{t+1}, \sum_{j=1}^{t+1} c'_j\beta_j(x_{t+2}), \ldots, \sum_{j=1}^{t+1} c'_j\beta_j(x_{2t+1})\right)$

*Procédure de multiplication :*

3. $(w_1, \ldots, w_{2t+1}) \leftarrow \left(c_1c'_1, \ldots, c_{2t+1}c'_{2t+1}\right)$
4. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{transcodage}_{(\widehat{\mathcal{C}} \mapsto \mathcal{C}')}\left(\widehat{\lambda}, (w_1, \ldots, w_{2t+1})\right)$
5. **Retourner** $(z_1, \ldots, z_{t+1})$

De plus, notre procédure de multiplication améliorée présentée dans la sous-section 3.4.2 permet d'améliorer l'algorithme précédent. En effet, considérons $\mathcal{C}$, $\widehat{\mathcal{C}}$ les codes de Reed-Solomon de paramètres respectifs $[2t+2, t+1]_q$ et $[2t+2, 2t+1]_q$. Ces codes étant des codes MDS, on a alors :

$$\begin{cases} d^\perp &= t+2 \\ \widehat{d}^\perp &= 2t+2 \end{cases} \tag{3.56}$$

D'où $e = \widehat{d}^\perp - d^\perp = t$. Comme à l'étape 3 de l'algorithme 18, $2t+1$ parts associées à $\widehat{\mathcal{C}}$ sont manipulées, on a $2t+1-e = t+1$. Par conséquent, notre amélioration peut alors être appliquée nécessitant seulement $t+1$ procédures de partage lors de la procédure de transcodage au lieu de $2t+1$. L'algorithme 19 présente le traitement de la procédure de multiplication prenant en compte l'amélioration de la section 3.4.2 et celle proposée dans [GM11] (Alg. 18).

**Algorithme 19** Adaptation de l'algorithme 17 pour le schéma de partage de secret de Shamir

ENTRÉES: Soient $\mathcal{C}$ et $\widehat{\mathcal{C}}$ les codes de Reed-Solomon de paramètres respectifs $[2t+2, t+1]_q$ et $[2t+2, 2t+1]_q$ ; $\mathcal{C}'$ le code de Reed-Solomon de paramètres $[t+2, t+1]_q$ obtenu en tronquant $\mathcal{C}$ ; $t+1$ éléments publics distincts non nuls : $x_{t+2}, \ldots, x_{2t+1}$ ; $\beta_j(x_i)$ des éléments pré-calculés pour $1 \leqslant j \leqslant t+1$ et $t+2 \leqslant i \leqslant 2t+1$
Soient $(c_1, \ldots, c_{t+1}) \in E_s(\mathcal{C}')$, $(c'_1, \ldots, c'_{t+1}) \in E_{s'}(\mathcal{C}')$ et $\widehat{\lambda}$ un vecteur de reconstruction de $\widehat{\mathcal{C}}$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{ss'}(\mathcal{C}')$

*Génération à la volée des t parts supplémentaires :*

1. $(c_1, \ldots, c_{t+1}, c_{t+2}, \ldots, c_{2t+1}) \leftarrow \left(c_1, \ldots, c_{t+1}, \sum_{j=1}^{t+1} c_j \beta_j(x_{t+2}), \ldots, \sum_{j=1}^{t+1} c_j \beta_j(x_{2t+1})\right)$
2. $\left(c'_1, \ldots, c'_{t+1}, c'_{t+2}, \ldots, c'_{2t+1}\right) \leftarrow \left(c'_1, \ldots, c'_{t+1}, \sum_{j=1}^{t+1} c'_j \beta_j(x_{t+2}), \ldots, \sum_{j=1}^{t+1} c'_j \beta_j(x_{2t+1})\right)$

*Procédure de multiplication améliorée :*

3. $(w_1, \ldots, w_{2t+1}) \leftarrow \left(c_1 c'_1, \ldots, c_{2t+1} c'_{2t+1}\right) + \mathsf{partage}_{\widehat{\mathcal{C}}}(0)$
4. $(w_{t+1}, \ldots, w_{2t+1}) \leftarrow \left(\sum_{i=1}^{t+1} \widehat{\lambda}_i w_i, \widehat{\lambda}_{t+2} w_{t+2}, \ldots, \widehat{\lambda}_{2t+1} w_{2t+1}\right)$
5. $(z_1, \ldots, z_{t+1}) \leftarrow \mathrm{transcodage}_{(\widehat{\mathcal{C}} \hookrightarrow \mathcal{C}')}\left(\mathbf{1}_{t+1}, (w_{t+1}, \ldots, w_{2t+1})\right)$
6. **Retourner** $(z_1, \ldots, z_{t+1})$

| | aléa | add | éval polynômes | mult |
|---|---|---|---|---|
| Procédure standard : Alg. 16 ([BOGW88, CCD88]) | $2t^2+t$ | $4t^2+2t$ | $2t+1$ (en $2t+1$ pts) $\Rightarrow \begin{cases} 4t^3+4t^2+t \text{ add} \\ 4t^3+4t^2+t \text{ mult} \end{cases}$ | $4t+2$ |
| Première amélioration : Alg. 10 ou Alg. 18 ([GM11]) | $2t^2+t$ | $4t^2+2t$ | $2t+1$ (en $t+1$ pts) $\Rightarrow \begin{cases} 2t^3+3t^2+t \text{ add} \\ 2t^3+3t^2+t \text{ mult} \end{cases}$ | $2t^2+6t+2$ |
| Seconde amélioration : Alg. 19 | $t^2+3t$ | $3t^2+4t+1$ | $1$ (en $2t+1$ pts) $\Rightarrow \begin{cases} 4t^2-1 \text{ add} \\ 4t^2+2t \text{ mult} \end{cases}$<br>$t+1$ (en $t+1$ pts) $\Rightarrow \begin{cases} t^3+2t^2+t \text{ add} \\ t^3+2t^2+t \text{ mult} \end{cases}$ | $2t^2+6t+2$ |

TABLE 3.1 – Coût des procédures de multiplication sur $\mathbb{F}_q$ résistantes aux attaques d'ordre $t$ appliquées pour le schéma de partage de secret de Shamir

Le coût des trois procédures de multiplication cités pour le schéma de partage de secret de Shamir est décrit dans la table 3.1. De plus, afin de comparer l'efficacité de ces trois procédures, on donne une application numérique de cette première table pour $t = 1, \ldots, 6$ (cf. table 3.2).

D'après les tables 3.1 et 3.2, on peut constater que l'algorithme décrit par Louis Goubin et Ange Martinelli (Alg. 18), et l'amélioration que l'on propose (Alg. 19) sont plus efficaces que la méthode dite standard de [BOGW88, CCD88] (Alg. 16) dès $t = 2$. De plus, notre procédure de multiplication est nettement plus efficace que celle de Louis Goubin et Ange Martinelli dès $t = 4$.

| | Procédure standard Alg. 16 ( [BOGW88, CCD88]) | Première amélioration Alg. 10 ou 18 ([GM11]) | Seconde amélioration Alg. 19 (Nous) |
|---|---|---|---|
| t=1 | **3 aléa**<br>**15 add**<br>**15 mult** | 3 aléa<br>12 add<br>16 mult | 4 aléa<br>15 add<br>20 mult |
| t=2 | 10 aléa<br>70 add<br>60 mult | **10 aléa**<br>**50 add**<br>**52 mult** | 10 aléa<br>54 add<br>60 mult |
| t=3 | 21 aléa<br>189 add<br>161 mult | **21 aléa**<br>**126 add**<br>**122 mult** | **18 aléa**<br>**123 add**<br>**128 mult** |
| t=4 | 36 aléa<br>396 add<br>342 mult | 36 aléa<br>252 add<br>238 mult | **28 aléa**<br>**228 add**<br>**230 mult** |
| t=5 | 55 aléa<br>715 add<br>627 mult | 55 aléa<br>440 add<br>412 mult | **40 aléa**<br>**375 add**<br>**372 mult** |
| t=6 | 78 aléa<br>1170 add<br>1040 mult | 78 aléa<br>702 add<br>656 mult | **54 aléa**<br>**570 add**<br>**560 mult** |

TABLE 3.2 – Application numérique des procédures de multiplication sur $\mathbb{F}_q$ résistantes aux attaques d'ordre $t = 1, \ldots, 6$ pour le schéma de partage de secret de Shamir

## 3.5 Procédure d'élévation à la puissance $p$

### 3.5.1 Description générale

Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$ vérifiant $d, d^\perp \geqslant 2$. La procédure d'élévation à la puissance $p$ d'un secret $s$ à partir d'un vecteur de partage $v_s \in E_s(\mathcal{C})$ peut être réalisée en appliquant $p$ fois une procédure de multiplication lorsque le code $\widehat{\mathcal{C}}$ vérifie $\widehat{d}, \widehat{d}^\perp \geqslant 2$.
Cependant, considérons le code suivant :

$$\mathcal{C}_p = \{c^p = \underbrace{c * \ldots * c}_{p \text{ fois}} : c \in \mathcal{C}\} \ . \tag{3.57}$$

**Notations 3.5.1.** *Dans la suite, on notera $d_p$ et $d_p^\perp$ respectivement les distances minimales de $\mathcal{C}_p$ et de son dual noté $\mathcal{C}_p^\perp$.*

Un tel code vérifie le lemme suivant :

**Lemme 3.5.2.** *Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$, avec $d, d^\perp \geqslant 2$. Le code $\mathcal{C}_p$ défini par la relation* (3.57) *est isomorphe au code $\mathcal{C}$ et le code $\mathcal{C}_p^\perp$ est isomorphe au code $\mathcal{C}^\perp$.*

*Démonstration.* Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Considérons $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$, avec $d, d^\perp \geqslant 2$. Afin de prouver que le code $\mathcal{C}_p$ est isomorphe au code $\mathcal{C}$, montrons que l'application suivante est bijective :

$$\begin{array}{rccc} \varphi : & \mathcal{C} & \longrightarrow & \mathcal{C}_p \\ & c & \longmapsto & c^p \end{array} \tag{3.58}$$

Par construction de $\mathcal{C}_p$, on peut en déduire directement la surjectivité de l'application $\varphi$. À présent montrons que $\varphi$ est injective. Par l'application de Frobenius, on a pour tout $c, c' \in \mathcal{C} : (c + c')^p = c^p + c'^p$. L'application $\varphi$ est donc une application linéaire. Montrons alors que $\varphi$ est injective en déterminant son noyau. Considérons un mot de code $c = (c_0, \ldots, c_n) \in \mathcal{C}$ et supposons que $\varphi(c) = (0, \ldots, 0)$. Ceci implique que $c_i^p = 0$, pour tout $i = 0, \ldots, n$. Or $\mathcal{C}_p$ étant défini sur le corps $\mathbb{F}_q$ de caractéristique $p$, on a par définition :

$$\forall \alpha \in \mathbb{F}_q : \ \alpha = 0 \Leftrightarrow \alpha^p = 0 \ . \tag{3.59}$$

Donc $c_i = 0$, pour tout $0 = 1, \ldots, n$. D'où $\mathrm{Ker}(\varphi) = (0, \ldots, 0)$. L'application $\varphi$ est donc injective. Par ailleurs, par la relation (3.59), on peut en déduire que $d_p = d$. Il s'en suit que :

$$\mathcal{C}_p \simeq \mathcal{C} \ . \tag{3.60}$$

Montrons à présent que $\mathcal{C}_p^\perp \simeq \mathcal{C}^\perp$. En appliquant la relation (3.60) au dual de $\mathcal{C}$, on peut en déduire directement :

$$(\mathcal{C}^\perp)_p \simeq \mathcal{C}^\perp \ . \tag{3.61}$$

Montrons alors que les deux codes suivants :

$$\begin{cases} (\mathcal{C}^{\perp})_p &= \{y^p : y \in \mathcal{C}^{\perp}\} \\ \mathcal{C}_p^{\perp} &= (\mathcal{C}_p)^{\perp} = \{y \in \mathbb{F}_q^{n+1} : \forall x \in \mathcal{C}_p, \langle y, x \rangle = 0\} \end{cases} \tag{3.62}$$

constituent le même code.
Soit $h$ un mot de code de $\mathcal{C}^{\perp}$. Par définition, on a pour tout mot de code $c \in \mathcal{C} : \langle h, c \rangle = 0$. Par l'application de Frobenius, on obtient $\langle h^p, c^p \rangle = 0$. D'où $h^p \in (\mathcal{C}^{\perp})_p$. D'après la définition de $(\mathcal{C}_p)^{\perp}$, on peut alors en déduire que :

$$(\mathcal{C}^{\perp})_p \subseteq (\mathcal{C}_p)^{\perp} \ . \tag{3.63}$$

De plus, d'après les relations (3.60) et (3.61), on peut en déduire que :

$$\begin{array}{rcl} \dim\ (\mathcal{C}_p)^{\perp} &=& \dim\ \mathcal{C}^{\perp} \\ \dim\ (\mathcal{C}^{\perp})_p &=& \dim\ \mathcal{C}^{\perp} \end{array} \tag{3.64}$$

où dim (.) désigne la dimension du code considéré.
Par conséquent, on peut en déduire que $\mathcal{C}_p^{\perp} = (\mathcal{C}_p)^{\perp} = (\mathcal{C}^{\perp})_p$.
Par ailleurs, par le même raisonnement entre $\mathcal{C}$ et $\mathcal{C}_p$, on peut en déduire que la distance minimale du code $\mathcal{C}_p^{\perp}$ est égale à celle de $\mathcal{C}^{\perp}$.

□

En exploitant le lemme 3.5.2, la procédure d'élévation à la puissance $p$ peut alors s'effectuer plus efficacement qu'en effectuant $p$ procédures de multiplication. En effet, considérons un secret $s \in S$ et un vecteur de partage $v_s \in E_s(\mathcal{C})$. Dans ce cas, on peut obtenir un vecteur $v_{s^p} \in E_{s^p}(\mathcal{C})$ :

- en mettant chacune des coordonnées du vecteur de partage $v_s$ à la puissance $p$,
- puis en appliquant la procédure de transcodage de $\mathcal{C}_p$ vers $\mathcal{C}$.

Cette procédure est décrite par l'algorithme 20.

**Algorithme 20** Procédure d'élévation à la puissance $p$ d'un vecteur de partage

ENTRÉES: Soit $\mathbb{F}_q$ un corps de caractéristique $p$.
Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k]_q$ vérifiant $d, d^{\perp} \geqslant 2$
Soient $(c_1, \ldots, c_n) \in E_s(\mathcal{C})$ et $\lambda$ un vecteur de reconstruction associé à $\mathcal{C}$
SORTIE: $(z_1, \ldots, z_n) \in E_{s^p}(\mathcal{C})$

**Fonction :** $\mathsf{puissance_p}(s)$
1. $(w_1, \ldots, w_n) \leftarrow (c_1^p, \ldots, c_n^p)$
2. $(z_1, \ldots, z_n) \leftarrow \mathsf{transcodage}_{(\mathcal{C}_p \mapsto \mathcal{C})}(\lambda^p, (w_1, \ldots, w_n))$
3. **Retourner** $(z_1, \ldots, z_n)$

**Lemme 3.5.3** (Procédure d'élévation à la puissance $p$ sécurisée)**.** *Soit $\mathbb{F}_q$ un corps de caractéristique $p$. Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$. La procédure d'élévation à la puissance $p$ décrite par l'algorithme 20 est résistante aux attaques d'ordre $t = d^\perp - 2$.*

*Démonstration.* Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$, de paramètre de sécurité $t$. D'après le lemme 3.5.2, le code $\mathcal{C}_p$ vérifie les conditions suivantes : $d_p = d$ et $d_p^\perp = d^\perp$. Les conditions attendues par la procédure sécurisée de transcodage de $\mathcal{C}_p$ vers $\mathcal{C}$ sont remplies. Par conséquent, la procédure d'élévation à la puissance $p$ décrite par l'algorithme 20 est résistante aux attaques d'ordre $t$.

□

**Remarque 3.5.4.** *Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$. On peut montrer aisément, pour tout $m \in \mathbb{N}^*$, que la procédure d'élévation à la puissance $p^m$, donnée par l'algorithme 20 en remplaçant $p$ par $p^m$, est résistante aux attaques d'ordre $t = d^\perp - 2$.*

### 3.5.2 Cas particulier

Considérons le schéma de partage de secret construit à partir d'un code linéaire $\mathcal{C}$ de paramètres $[n+1,k,d]_q$ vérifiant $d, d^\perp \geqslant 2$. Supposons de plus que le schéma de partage de secret est construit à partir d'un code linéaire $\mathcal{C}$ dont les coefficients de sa matrice génératrice sont définis dans $\mathbb{F}_p$. Par le lemme suivant, on en déduit que la procédure d'élévation à la puissance $p$ est une transformation stable, ne nécessitant donc pas de procédure de transcodage, *i.e.,* :

$$\forall (c_1, \ldots, c_n) \in E_s(\mathcal{C}) : (c_1^p, \ldots, c_n^p) \in E_{s^p}(\mathcal{C}) \ . \tag{3.65}$$

**Lemme 3.5.5.** *Soit $\mathcal{C}$ un code linéaire défini sur un corps fini $\mathbb{F}_q$ de caractéristique $p$. Les assertions suivantes sont équivalentes :*

1. *$\mathcal{C}_p = \mathcal{C}$.*
2. *$\mathcal{C}$ est un code linéaire caractérisé par une matrice génératrice ayant ses coefficients définis dans $\mathbb{F}_p$.*

*Démonstration.* Soit $\mathbb{F}_q$ un corps fini de caractéristique $p$. Soit $\mathcal{C}$ un code linéaire de paramètres $[n,k]_q$ ayant une base définie dans $\mathbb{F}_p$ : $b_1, \ldots, b_k$ avec $b_i \in \mathbb{F}_p^n$ pour $i \in \{1, \ldots, k\}$, c'est-à-dire pour tout $i$, $b_i^p = b_i$.
Si $c \in \mathcal{C}$, alors il existe une combinaison linéaire telle que $c = \sum_{i=1}^k h_i b_i$ avec $h_i \in \mathbb{F}_q$. On a alors $c^p = \sum_{i=1}^k h_i^p b_i^p = \sum_{i=1}^k h_i^p b_i \in \mathcal{C}$.

Inversement, considérons un code linéaire $\mathcal{C}$ de paramètres $[n,k]_q$ tel que pour tout mot de code $c \in \mathcal{C}$, $c^p \in \mathcal{C}$. Considérons $G$ une matrice génératrice de $\mathcal{C}$ sous forme systématique. Notons $b_i$ la $i$-ième ligne de $G$, on peut écrire :

$$b_i = (0, \ldots, 0, 1, 0, \ldots, 0, a_1, \ldots, a_{n-k}) \ , \tag{3.66}$$

avec $a_1, \ldots, a_{n-k} \in \mathbb{F}_q$.
Par hypothèse $b_i^p = (0, \ldots, 0, 1, 0, \ldots, 0, a_1^p, \ldots, a_{n-k}^p) \in \mathcal{C}$, il existe donc une combinaison linéaire telle que $b_i^p = \sum_{j=1}^{k} h_j b_j$. À partir de la dernière égalité, la j-ème coordonnée de $b_i^p$ est égale à $h_j$ pour $j \in \{1, \ldots, k\}$. Par identification, on a pour $1 \leqslant j \leqslant k$ et $j \neq i$ : $h_j = 0$ et $h_i = 1$. Par conséquent $b_i^p = b_i$, et donc $a_j^p = a_j$ pour $j \in \{1, \ldots, k\}$. Ainsi $b_i \in \mathbb{F}_p^n$.

□

### 3.5.3 Applications

Dans cette sous-section, on étudie l'application de la procédure d'élévation à la puissance $p^m$, avec $m \in \mathbb{N}^*$, lorsque l'on considère les exemples de schémas de partage de secret cités dans la sous-section 3.3.3.1. Plus précisément, on considère le schéma de partage de secret basique et celui de Shamir définis sur $\mathbb{F}_q$ de caractéristique $p$.

#### 3.5.3.1 Schéma de partage de secret basique

Pour construire le schéma de partage de secret basique de paramètre de sécurité $t$, considérons le code de parité de paramètre $[t+2, t+1, 2]_q$ (avec $t + 2 \geqslant 2$) de matrice génératrice $G$ sous forme systématique (cf. sous-section 3.3.3.1).
D'après le lemme 3.5.3, la procédure d'élévation à la puissance $p = 2$ décrite par l'algorithme 20 est applicable et résistante aux attaques d'ordre $t = d^\perp - 2$. De plus, comme la matrice génératrice d'un tel code a ses coefficients définis dans $\mathbb{F}_2$, on sait d'après le lemme 3.5.5 que la procédure d'élévation à la puissance 2 est une transformation stable ne nécessitant pas de transcodage :

$$\forall c \in \mathcal{C} \Rightarrow c^2 \in \mathcal{C} \ . \tag{3.67}$$

Ainsi, on a aussi : $\forall c^2 \in \mathcal{C} \Rightarrow c^4 \in \mathcal{C}$. Il s'en suit que pour tout $m \in \mathbb{N}^*$ :

$$\forall c \in \mathcal{C} \Rightarrow c^{2^m} \in \mathcal{C} \ . \tag{3.68}$$

Par conséquent, la procédure d'élévation à la puissance $2^m$, avec $m \in \mathbb{N}^*$, est une transformation stable.

#### 3.5.3.2 Schéma de partage de secret de Shamir

Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Pour construire le schéma de partage de secret de Shamir de paramètre de sécurité $t$, considérons le code de Reed-Solomon $\mathcal{C}$ de paramètres $[n+1, t+1]_q$ avec $n \leqslant q$ et $n \geqslant t+1$. D'après la remarque 3.5.4, la procédure d'élévation à la puissance $p^m$, pour tout $m \in \mathbb{N}^*$, décrite par l'algorithme 20 en remplaçant $p$ par $p^m$ est applicable et résistante aux attaques d'ordre $t$.

Par ailleurs pour $n = t + 1$, la procédure d'élévation à la puissance $p^m$ décrite par cet algorithme coïncide avec la solution proposée dans [GM11]. En effet, pour tout secret $s \in S$, chaque $v_s \in E_s(\mathcal{C})$ correspond à l'évaluation d'un polynôme de degré $t$ en $t+1$ points distincts et fixés : $x_1, \ldots, x_{t+1}$. Considérons un vecteur de partage $v_s = (f(x_1), \ldots, f(x_{t+1}))$, où $f$ un polynôme de degré $t$, vérifiant :

$$s = \sum_{i=1}^{t+1} f(x_i)\beta_i \ . \tag{3.69}$$

Par l'application de Frobenius, on en déduit :

$$s^p = \left(\sum_{i=1}^{t+1} f(x_i)\beta_i\right)^p = \sum_{i=1}^{t+1} f(x_i)^p \beta_i^p = \sum_{i=1}^{t+1} f(x_i^p)\beta_i^p \ . \tag{3.70}$$

Le vecteur de partage $(f(x_1)^p, \ldots, f(x_{t+1})^p)$ correspond donc à l'évaluation du polynôme $f$ en $t+1$ points distincts : $x_1^p, \ldots, x_{t+1}^p$. Le code $\mathcal{C}_p$ est donc le code de Reed-Solomon de paramètres $[t+2, t+1]_q$ dont les mots de code sont constitués de l'ensemble des polynômes de degré $t$ évalués en les points : $x_1^p, \ldots, x_{t+1}^p$. La procédure d'élévation à la puissance $p$ requiert donc une procédure de transcodage afin de retourner un vecteur de partage associé au code $\mathcal{C}$, *i.e.,* à l'ensemble des polynômes de degré $t$ évalués en les points : $x_1, \ldots, x_{t+1}$. De la même façon, la procédure d'élévation à la puissance $p^m$ requiert une procédure de transcodage du code $\mathcal{C}_{p^m}$ vers le code $\mathcal{C}$.

Cependant, l'amélioration proposée dans la version longue de [PR11a] (cf. sous-section 2.4.2.3) qui permet de remplacer la procédure de transcodage par une simple permutation de parts reste applicable. Cette dernière requiert de choisir au préalable de la construction du schéma de partage de secret, des points $x_1, \ldots, x_{t+1}$ tels que l'ensemble de ces points soient stables par l'application de Frobenius $x \mapsto x^p$. Dans ce cas, une élévation à la puissance $p$ peut s'effectuer en appliquant une permutation sur les parts. En effet, comme explicité dans la sous-section 2.4.2.3, on peut considérer des points $x_1, \ldots, x_{t+1}$ tels que :

$$\forall i \in \{1, \ldots, t+1\}, \exists j \neq i \in \{1, \ldots, t+1\} \text{ tel que } x_i^p = x_j \ , \tag{3.71}$$

et ainsi, pour tout $(c_1, \ldots, c_{t+1}) \in E_s(\mathcal{C})$, on a $c_i^p = f(x_i)^p = g(x_i^p) = g(x_j) = c'_j$, avec $g$ un polynôme de degré $t$. D'où, en permuttant les parts, on obtient : $(c'_1, \ldots, c'_{t+1}) \in E_{s^p}(\mathcal{C})$. Il s'en suit que pour tout $m \in \mathbb{N}^*$, une procédure d'élévation à la puissance $p^m$ d'un secret requiert $t+1$ élévations à la puissance $p^m$ et un simple ordonnancement des parts au lieu d'une procédure de transcodage.

## 3.6 Coût des opérations masquées

Soit $\mathbb{F}_q$ une extension de $\mathbb{F}_p$. Soit $\mathcal{C}$ un code linéaire de paramètres $[n+1, k, d]_q$ avec $d, d^\perp \geqslant 2$. Supposons que $\widehat{\mathcal{C}}$ de paramètres $[n+1, \widehat{k}]_q$ vérifie $\widehat{d}, \widehat{d}^\perp \geqslant 2$. Notons $t$ et $\widehat{t}$ les

paramètres de sécurité respectifs de $\mathcal{C}$ et de $\widehat{\mathcal{C}}$ et supposons que l'élément $e = \widehat{t} - t$ soit strictement positif.
Dans cette section, on détermine le coût de chacune des opérations masquées, c'est-à-dire résistantes aux attaques d'ordre $t$, à savoir le coût de :

- la procédure de partage d'un secret dans $\mathcal{C}$
- la procédure de partage d'un secret dans $\widehat{\mathcal{C}}$
- la procédure d'addition entre un vecteur de partage associé à $\mathcal{C}$ et un élément de $\mathbb{F}_q$
- la procédure d'addition entre deux vecteurs de partage de $\mathcal{C}$
- la procédure de multiplication d'un vecteur de partage de $\mathcal{C}$ par un scalaire
- la procédure de multiplication donnée par l'algorithme 16
- la procédure de multiplication améliorée donnée par l'algorithme 17
- la procédure d'élévation à la puissance $p$ donnée par l'algorithme 20.

Dans la table 3.3, *aléa, additions, multiplications* réfèrent sur $\mathbb{F}_q$ respectivement aux nombres de variables uniformément distribués générées, d'additions et de multiplications requises. De plus $L$ et $L'$ indiquent respectivement le nombre d'additions et de multiplications nécessaires lors d'une procédure de partage qui dépendent de la matrice génératrice $G$ considérée. De même, $\widehat{L}$ et $\widehat{L}'$ indiquent respectivement le nombre d'additions et de multiplications nécessaires lors d'une procédure de partage de $\widehat{\mathcal{C}}$ dont la dimension est $\widehat{k}$. Enfin, on note $n_\lambda = \text{card}(\{\lambda_i \neq 0, 1 : i = 1, \ldots, n\})$.

| | | aléa | additions | multiplications |
|---|---|---|---|---|
| Partage d'un secret dans $\mathcal{C}$ : $\mathsf{partage}_{\mathcal{C}}$ | | $k-1$ | $L$ | $L'$ |
| Partage d'un secret dans $\widehat{\mathcal{C}}$ : $\mathsf{partage}_{\widehat{\mathcal{C}}}$ | | $\widehat{k}-1$ | $\widehat{L}$ | $\widehat{L}'$ |
| Addition d'un vecteur avec un élément de $\mathbb{F}_q$ | | $k-1$ | $n+L$ | $L'$ |
| Addition entre deux vecteurs de partage | | - | $n$ | - |
| Multiplication par un scalaire | | - | - | $n$ |
| Procédure de multiplication (Alg. 16) | | $(k-1)n$ | $n(L+n-1)$ | $n(1+L')+n_\lambda$ |
| Procédure de multiplication améliorée (Alg. 17) | | $\widehat{k}-1+(k-1)(n-e)$ | $\widehat{L}+(n-e)(L+n)-n$ | $\widehat{L}'+n+n_\lambda+L'(n-e)$ |
| Élévation à la puissance $p$ : | Stable | - $+n$ élévations à la puissance $p$ | - | - |
| | (Alg. 20) | $n(k-1)$ $+n$ élévations à la puissance $p$ | $n(L+n-1)$ | $n(L'+1)$ |

TABLE 3.3 – Coût des procédures résistantes aux attaques d'ordre $t$ à partir d'un schéma de partage de secret construit avec un code linéaire de paramètres $[n+1,k]_q$ de distance minimale duale $d^\perp = t+2$ et tel que $e = \widehat{d}^\perp - d^\perp \geqslant 0$

Cette table permet de déterminer le coût de chaque opération masquée pour tout code linéaire $\mathcal{C}$ vérifiant $d, d^\perp \geqslant 2$, $\widehat{d} \geqslant 2$ et $\widehat{d}^\perp \geqslant d^\perp$. En particulier, cette table sera exploitée dans le chapitre 4.

## 3.7 Conclusion

Dans ce chapitre, on a présenté la construction des schémas de partage de secret linéaires à partir des codes linéaires. En particulier, on a illustré l'équivalence existante entre les deux notions en présentant cette construction pour les schémas de partage de secret les plus connus, à savoir le schéma de partage de secret basique et le schéma de partage de secret de Shamir.

Afin d'utiliser une telle construction pour implanter des algorithmes résistants aux attaques d'ordre supérieur, on a décrit des procédures sécurisées permettant d'effectuer différentes opérations entre plusieurs secrets à partir des vecteurs de partage associés. En particulier, on a présenté la généralisation de la procédure de multiplication sécurisée fournie pour les codes linéaires. De plus, en tirant profit des propriétés des codes linéaires, on a proposé une amélioration de cette procédure qui s'avère à la fois efficace et résistante contre les attaques par canaux cachés d'ordre supérieur, notamment lorsque l'on considère le schéma de partage de secret de Shamir.

En pratique, on peut constater que seuls les schémas de partage à seuil, autrement dit basés sur des codes MDS, sont utilisés, alors que de nombreux schémas peuvent être construits en considérant les codes linéaires. On a vu dans le chapitre 2 que les schémas linéaires à seuil sont en effet adaptés pour les implantations nécessitant seulement des transformation linéaires pouvant être effectuées directement coordonnée par coordonnée. Cependant, ces mêmes schémas sont-ils les plus appropriés pour des transformations non linéaires telles que la multiplication entre secrets à partir des vecteurs de partage associés ? En étudiant les propriétés fournies par les codes linéaires, on pourrait concevoir de nouveaux schémas de partage de secret adaptés aux algorithmes à sécuriser. En particulier, en identifiant les codes pour lesquels la procédure de multiplication est applicable, on propose dans le chapitre 4, des schémas de partage de secret adaptés pour implanter un AES résistant aux attaques d'ordre supérieur.

# Chapitre 4

# Applications à l'AES

## Sommaire



## 4.1 Introduction

Dans ce chapitre, on s'intéresse à l'élaboration de nouveaux schémas de partage de secret pour implanter l'AES de manière sécurisée, *i.e.,* résistante contre les attaques d'ordre $t$. Comme la plupart des transformations requises sont linéaires, il est alors judicieux de choisir un schéma de partage de secret linéaire pour implanter ces transformations comme des transformations stables. À première vue, les schémas de partage de secret construits à partir de code MDS semblent être les plus pertinents car ils permettent de manipuler un nombre minimal de coordonnées lors des transformations stables. C'est d'ailleurs le cas des solutions présentées dans la sous-section 2.4.2. Cependant, on a vu dans la sous-section 2.4.2.1 que la solution proposée récemment dans [CPRR13] utilisant le schéma de partage de secret basique semble efficace mais qu'aucune preuve de sécurité ne garantit pour le moment la résistance de l'implantation de la transformation $SubBytes$. À l'inverse,

la solution proposée par [GM11, PR11b], basée sur l'utilisation du schéma de partage de secret de Shamir (cf. sous-section 2.4.2.3), est très coûteuse mais une preuve de sécurité peut être fournie. On peut alors se demander si l'utilisation de tels codes pour implanter la transformation *SubBytes* de manière sécurisée est la plus pertinente ?

En tirant profit des résultats présentés dans le chapitre 3, il serait intéressant de déterminer les propriétés que doivent vérifier les codes pour construire des contre-mesures permettant de garantir la sécurité de l'implantation de l'AES tout en restant la plus efficace possible. Dans cette optique, on propose, dans les sections 4.2 et 4.3, l'utilisation de deux schémas de partage de secret construits à partir de codes linéaires non MDS pour lesquels la procédure de multiplication fournie par les codes linéaires est applicable. En particulier, la première solution proposée est construite à partir d'un code d'évaluation proche de celui du code de Reed-Solomon tandis que la seconde considère la famille des codes auto-duaux et faiblement auto-duaux ayant leur matrice génératrice à coefficient sur $\mathbb{F}_2$ ou $\mathbb{F}_4$. De plus, l'efficacité de ces dernières est comparée dans la section 4.4 avec les méthodes présentées dans la sous-section 2.4.2.

## 4.2 Schéma de partage de secret non-idéal

Dans cette section, on propose une variante du schéma de partage de secret de Shamir pour implanter l'AES de manière résistante aux attaques d'ordre $t$. Cette solution est basée sur l'utilisation d'un schéma de partage de secret linéaire non-idéal construit à partir d'un code d'évaluation proche de celui du code de Reed-Solomon. Après avoir présenté ce schéma dans la sous-section 4.2.1, on décrit dans les sous-sections 4.2.2 et 4.2.3, l'implantation des principales transformations de l'AES.

### 4.2.1 Variante du schéma de partage de secret de Shamir

Comme rappelé dans la sous-section 2.4.2, le cryptosystème AES est défini sur le corps fini $\mathbb{F}_{256} \simeq \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. Une alternative est de considérer une autre représentation de ce corps en construisant une extension quadratique. Pour ce faire, considérons $U$ un polynôme irréductible de degré 4 et construisons le corps $\mathbb{F}_{2^4} \simeq \mathbb{F}_2[x]/U(x)$, que l'on note dans la suite $\mathbf{K}$. Soit $P$ est un polynôme irréductible de degré 2 défini dans $\mathbf{K}$. On construit alors le corps $\mathbb{F}_{2^8} \simeq \mathbf{K}[x]/P(x)$, que l'on note $\mathbf{L}$.
Soient $\theta$ une racine de $U$ dans $\mathbf{K}$ et $\omega$ une racine de $P$ dans $\mathbf{L}$. On en déduit que $(1, \theta, \theta^2, \theta^3)$ est une base de $\mathbf{K}$ sur $\mathbb{F}_2$ et $(1, \omega)$ une base de $\mathbf{L}$ sur $\mathbf{K}$. Par conséquent, on peut alors construire la base de $\mathbf{L}$ sur $\mathbb{F}_2$ suivante :

$$\mathcal{B} = (1, \theta, \theta^2, \theta^3, \omega, \omega\theta, \omega\theta^2, \omega\theta^3) \ . \tag{4.1}$$

D'après l'article [BLV13], on peut choisir par exemple les polynômes suivants :

$$\begin{array}{rcl} U(x) & = & x^4 + x + 1 \\ P(x) & = & x^2 + x + \theta^7 \end{array} \tag{4.2}$$

En exprimant $\theta$ et $\omega$ dans le corps de l'AES de base $\mathcal{B}_{aes} = (\zeta^7, \ldots, \zeta, 1)$, on construit une matrice de passage permettant de passer d'une représentation dans une base à une autre. Pour ce faire, après avoir factorisé $U(x)$ et $P(x)$ dans la base $B_{aes}$, on choisit :

$$\begin{array}{rcl} \theta & = & \zeta^6 + \zeta^3 + \zeta^2 + \zeta \\ \omega & = & \zeta^4 + \zeta \end{array} \tag{4.3}$$

et ainsi on en déduit la matrice passage de la base $\mathcal{B}_{aes}$ à la base $\mathcal{B}$ suivante :

$$M_p = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.4}$$

Chaque élément $a$ représenté dans la base $\mathcal{B}_{aes}$ est alors représenté dans la base $\mathcal{B}$ en calculant :

$$b \leftarrow M_p^{-1} \cdot a \ . \tag{4.5}$$

**Remarque 4.2.1.** *Dans la suite, un tel élément b pourra être vu soit comme un nombre binaire* $(b_0b_1b_2b_3b_4b_5b_6b_7)_2$ *(où* $b = b_0 + b_1\theta + b_2\theta^2 + b_3\theta^3 + b_4\omega + b_5\omega\theta + b_6\omega\theta^2 + b_7\omega\theta^3$*), soit comme une paire de 4 bits représentée par un polynôme de degré 1, noté alors* $b(x) = b_l + b_h x$ *avec* $b_l = b_0 + b_1\theta + b_2\theta^2 + b_3\theta^3$ *et* $b_h = b_4 + b_5\theta + b_6\theta^2 + b_7\theta^3$*.*

**Notation 4.2.2.** *Pour simplifier, on note dans la suite* $\mathbf{K}[x]_d$ *l'ensemble des polynômes de degré au plus d à coefficients sur* $\mathbf{K}$*, où* $\mathbf{K}$ *est la représentation du corps* $\mathbb{F}_{2^4}$ *définie par l'extension* $\mathbb{F}_2[x]/U(x)$*.*

En considérant chaque élément défini sur l'extension $\mathbf{L} = \mathbf{K}[x]/P(x)$, on décrit une variante du schéma de partage de secret de Shamir associée à l'ensemble suivant :

$$E = \{(s(x), v_{s(x)}) : s(x) \in \mathbf{K}[x]_1, v_{s(x)} \in E_{s(x)} = \mathbf{K}[x]/p_1(x) \times \ldots, \mathbf{K}[x]/p_n(x)\} \ , \tag{4.6}$$

où $p_1(x), \ldots, p_n(x)$ sont des polynômes deux à deux premiers entre *eux* et premiers avec $P(x)$.
On définit le schéma de partage de secret associé à cet ensemble $E$ à l'aide de la **procédure de partage** décrite par l'algorithme suivant :

**Algorithme 21** Procédure de partage

ENTRÉES: Soient $E$ l'ensemble défini par la relation (4.6) ; $n$, $k$ des entiers, où $n > k$. Considérons $n$ polynômes de degré non nul à coefficients sur $\mathbf{K}$, deux à deux premiers entre eux et premiers avec $P(x)$, notés : $p_1(x), \ldots, p_n(x)$ et tels que le degré de $\prod_{i=1}^{n} p_i(x)$ soit supérieur ou égal à $k+1$. Soit $s(x) \in \mathbf{K}[x]_1$ un secret
SORTIE: $v_{s(x)}$ un vecteur de partage appartenant à $E_{s(x)}$

**Fonction :** partage$(s(x))$
1. Générer un polynôme $r(x) \in_R \mathbf{K}[x]_{k-2}$
2. Construire le polynôme $f \in \mathbf{K}[x]_k$ défini tel que : $f(x) = s(x) + r(x)P(x)$
3. **Pour** $i = 1$ *à* $n$ **faire:**
4. $\quad c_i \leftarrow f(x) \mod p_i(x)$
5. **Retourner** $v_{s(x)} \leftarrow (c_1, \ldots, c_n)$

Les polynômes $p_1(x), \ldots, p_n(x)$ utilisés par l'algorithme 21 étant choisis deux à deux premiers entre eux, on peut déduire d'après le théorème des restes chinois l'isomorphisme suivant :

$$\mathbf{K}[x] \,/\, \prod_{i=1}^{n} p_i(x) \simeq \mathbf{K}[x]/p_1(x) \times \ldots \times \mathbf{K}[x]/p_n(x) \ , \tag{4.7}$$

avec par hypothèse le degré de $\prod_{i=1}^{n} p_i(x)$ supérieur ou égal à $k+1$.
Par conséquent, tout polynôme $f(x) \in \mathbf{K}[x]_k$ peut être reconstruit à partir des évaluations modulo $p_i(x)$, avec $i = 1, \ldots, n$. Ainsi, pour tout vecteur de partage $(c_1, \ldots, c_n) \in E_{s(x)}$, construit à partir de la procédure de partage décrite par l'algorithme 21, il existe un unique représentant $f(x) \in \mathbf{K}[x]_k$ vérifiant $f(x) = s(x) + r(x)P(x)$. Pour tout vecteur de partage $(c_1, \ldots, c_n) \in E_{s(x)}$, le secret $s(x)$ peut alors être reconstruit. La **procédure de reconstruction** consiste alors à déterminer un groupe qualifié $Q \subseteq \{1, \ldots, n\}$ tel que le degré de $\prod_{i \in Q} p_i(x)$ soit $\geqslant k+1$, puis à effectuer :

$$\text{reconstruction}((c_i)_{i \in Q}) = \sum_{i \in Q} c_i \lambda_i \mod P(x) \tag{4.8}$$

où le vecteur $\lambda = (\lambda_1, \ldots, \lambda_n)$ est défini pour tout $i = 1, \ldots, n$ par :

$$\lambda_i = \begin{cases} (q_i(x)^{-1} \mod p_i(x)) q_i(x) \mod \prod_{i \in Q} p_i(x), \text{ si } i \in Q \\ 0 \text{ sinon} \end{cases} \tag{4.9}$$

avec $q_i(x) = \prod_{j \in Q, j \neq i} p_j(x)$.

Comme on effectue une réduction modulo $P(x)$, les coordonnées du vecteur $\lambda$ peuvent alors être considérées comme des polynômes de degré 1, à coefficient dans $\mathbf{K}$ : $\lambda_i \in \mathbf{K}[x]_1$, pour tout $i \in Q$.

**Remarque 4.2.3.** *Si les polynômes $p_1(x), \ldots, p_n(x)$ sont de degré 1,* i.e., *$p_i(x) = x - \alpha_i$ avec les $\alpha_i$ deux à deux distincts, pour $i = 1, \ldots, n$, alors la réduction du polynôme $f(x)$ modulo $p_i(x)$ revient à évaluer $f(\alpha_i)$. De plus, le vecteur $\lambda$ correspond aux polynômes d'interpolation de Lagrange réduits modulo $P(x)$. En effet pour tout $i \in Q$ avec $Q \subseteq \{1, \ldots, n\}$ un groupe qualifié tel que le degré de $\prod_{i \in Q} p_i(x)$ soit $\geqslant k+1$ :*

$$q_i(x) = \prod_{j \in Q, j \neq i} x - \alpha_j \ , \tag{4.10}$$

*d'où, pour tout $i \in Q$ :*

$$\begin{array}{rcl} \lambda_i & = & (q_i(x)^{-1} \mod p_i(x)) q_i(x) \mod P(x) \\ & = & (\prod_{j \in Q, j \neq i} \frac{1}{x - \alpha_j} \mod x - \alpha_i)(\prod_{j \in Q, j \neq i} x - \alpha_j) \mod P(x) \\ & = & \prod_{j \in Q, j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j} \mod P(x) \ . \end{array} \tag{4.11}$$

Considérons $(c_1, \ldots, c_n) \in E_{s(x)}$ un vecteur de partage associé à un polynôme $f(x) \in \mathbf{K}[x]_k$ et $Q \subseteq \{1, \ldots, n\}$ un groupe qualifié tel que le degré de $\prod_{i \in Q} p_i(x)$ soit $\geqslant k+1$. Pour simplifier, on suppose dans la suite que les polynômes $p_1(x), \ldots, p_n(x)$ sont tous de degré 1. Dans ce cas, les paramètres de reconstruction $r$ et de sécurité $t$ vérifient :

$$\begin{array}{rcl} r & = & k+1 \\ t & = & k-1 \end{array} \tag{4.12}$$

En effet, d'après le choix des polynômes $p_i(x)$, on peut choisir $Q$ tel que $\mathrm{card}(Q) = k+1$. D'après la relation (4.8), le secret $s(x)$ peut être déterminé à partir de ces $k+1$ coordonnées en reconstruisant le polynôme $f(x) \mod \prod_{i \in Q} p_i(x)$, puis en le réduisant modulo $P(x)$. Cependant, avec seulement $k$ coordonnées, on peut reconstruire au mieux un polynôme de degré $k-1$, d'où $r = k+1$. De plus, comme les coordonnées du vecteur de partage sont des éléments définis sur $\mathbf{K}$, la connaissance de $k$ coordonnées permet à un attaquant de construire 16 polynômes $f(x)$ possibles distincts de degré $k$ en testant les 16 valeurs possibles de la $(k+1)$-ième coordonnée manquante. Ainsi en faisant cette recherche exhaustive, le nombre de solutions possibles pour le secret $s(x)$ est 16 au lieu de 256. C'est donc pour assurer l'équiprobabilité de toutes les valeurs $s(x) \in \mathbf{K}[x]_1$ possibles que le **paramètre de sécurité** est réduit à $t = k-1$. Par conséquent, le schéma de partage de secret construit à partir de la procédure de partage donnée par l'algorithme 21 n'est pas un schéma à seuil (*i.e.,* $r \neq t+1$).

On peut noter qu'un tel schéma est **linéaire**. En effet, il est facile de vérifier que pour tout vecteur de partage $v_{s(x)}, v_{s'(x)}$ associé à $E$ et tout scalaire $\gamma$, on a :

$$\begin{array}{rcl} v_{s(x)} + v_{s'(x)} & = & v_{s(x)+s'(x)} \in E_{s(x)+s'(x)} \\ \gamma v_{s(x)} & = & v_{\gamma s(x)} \in E_{\gamma s(x)} \end{array} \tag{4.13}$$

De ce fait, le schéma de partage de secret associé à $E$ peut être construit en considérant un code linéaire $\mathcal{C}$. En particulier, en supposant les polynômes $p_i(x)$, pour $i = 1, \ldots, n$ de degré 1 : *i.e.,* $p(x_i) = x - \alpha_i$, avec les $\alpha_i$ distincts, on peut décrire la procédure de partage à l'aide de la matrice génératrice $G$ suivante :

$$G = \begin{pmatrix} 1 & 0 & 1 & \ldots & \ldots & 1 \\ 0 & 1 & \alpha_1 & \ldots & \ldots & \alpha_n \\ 0 & 0 & \theta^7 + \alpha_1 + \alpha_1^2 & \ldots & \ldots & \theta^7 + \alpha_n + \alpha_n^2 \\ 0 & 0 & (\theta^7 + \alpha_1 + \alpha_1^2)\alpha_1 & \ldots & \ldots & (\theta^7 + \alpha_n + \alpha_n^2)\alpha_n \\ 0 & 0 & (\theta^7 + \alpha_1 + \alpha_1^2)\alpha_1^2 & \ldots & \ldots & (\theta^7 + \alpha_n + \alpha_n^2)\alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & (\theta^7 + \alpha_1 + \alpha_1^2)\alpha_1^{k-2} & \ldots & \ldots & (\theta^7 + \alpha_n + \alpha_n^2)\alpha_n^{k-2} \end{pmatrix} . \tag{4.14}$$

Le code linéaire $\mathcal{C}$ est donc de paramètres $[n+2, k+1]$ défini sur $\mathbf{K}$. Les deux premières coordonnées de chaque mot de $\mathcal{C}$ sont donc considérées comme secrètes. On a alors :

$$\forall (s_l, s_h, c_1, \ldots, c_n) \in \mathcal{C} \Leftrightarrow (c_1, \ldots, c_n) \in E_{s(x)}(\mathcal{C}) \ , \tag{4.15}$$

avec $s(x) = s_l + s_h x \in \mathbf{K}[x]_1$.

La procédure de partage appliquée sur un secret $s(x)$ consiste alors :
- à générer $k-1$ éléments : $r_0, \ldots, r_{k-2} \in_R \mathbf{K}$,
- puis à calculer : $(s_l, s_h, c_1, \ldots, c_n) \leftarrow (s_l, s_h, r_0, \ldots, r_{k-2}) \cdot G$,
- et enfin à retourner $(c_1, \ldots, c_n)$.

Par construction, on obtient alors un vecteur de partage de $s(x)$ comme attendu, *i.e.,* pour tout $i = 1, \ldots, n : c_i = f(\alpha_i)$, où $f(x) = s(x) + r(x)P(x)$ et $r(x) = r_0 + r_1 x + \ldots + r_{k-2}x^{k-2}$.

Dans la suite, on supposera que la procédure de partage décrite par l'algorithme 21 est implantée comme décrit précédemment. Ainsi, cette procédure requiert :
- $k-1$ éléments générés uniformément sur $\mathbf{K}$,
- $kn$ multiplications entre des éléments définis sur $\mathbf{K}$ (utilisant la table pré-calculée décrite par la relation (4.17)),
- $kn$ additions entre des éléments définis sur $\mathbf{K}$,
- et le pré-calcul des éléments $(\theta^7 + \alpha_i + \alpha_i^2)\alpha_i^j$, pour $i = 1, \ldots, n$ et $j = 1, \ldots, k-2$.

### 4.2.2 Implantation des transformations linéaires

Considérons le schéma de partage de secret construit à partir de la procédure de partage donnée par l'algorithme 21 avec $n = k + 1$. Par construction, $n = t + 2$, où $t$ est le paramètre de sécurité, les vecteurs de partage de ce schéma sont donc de longueur $t + 2$, constitués d'éléments définis sur $\mathbb{F}_{2^4}$. Bien que ce schéma ne soit pas à seuil, l'implantation des transformations linéaires de l'AES résistant à l'ordre $t$ à l'aide de ce dernier

reste pertinent, car le coût imputé par la coordonnée supplémentaire par vecteur de partage reste négligeable. De plus, comme les coordonnées des vecteurs de partage sont des éléments définis sur 4 bits, on peut supposer que sur un processeur 8 bits des optimisations d'implantation peuvent être appliquées.

### 4.2.3 Procédure sécurisée de la transformation *SubBytes*

Pour implanter la transformation *SubBytes* de manière sécurisée (*i.e.,* résistant à l'ordre $t$), supposons qu'en entrée et en sortie, on attende 16 vecteurs de partage associés à $E$ construit à partir de la procédure de partage donnée par l'algorithme 21 avec $n = t+2$. On propose d'appliquer indépendamment sur chacun des vecteurs de partage :

- la procédure d'inversion décrite à partir de la relation (2.48) requérant l'application de 4 procédures de multiplication et de 7 procédures d'élévation au carré,
- puis la transformation affine.

Pour ce faire, on propose respectivement dans les sous-sections 4.2.3.1, 4.2.3.2 et 4.2.3.3, l'implantation des procédures de multiplication, d'élévation au carré et de la transformation affine.

#### 4.2.3.1 Procédure de multiplication sécurisée

Dans un premier temps, considérons le schéma de partage de secret construit à partir de la procédure de partage donnée par l'algorithme 21 avec $n = 2t + 3$ et $k = t + 1$, où $t$ est le paramètre de sécurité. Comme explicité dans la sous-section 4.2, un tel schéma correspond à un code linéaire de paramètres $[2t + 5, t + 2]$ défini sur $\mathbf{K}$ que l'on note $\mathcal{C}$. Comme le produit de deux polynômes de degré $t + 1$ retourne un polynôme de degré $2t + 2$, on peut en déduire que le code $\widehat{\mathcal{C}}$ engendré par $\{c * c' : c, c' \in \mathcal{C}\}$ est associé au schéma de partage de secret construit à partir de la procédure de partage donnée par l'algorithme 21 avec les paramètres $n = 2t + 3$, $k = 2t + 2$ et de paramètre de sécurité $\widehat{t} = 2t + 1$. Autrement dit, $\widehat{\mathcal{C}}$ est un code linéaire de paramètres $[2t + 5, 2t + 3]$ défini sur $\mathbf{K}$ vérifiant les propriétés suivantes :

$$\begin{cases} \widehat{d} & \geqslant 2 \\ \widehat{t} & \geqslant t \end{cases} \tag{4.16}$$

avec $\widehat{d}$ la distance minimale de $\widehat{\mathcal{C}}$.
Par conséquent, en considérant le code linéaire $\mathcal{C}$, les procédures de multiplication décrites pour les codes linéaires peuvent être adaptées (cf. chapitre 3). En effet, considérons deux vecteurs de partage $(c_1, \ldots, c_{2t+3}) \in E_{s(x)}(\mathcal{C})$ et $(c'_1, \ldots, c'_{2t+3}) \in E_{s'(x)}(\mathcal{C})$. En appliquant le produit de Schur entre ces deux vecteurs de partage, on obtient un vecteur de partage $(c_1c'_1, \ldots, c_{2t+3}c'_{2t+3}) \in E_{s''(x)}(\widehat{\mathcal{C}})$, avec $s''(x) = s(x)s'(x) \mod P(x)$. Notons $\widehat{\lambda}$ le vecteur de reconstruction du schéma de partage de secret associé à $\widehat{\mathcal{C}}$. D'après le paramètre de sécurité relatif au schéma $E(\widehat{\mathcal{C}})$, *i.e.,* comme $\widehat{t} > t$, l'observation de $t$ valeurs intermédiaires

durant la manipulation du vecteur $(c_1c'_1, \ldots, c_{2t+3}c'_{2t+3})$ n'apporte pas d'information sur le secret $s''(x)$, et donc l'observation de $t$ valeurs intermédiaires durant la manipulation du vecteur $(c_1c'_1\widehat{\lambda}_1, \ldots, c_{2t+3}c'_{2t+3}\widehat{\lambda}_{2t+3})$ non plus. De plus, on peut noter que $\widehat{\lambda}_i \in K[x]_1$, pour tout $i = 1, \ldots, 2t+3$, d'où : $c_ic'_i\widehat{\lambda}_i \in K[x]_1$. La procédure de partage peut donc être appliquée sur chacune des parts : $c_ic'_i\widehat{\lambda}_i$, pour $i = 1, \ldots, 2t+3$. Par conséquent, la **procédure de multiplication standard** (Alg. 16) est applicable en considérant ces codes $\mathcal{C}$ et $\widehat{\mathcal{C}}$, avec la procédure de partage décrite par l'algorithme 21.

De plus, comme $\widehat{t} - t = t + 1$ et $c_ic'_i\widehat{\lambda}_i \in K[x]_1$, pour $i = 1, \ldots, 2t+3$, d'après le lemme 3.4.5, on peut additionner $e+1 = t+2$ parts du vecteur $(c_1c'_1\widehat{\lambda}_1, \ldots, c_{2t+3}c'_{2t+3}\widehat{\lambda}_{2t+3})$ entre elles, sans révéler de l'information sur le secret $s''(x)$. D'où **l'amélioration de la procédure de multiplication standard** est également applicable avec $e = t + 1$ (Alg. 17).

Par ailleurs, l'amélioration proposée par Louis Goubin et Ange Martinelli pour le schéma de partage de secret de Shamir (Alg. 18), qui consiste à générer à la volée des parts supplémentaires pour permettre de manipuler moins de parts en entrée et en sortie de l'algorithme, peut également être adaptée. En particulier, l'algorithme 22 combine cette amélioration avec celle décrite par l'algorithme 17 avec $e = t + 1$.

---

**Algorithme 22** Procédure de multiplication sécurisée pour la variante du schéma de partage de secret de Shamir

---

ENTRÉES: Soient $\mathcal{C}$ et $\widehat{\mathcal{C}}$ les codes de longueur $2t + 5$ et de dimensions respectives $t + 2$ et $2t + 3$ permettant de définir respectivement les schémas de partage de secret associé à $E(\mathcal{C})$ et $E(\widehat{\mathcal{C}})$ donnée par la relation (4.6) ; $\mathcal{C}'$ le code de longueur $t + 3$ et de dimension $t + 2$ obtenu en tronquant $\mathcal{C}$ ; $t + 1$ éléments publics distincts non nuls : $\alpha_{t+3}, \ldots, \alpha_{2t+3}$ ; $\beta_j(\alpha_i)$ des éléments pré-calculés définis sur $\mathbb{F}_{2^4}$ pour $1 \leqslant j \leqslant t + 2$ et $t + 3 \leqslant i \leqslant 2t + 3$, où les $\beta_j(x)$ sont des polynômes d'interpolation de Lagrange et $\widehat{\lambda}$ un vecteur de reconstruction de $\widehat{\mathcal{C}}$
Soit $(c_1, \ldots, c_{t+2}) \in E_{s(x)}(\mathcal{C}')$, $(c'_1, \ldots, c'_{t+2}) \in E_{s'(x)}(\mathcal{C}')$
SORTIE: $(z_1, \ldots, z_{t+2}) \in E_{s''(x)}(\mathcal{C}')$, avec $s''(x) = s(x)s'(x) \mod P(x)$

---

*Génération à la volée des $t + 1$ parts supplémentaires :*

1. $(c_1, \ldots, c_{t+2}, c_{t+3}, \ldots, c_{2t+3}) \leftarrow \left(c_1, \ldots, c_{t+2}, \sum_{j=1}^{t+2} c_j\beta_j(\alpha_{t+3}), \ldots, \sum_{j=1}^{t+2} c_j\beta_j(\alpha_{2t+3})\right)$
2. $\left(c'_1, \ldots, c'_{t+2}, c'_{t+3}, \ldots, c'_{2t+3}\right) \leftarrow \left(c'_1, \ldots, c'_{t+2}, \sum_{j=1}^{t+2} c'_j\beta_j(\alpha_{t+3}), \ldots, \sum_{j=1}^{t+2} c'_j\beta_j(\alpha_{2t+3})\right)$

*Procédure de multiplication améliorée :*

3. $(w_1, \ldots, w_{2t+3}) \leftarrow \left(c_1c'_1, \ldots, c_{2t+3}c'_{2t+3}\right) + \mathsf{partage}_{\widehat{\mathcal{C}}}(0)$
4. $(w_{t+2}, \ldots, w_{2t+3}) \leftarrow \left(\sum_{i=1}^{t+2} \widehat{\lambda}_i w_i, \widehat{\lambda}_{t+3}w_{t+3}, \ldots, \widehat{\lambda}_{2t+3}w_{2t+3}\right)$
5. $(z_1, \ldots, z_{t+2}) \leftarrow \mathrm{transcodage}_{(\widehat{\mathcal{C}} \mapsto \mathcal{C}')}\left(\mathbf{1}_{t+2}, (w_{t+2}, \ldots, w_{2t+3})\right)$
6. **Retourner** $(z_1, \ldots, z_{t+2})$

---

Le schéma de partage de secret considéré étant défini sur $\mathbf{K}$, toutes les opérations requises durant les procédures de multiplication peuvent alors être effectuées sur $\mathbf{K}$. Les

multiplications pourront alors être implantées à l'aide d'une unique table pré-calculée :

$$\forall a, b \in \mathbf{K}, \mathsf{table}[a][b] = ab \ , \tag{4.17}$$

au lieu de faire appel à un traitement plus coûteux utilisant les tables pré-calculées alog,log. En particulier, à l'étape 4 de l'algorithme 22, comme les $\widehat{\lambda}_i$, pour $i = 1 \ldots, 2t+3$ sont des éléments de $\mathbf{K}[x]_1$, les multiplications $\widehat{\lambda}_i w_i$ sont effectuées comme deux multiplications par des scalaires qui requiert donc 2 accès à la table pré-calculée table.

L'implantation de la procédure de multiplication résistante aux attaques d'ordre $t$, décrite par l'algorithme 22 requiert alors :

- $t^2 + 4t + 1$ éléments générés uniformément sur $\mathbf{K}$
- $t^3 + 12t^2 + 26t + 14$ additions entre des éléments définis sur $\mathbf{K}$
- $t^3 + 9t^2 + 28t + 20$ multiplications entre des éléments définis sur $\mathbf{K}$ utilisant la table pré-calculée décrite par la relation (4.17)

La table 4.1 donne une application numérique de l'algorithme 22 résistant aux attaques d'ordre $t = 1, \ldots, 6$. Plus précisément, cette table indique une estimation en nombre de cycles lorsque l'on suppose cet algorithme implanté sur un composant avec un processeur 8 bits, où une simple addition bit à bit requiert seulement un cycle, la génération de 4 bits d'aléa requiert également un cycle et l'accès à une table pré-calculée requiert 3 cycles. On peut supposer à titre d'indication que le traitement permettant d'effectuer une multiplication entre deux éléments de $\mathbb{F}_{256}$ utilisant les tables pré-calculées alog, log requiert grossièrement 20 cycles.
En comparant cette table, avec celle présentant le coût des différentes procédures de multiplication appliquées au schéma de partage de secret de Shamir, *i.e.,* table 3.2, on peut voir que l'algorithme 22 utilisant le schéma de partage de secret que l'on propose est plus efficace que l'utilisation du schéma de partage de secret de Shamir, dès $t = 1$.

| | aléa sur $\mathbf{K}$ | additions sur $\mathbf{K}$ | table | estimation (en cycles) |
|---|---|---|---|---|
| $t = 1$ | 6 | 53 | 60 | 239 |
| $t = 2$ | 13 | 122 | 128 | 519 |
| $t = 3$ | 22 | 227 | 230 | 939 |
| $t = 4$ | 33 | 374 | 372 | 1523 |
| $t = 5$ | 46 | 569 | 560 | 2295 |
| $t = 6$ | 61 | 818 | 800 | 3279 |

TABLE 4.1 – Estimation du coût de l'algorithme 22 résistant aux attaques d'ordre $t$, pour $t = 1, \ldots, 6$

#### 4.2.3.2 Procédure d'élévation à la puissance $2^m$

Avec le même raisonnement que celui donné dans la sous-section précédente, comme $\mathbf{K}$ est de caractéristique 2, on peut en déduire que la procédure d'élévation au carré décrite pour les codes linéaires (Alg. 20) est applicable sur $E$, où $E$ est construit à partir de la procédure de partage donnée par l'algorithme 21 avec les paramètres $n = t + 1$. Cette dernière consiste simplement à effectuer l'élévation au carré sur chacune des coordonnées du vecteur de partage, puis à transcoder le vecteur obtenu en un nouveau vecteur de partage associé à $E$. La procédure d'élévation à la puissance $2^m$, pour $m = 1, 2, 4$, sur un vecteur de partage de longueur $t + 2$ de $E$, requiert alors :

- $t + 2$ élévations à la puissance $2^m$,
- $t^2 + 2t$ éléments générés uniformément sur $\mathbf{K}$
- $t^3 + 6t^2 + 11t + 6$ additions entre des éléments définis sur $\mathbf{K}$
- $t^3 + 5t^2 + 10t + 8$ multiplications entre des éléments définis sur $\mathbf{K}$ utilisant la table pré-calculée décrite par la relation (4.17)

On peut noter que contrairement au schéma de partage de secret de Shamir défini sur $\mathbb{F}_{256}$, on ne peut pas appliquer l'optimisation proposée dans [PR11b] (cf. sous-sections 2.4.2.3 et 3.5.3.2) qui consiste à remplacer la procédure de transcodage par un simple ré-ordonnement des coordonnées. En effet, dans notre cas, en déterminant des points $\alpha_1, \ldots, \alpha_{t+2} \in \mathbf{K}$, tels que $\alpha_i^2 = \alpha_j$, avec $i, j = 1, \ldots, t + 2$, le vecteur de reconstruction $\lambda^2 = (\lambda_1^2, \ldots, \lambda_{t+2}^2)$ est défini pour tout $i, j = 1, \ldots, t + 2$ par :

$$\lambda_i^2 = \prod_{u=1, u\neq i}^{t+2} \frac{x^2 + \alpha_u^2}{\alpha_i^2 + \alpha_u^2} = \prod_{u=1, u\neq j}^{t+2} \frac{x + \theta^7 + \alpha_u}{\alpha_j + \alpha_u} \neq \lambda_j \ . \tag{4.18}$$

#### 4.2.3.3 Procédure sécurisée de la transformation affine

Considérons le schéma de partage de secret associé à $E$, où $E$ est construit à partir de la procédure de partage donnée par l'algorithme 21 avec les paramètres $n = t + 2$. La transformation affine pourrait être implantée à partir de la relation (2.67) qui décrit cette transformation sous sa forme polynomiale. Ce traitement requerrait alors : 7 procédures d'élévation au carré, 7 multiplications de vecteur de partage par des constantes définies sur $\mathbf{K}[x]_1$ et 8 additions de vecteurs de partage. Or comme chaque procédure d'élévation au carré requiert une procédure de transcodage, un tel traitement appliqué à notre schéma de partage de secret est alors extrêmement coûteux comparé à l'utilisation du schéma de partage de secret basique qui effectue cette transformation comme une transformation stable. Pour pallier cet inconvénient, on propose une alternative moins coûteuse requérant seulement une procédure de transcodage.

Par définition, la transformation affine, définie sur le corps de l'AES de base $\mathcal{B}_{aes}$,

associe à tout entier $s \in \mathbb{F}_{2^8}$ :

$$\mathsf{TransAff}_{\mathcal{B}_{aes}}[s] = \varphi_{A_{\mathcal{B}_{aes}}}(s) + b_{\mathcal{B}_{aes}} \ , \tag{4.19}$$

où $\varphi_A(s) = A_{\mathcal{B}_{aes}} \cdot s$ est une fonction linéaire sur $\mathbb{F}_2$ définie par une matrice binaire $A_{\mathcal{B}_{aes}}$ de dimension $8 \times 8$ et $b_{\mathcal{B}_{aes}}$ est un élément défini dans la base $\mathcal{B}_{aes}$.
Pour pouvoir appliquer la transformation affine sur le corps $\mathbf{L}$, on détermine la matrice $A_{\mathcal{B}}$ et l'élément $b_{\mathcal{B}}(x)$ correspondant respectivement à la représentation de la matrice $A_{\mathcal{B}_{aes}}$ dans la base $\mathcal{B}$ et celle de $b_{\mathcal{B}_{aes}}$. Pour ce faire on effectue les changements de base suivants :

$$\begin{array}{rcl} A_{\mathcal{B}} & \leftarrow & M_p^{-1} \cdot A_{\mathcal{B}_{aes}} \cdot M_p \\ b_{\mathcal{B}}(x) & \leftarrow & M_p^{-1} \cdot b_{\mathcal{B}_{aes}} \end{array} \tag{4.20}$$

Ainsi, on peut définir la transformation affine sur $\mathbf{L}$ par :

$$\mathsf{TransAff}[s(x)] = \varphi_{A_{\mathcal{B}}}(s(x)) + b_{\mathcal{B}}(x) \ . \tag{4.21}$$

De plus, pour tout vecteur de partage $(c_1, \ldots, c_{t+2}) \in E_{s(x)}$, on a par linéarité de $\varphi_{A_{\mathcal{B}}}$ :

$$\varphi_{A_{\mathcal{B}}}\left(s(x)\right) = \varphi_{A_{\mathcal{B}}}\left(\sum_{i=1}^{t+2} c_i\lambda_i\right) = \sum_{i=1}^{t+2} \varphi_{A_{\mathcal{B}}}(c_i\lambda_i) \ . \tag{4.22}$$

D'après cette remarque, on propose alors d'effectuer la transformation affine en déterminant dans un premier temps, le vecteur $(\varphi_{A_{\mathcal{B}}}(c_1\lambda_1), \ldots, \varphi_{A_{\mathcal{B}}}(c_{t+2}\lambda_{t+2}))$. Puis, on propose de transcoder ce vecteur en un vecteur de partage de $E_{\varphi_{A_{\mathcal{B}}}(s(x))}$. En effet, en notant $(w_{i,1}, \ldots, w_{i,t+2})$ le résultat de la procédure $\mathsf{partage}(\varphi_{A_{\mathcal{B}}}(c_i\lambda_i))$, pour tout $i = 1, \ldots, t+2$, on a :

$$\begin{array}{cccccc} & (w_{1,1}, & \ldots, & w_{1,t+2}) & \in E_{\varphi_{A_{\mathcal{B}}}(c_i\lambda_i)} \\ + & \vdots & \vdots & \vdots & \vdots \\ + & (w_{t+2,1}, & \ldots, & w_{t+2,t+2}) & \in E_{\varphi_{A_{\mathcal{B}}}(c_{t+2}\lambda_{t+2})} \\ \hline = & (\sum_{i=1}^n w_{i,1}, & \ldots, & \sum_{i=1}^n w_{i,t+2}) & \in E_{\varphi_{A_{\mathcal{B}}}(s(x))} \end{array} \tag{4.23}$$

Enfin, en additionnant le vecteur de partage de $E_{\varphi_{A_{\mathcal{B}}}(s(x))}$ obtenu avec un vecteur de partage de $E_{b_{\mathcal{B}}(x)}$, on obtient un vecteur de partage de $E_{\varphi_{A_{\mathcal{B}}}+b_{\mathcal{B}}(x)}$ comme souhaité.

Cette procédure est résistante aux attaques d'ordre $t$. En effet, l'observation de $t$ valeurs intermédiaires durant la manipulation du vecteur $(c_1\lambda_1, \ldots, c_{t+2}\lambda_{t+2})$ n'apporte pas d'information sur le secret $s(x)$. On a de même pour l'observation de $t$ valeurs intermédiaires durant la manipulation du vecteur $(\varphi_{A_{\mathcal{B}}}(c_1\lambda_1), \ldots, \varphi_{A_{\mathcal{B}}}(c_{t+2}\lambda_{t+2}))$ car $\varphi_{A_{\mathcal{B}}}$ est bijective et qu'elle s'applique indépendamment sur chaque coordonnée.

## 4.3 Schéma de partage de secret idéal

Dans cette section, en tirant profit des méthodes proposées dans la sous-section 2.4.2 et des propriétés issues de la construction des schémas de partage de secret par les codes linéaires, on propose une nouvelle solution pour implanter l'AES résistant aux attaques d'ordre $t$. Plus précisément, dans les sous-sections 4.3.1 et 4.3.2, on suggère l'utilisation de codes auto-duaux ou faiblement auto-duaux pour implanter la procédure d'inversion, puis de changer de code afin d'effectuer efficacement les transformations stables à l'aide d'un code MDS. L'implantation de notre solution est présentée en sous-section 4.3.3.

Il est à noter que les résultats présentés dans cette section ont fait l'objet d'une publication à IMA CC 2013 [CRZ13].

### 4.3.1 Codes auto-duaux ou faiblement auto-duaux

Pour effectuer la procédure d'inversion sécurisée requise par la transformation *SubBytes* de l'AES, on suggère de considérer un schéma de partage de secret construit à partir d'un code linéaire pour lequel le lemme 3.4.2 est vérifié. Dans ce cas, les procédures de multiplication sécurisées décrites dans le chapitre 3 seront applicables. On considère alors la famille des codes auto-duaux (et faiblement auto-duaux) qui satisfait les propriétés requises par le lemme 3.4.2 :

**Théoreme 4.3.1.** *Soit $\mathcal{C}$ un code auto-dual (ou faiblement auto-dual) de longueur $n+1$ défini sur $\mathbb{F}_q$ de dual $\mathcal{C}^\perp$, tel que $d, d^\perp \geqslant 2$. Le code $\widehat{\mathcal{C}}$ satisfait les propriétés suivantes :*

$$\begin{cases} \widehat{d} & \geqslant 2 \\ \widehat{d}^\perp & \geqslant d^\perp \end{cases} \tag{4.24}$$

**Lemme 4.3.2.** *Si $\mathcal{C}$ est un code linéaire auto-dual (ou faiblement auto-dual) de longueur $n+1$ défini sur $\mathbb{F}_q$, alors le mot de code $\mathbf{1}_{n+1}$ est un mot de code de $\widehat{\mathcal{C}}^\perp$ (le code produit dual de $\mathcal{C}$), où $\mathbf{1}_{n+1}$ est le vecteur tout à 1 de longueur $n+1$.*

*Démonstration.* Pour tous mots de code $c$ et $c'$ d'un code linéaire auto-dual (ou ou faiblement auto-dual) de longueur $n+1$, on a :

$$\langle c, c' \rangle = \langle c * c', \mathbf{1}_{n+1} \rangle = 0 \ . \tag{4.25}$$

□

*Preuve du théorème 4.3.1.* Soit $\mathcal{C}$ un code auto-dual (ou faiblement auto-dual) de longueur $n+1$ défini sur $\mathbb{F}_q$ de dual $\mathcal{C}^\perp$, tel que $d, d^\perp \geqslant 2$. Montrons par l'absurde que $\widehat{d} \geqslant 2$.

Supposons que $\widehat{d} = 1$. Dans ce cas, on peut supposer que le mot de code $(1, 0, \ldots, 0) \in \widehat{\mathcal{C}}$. Or d'après le lemme 4.3.2, le mot $\mathbf{1}_{n+1} \in \widehat{\mathcal{C}}^{\perp}$, on a alors par définition :

$$\langle (1, 0, \ldots, 0), \mathbf{1}_{n+1} \rangle = 0 \ , \tag{4.26}$$

ce qui est absurde, d'où $\widehat{d} \geqslant 2$.
À présent montrons que $\widehat{d}^{\perp} \geqslant d^{\perp}$. Pour ce faire, supposons par l'absurde qu'il existe un mot de code $h \in \widehat{\mathcal{C}}^{\perp}$ tel que $\mathrm{w_H}(h) < d^{\perp}$. Pour tout mot de code $c, c' \in \mathcal{C}$, on a par définition du code $\widehat{\mathcal{C}}$ :

$$\langle c * c', h \rangle = 0 \ \Leftrightarrow \langle c, c' * h \rangle = 0 \ , \tag{4.27}$$

ce qui implique que $(c' * h) \in \mathcal{C}^{\perp}$. Or comme $\mathrm{w_H}(h) < d^{\perp}$, on en déduit : $\mathrm{w_H}(c' * h) < d^{\perp}$, ce qui est impossible. On a donc $\widehat{d}^{\perp} \geqslant d^{\perp}$.

□

Dans le but d'implanter une procédure d'inversion sécurisée, on propose de considérer dans un premier temps des schémas de partage de secret construits à partir des codes définis sur $\mathbb{F}_{256}$ engendrés par des codes auto-duaux ou faiblement auto-duaux binaires, puis à partir des codes engendrés par des codes auto-duaux ou faiblement auto-duaux définis sur $\mathbb{F}_4$.

#### 4.3.1.1 Construction à partir de codes auto-duaux ou faiblement auto-duaux binaires

D'après le lemme 3.5.5, en choisissant un code linéaire $\mathcal{C}$ de paramètres $[n+1, k]$ défini sur $\mathbb{F}_{256}$ engendré à partir d'un code ayant une matrice génératrice binaire [1], la procédure d'élévation au carré peut alors être implantée comme une transformation stable. De plus, une telle construction assure que le code engendré sur $\mathbb{F}_{256}$ a les mêmes paramètres et les mêmes propriétés que le code défini sur $\mathbb{F}_2$ (cf. propriété 3.2.17). De ce fait, on propose alors de construire des schémas de partage de secret à partir de codes linéaires définis sur $\mathbb{F}_{256}$ engendrés par des codes auto-duaux ou faiblement auto-duaux binaires. En plus d'obtenir une procédure d'élévation au carré peu coûteuse, les procédures de multiplication sécurisées décrites dans le chapitre 3 (*i.e.,* Alg. 16 et Alg. 17) sont applicables (cf. théorème 4.3.1) avec une procédure de transcodage ne requérant aucune multiplication. De plus, en prenant le vecteur de reconstruction de $\widehat{\mathcal{C}}$ : $\widehat{\lambda} = \mathbf{1}_{n+1}$ (cf. lemme 4.3.2), les procédures de multiplication sécurisées s'effectuent avec une complexité en $\mathcal{O}(t)$ multiplications [2], ce qui est plus efficace qu'avec le schéma de partage de secret de Shamir qui a une complexité en $\mathcal{O}(t^3)$ multiplications [3] (cf. sous-section 3.3.3.2) et que le schéma de

1. *i.e.,* ayant ses coefficients définis sur $\mathbb{F}_2$.
2. La longueur du code $n + 1$ est linéaire en $t$.
3. Elle peut être améliorée en $\mathcal{O}(t^2 \log^4 t)$ multiplications en utilisant la transformation de Fourier discrète comme décrit dans [CPR12].

partage de secret basique (cf. sous-section 2.4.2.1) qui a une complexité en $\mathcal{O}(t^2)$ multiplications.

Les codes auto-duaux et faiblement auto-duaux binaires sont des codes très étudiés dans la littérature. En particulier, la table 4.2 présente une liste de codes binaires $\mathcal{C}$ ayant une distance minimale duale $d^\perp = t + 2$ pour $1 \leqslant t \leqslant 6$. On peut noter que les codes ayant une distance $d^\perp$ paire sont des codes auto-duaux et à l'inverse, les codes ayant une distance $d^\perp$ impaire sont des codes faiblement auto-duaux.
Dans cette table, le code de Golay raccourci [22,11,6] est construit à partir du code de Golay étendu en utilisant les mots de code commençant par 00 et 11. Le code $C_{21}$ [21,11,5] est quant à lui obtenu en supprimant une coordonnée du code de Golay raccourci. Les matrices génératrices de ces codes $\mathcal{C}$ sont données en appendice A.1. Pour des valeurs de $t$ plus grandes, le lecteur peut se référer aux codes donnés par exemple dans [CS90, GO03].

| $t$ | Code binaire $\mathcal{C}$ $[n+1, k]$ | Code binaire dual $\mathcal{C}^\perp$ avec $d^\perp = t + 2$ | Nombre d'additions lors d'une procédure de partage |
|---|---|---|---|
| 1 | Code $[7, 3]$ | Code de Hamming $[7, 4, 3]$ | 5 |
| 2 | Code de Hamming étendu $[8, 4, 4]$ | | 8 |
| 3 | Code $[21, 10]$ | $C_{21}$ $[21, 11, 5]$ | 48 |
| 4 | Code de Golay raccourci $[22, 11, 6]$ | | 44 |
| 5 | Code $[23, 11]$ | Code de Golay $[23, 12, 7]$ | 64 |
| 6 | Code de Golay étendu $[24, 12, 8]$ | | 72 |

TABLE 4.2 – Codes auto-duaux et faiblement auto-duaux binaires

À partir des codes $\mathcal{C}$ donnés dans la table 4.2, on peut alors engendrer des codes définis sur $\mathbb{F}_{256}$ permettant de construire des schémas de partage de secret pour lesquels la procédure d'élévation au carré est stable et les procédures de multiplication sécurisées (*i.e.,* Alg. 16 et Alg. 17) sont applicables. En particulier, pour chacun des codes linéaires de la table 4.2, le code $\widehat{\mathcal{C}}^\perp$ a pour distance minimale duale $\widehat{d}^\perp = n + 1$. La procédure de multiplication améliorée (Alg. 17) est alors applicable avec : $e = n + 1 - d^\perp$. Pour juger de l'efficacité de la procédure de multiplication améliorée, une comparaison numérique pour $t = 1, \ldots, 6$ entre les deux procédures de multiplication (Alg. 16 et Alg. 17) est donnée dans la table 4.4 par la colonne intitulée Code $\mathbb{F}_2$. D'après cette table, on peut voir que la procédure de multiplication améliorée donnée par l'algorithme 17 est plus efficace que la procédure de multiplication donnée par l'algorithme 16 dès $t = 1$.

#### 4.3.1.2 Construction à partir de codes auto-duaux ou faiblement auto-duaux définis sur $\mathbb{F}_4$

En vue des paramètres indiqués dans la table 4.2, on peut noter qu'un schéma de partage de secret de paramètre de sécurité $t$, construit par un code auto-dual (ou faiblement auto-dual) binaire requiert un grand nombre de parts comparé à un schéma construit par code MDS. Par exemple, pour $t = 3$, le schéma de partage de secret proposé requiert 20 coordonnées alors que pour un schéma à seuil, seulement 4.

Pour réduire le coût de notre proposition, une solution est de considérer des codes définis sur $\mathbb{F}_{256}$ engendrés par des codes auto-duaux (ou faiblement auto-duaux) définis sur $\mathbb{F}_4$ au lieu de $\mathbb{F}_2$. D'après la propriété 3.2.17, de tels codes ont les mêmes paramètres et les mêmes propriétés que les codes associés sur $\mathbb{F}_4$ et de plus, de tels codes peuvent offrir un meilleur rapport $n/t$ comparé aux codes linéaires binaires [GO03]. En particulier, la table 4.3 indique une liste de codes auto-duaux (et faiblement auto-duaux) optimaux sur $\mathbb{F}_4$ qui offrent un meilleur rapport $n/t$ que les codes binaires. De plus, cette table précise le nombre d'additions et de multiplications par $w$ nécessaire lors d'une procédure de partage, en fonction de la matrice génératrice considérée (cf. appendice A.2). La matrice génératrice étant à coefficient dans $\mathbb{F}_4 = \{0, 1, w, w + 1\}$, où $w^4 = 1$ avec $w \in \mathbb{F}_{256}$, une procédure de partage nécessite alors quelques multiplications avec $w$ qui peuvent être implantées avec un coût négligeable contrairement au schéma de partage de secret de Shamir. En effet seule la valeur constante $w$ est manipulée : la multiplication entre un élément de $\mathbb{F}_{256}$ et $w$ peut être pré-calculée à l'aide d'une table. De même, la multiplication d'un élément de $\mathbb{F}_{256}$ avec $(w + 1)$ requiert un accès à cette table et une addition.

| $t$ | Code $\mathcal{C}$ $[n+1, k]_4$ | Code dual $\mathcal{C}^\perp$ avec $d^\perp = t + 2$ | Proc. de partage add | mult $w$ |
|---|---|---|---|---|
| 1 | Code de résidus quadratiques étendu XQR(3) $[4, 2, 3]$ | | 4 | 2 |
| 3 | Code $[11, 5]$ | Code de résidus quadratiques QR(11) $[11, 6, 5]$ | 25 | 5 |
| 4 | Code de résidus quadratiques étendu XQR(11) $[12, 6, 6]$ | | 32 | 6 |
| 5 | Code $[19, 9]$ | Code de résidus quadratiques QR(19) $[19, 10, 7]$ | 69 | 9 |

TABLE 4.3 – Codes auto-duaux et faiblement auto-duaux définis sur $\mathbb{F}_4$

Comme dans le cas des codes auto-duaux et faiblement auto-duaux binaires, pour chacun des codes linéaires de la table 4.3, le code $\widehat{\mathcal{C}}^\perp$ a pour distance minimale duale $\widehat{d}^\perp = n + 1$. Par conséquent, la procédure de multiplication améliorée (Alg. 17) est aussi applicable avec : $e = n + 1 - d^\perp$.

D'après la table 3.3 et les paramètres de la table 4.3, la colonne intitulée : "Code $\mathbb{F}_4$" de la table 4.4 indique le nombre d'opérations à effectuer lors des procédures de multipli-

| $t$ | Code $\mathbb{F}_2$ | | | | Code $\mathbb{F}_4$ | | | |
|---|---|---|---|---|---|---|---|---|
| | Algo.16 | | Algo.17 | | Algo.16 | | Algo.17 | |
| 1 | 12 | aléa | 7 | aléa | 3 | aléa | **3** | **aléa** |
| | 60 | add | 20 | add | 18 | add | **10** | **add** |
| | 6 | mult | 6 | mult | 3 | mult | **3** | **mult** |
| | | | | | 6 | multw | **2** | **multw** |
| 2 | 21 | aléa | **12** | **aléa** | | | | |
| | 98 | add | **40** | **add** | | - | | - |
| | 7 | mult | **7** | **mult** | | | | |
| 3 | 180 | aléa | 46 | aléa | 40 | aléa | **21** | **aléa** |
| | 1340 | add | 239 | add | 340 | add | **120** | **add** |
| | 20 | mult | 20 | mult | 10 | mult | **10** | **mult** |
| | | | | | 50 | multw | **15** | **multw** |
| 4 | 210 | aléa | 60 | aléa | 55 | aléa | **30** | **aléa** |
| | 1344 | add | 296 | add | 462 | add | **188** | **add** |
| | 21 | mult | 21 | mult | 11 | mult | **11** | **mult** |
| | | | | | 66 | multw | **24** | **multw** |
| 5 | 220 | aléa | 71 | aléa | 144 | aléa | **57** | **aléa** |
| | 1870 | add | 467 | add | 1548 | add | **464** | **add** |
| | 22 | mult | 22 | mult | 18 | mult | **18** | **mult** |
| | | | | | 162 | multw | **45** | **multw** |
| 6 | 253 | aléa | **88** | **aléa** | | | | |
| | 2162 | add | **608** | **add** | | - | | - |
| | 23 | mult | **23** | **mult** | | | | |

TABLE 4.4 – Coût des procédures de multiplication données par les algorithmes 16 et 17 résistantes aux attaques d'ordre $t = 1, \ldots, 6$ appliquées aux schémas de partage de secret construits avec des codes définis sur $\mathbb{F}_{256}$ engendrés par des codes auto-duaux (ou faiblement auto-duaux) binaires (Code sur $\mathbb{F}_2$) et définis sur $\mathbb{F}_4$ (Code sur $\mathbb{F}_4$)

cation sécurisées : Alg. 16 et Alg. 17 en fonction du schéma de partage de secret considéré pour $t = 1, 3, 4$ et 5. En particulier, *multw* indique le nombre requis de multiplications avec la constante $w$. On peut constater que la procédure de multiplication améliorée donnée par l'algorithme 17 est plus efficace que la procédure de multiplication donnée par l'algorithme 16 dès $t = 1$. De plus, l'implantation de la procédure de multiplication améliorée avec de tels codes est plus efficace qu'avec des codes engendrés par des codes auto-duaux (ou faiblement auto-duaux) binaires, comme attendu.

Lorsque l'on considère un schéma de partage de secret construit à partir d'un code $\mathcal{C}$ défini sur $\mathbb{F}_{256}$ engendré à partir d'un code auto-dual (ou faiblement auto-dual) défini

sur $\mathbb{F}_4$, la procédure d'élévation au carré n'est pas une transformation stable sur $\mathcal{C}$. Par conséquent, cette dernière requiert une procédure de transcodage comme décrit par l'algorithme 20). Cependant, d'après le lemme 3.5.5, on en déduit que la procédure d'élévation à la puissance quatre s'effectue comme une transformation stable :

$$\forall c \in \mathcal{C} \Rightarrow c^4 \in \mathcal{C} \ . \tag{4.28}$$

D'après la relation 2.48, on peut voir que 3 procédures à la puissance 4 peuvent être requises. L'utilisation de tels schémas de partage de secret pour implanter une procédure d'inversion requiert 5 procédures de multiplication sécurisées et 3 procédures d'élévation à la puissance 4.

### 4.3.2 Implantation des transformations linéaires

Pour minimiser le coût de l'implantation sécurisée des transformations linéaires de l'AES, on propose d'effectuer un changement de code, de sorte à effectuer ces transformations à l'aide d'un schéma de partage de secret à seuil. Pour implanter un AES sécurisé, on suggère alors d'utiliser :

- le code MDS de parité de paramètres $[t+2, t+1]_{256}$ pour implanter les transformations stables
- un code auto-dual (ou faiblement auto-dual) de paramètres $[n+1, k+1]_{256}$ de distance minimale duale $d^{\perp} = t+2$.

Les transformations stables seront alors effectuées avec le même coût que les solutions proposées dans la sous-section 2.4.2. En particulier, la transformation affine de la transformation *SubBytes* peut être effectuée comme une transformation stable (cf. relation 2.44). L'implantation de la transformation *SubBytes* que nous proposons est décrite plus précisément par l'algorithme 23.

### 4.3.3 Procédure sécurisée de la transformation *SubBytes*

Dans cette sous-section, après avoir explicité l'implantation sécurisée de la transformation *SubBytes* que l'on propose (cf. Alg. 23), on décrit la procédure sur un exemple utilisant le code de résidus quadratiques étendu `XQR`(3) $[4, 2, 3]$ défini dans la table 4.3. De plus, on illustre la résistance en pratique de notre solution en présentant les résultats obtenus lors d'attaques CPA d'ordre 1 et d'ordre 2 ciblant des implantations résistantes à ces attaques.

#### 4.3.3.1 Description

L'algorithme 23 présente l'implantation de la transformation *SubBytes* que l'on propose. En particulier, transcodage et SecMult2 réfèrent aux procédures sécurisées décrites respectivement par l'algorithme 15 et l'algorithme 17.

**Algorithme 23** Implantation de la transformation *SubBytes* résistante aux attaques d'ordre $t$

ENTRÉES: Soient $\mathcal{C}^\star$ le code de parité de paramètres $[t+2, t+1]$ défini sur $\mathbb{F}_{256}$ et $\mathcal{C}$ un code défini sur $\mathbb{F}_{256}$ de paramètres $[n+1, k+1]$ engendré par un code auto-dual (ou faiblement auto-dual) tel que $d^\perp = t+2$, $e = \widehat{d}^\perp - d^\perp > 0$ et $\widehat{\lambda} = \mathbf{1}_{n+1}$, le vecteur de reconstruction associé à $\widehat{\mathcal{C}}$
Soit $(x_1, \ldots, x_{t+1}) \in E_s(\mathcal{C}^\star)$
Notons la transformation affine, pour tout $x \in \mathbb{F}_{256}$ : $x \mapsto A(x) + b$
SORTIE: $(z_1, \ldots, z_{t+1}) \in E_{sbox(s)}(\mathcal{C}^\star)$

1. **Si** $\mathcal{C}$ est un code engendré par un code binaire **alors**
2. $\quad (c_1, \ldots, c_n) \leftarrow \mathsf{transcodage}_{(\mathcal{C}^\star \to \mathcal{C})}(\mathbf{1}_{t+1}, (x_1, \ldots, x_{t+1}))$ *(vecteur de partage de $E_s(\mathcal{C})$)*
3. $\quad (y_1, \ldots, y_n) \leftarrow (c_1^2, \ldots, c_n^2)$ *(vecteur de partage de $E_{s^2}(\mathcal{C})$)*
4. **Sinon** (*i.e.,* $\mathcal{C}$ est un code engendré par un code défini sur $\mathbb{F}_4$), **alors**
5. $\quad (y_1, \ldots, y_{t+1}) \leftarrow (x_1^2, \ldots, x_{t+1}^2)$ *(vecteur de partage de $E_{s^2}(\mathcal{C}^\star)$)*
6. $\quad (c_1, \ldots, c_n) \leftarrow \mathsf{transcodage}_{(\mathcal{C}^\star \to \mathcal{C})}(\mathbf{1}_{t+1}, (x_1, \ldots, x_{t+1}))$ *(vecteur de partage de $E_s(\mathcal{C})$)*
7. $\quad (y_1, \ldots, y_n) \leftarrow \mathsf{transcodage}_{(\mathcal{C}^\star \to \mathcal{C})}(\mathbf{1}_{t+1}, (y_1, \ldots, y_{t+1}))$ *(vecteur de partage de $E_{s^2}(\mathcal{C})$)*
8. $(z_1, \ldots, z_n) \leftarrow \mathsf{SecMult2}((c_1, \ldots c_n), (y_1, \ldots, y_n))$ *(vecteur de partage de $E_{s^3}(\mathcal{C})$)*
9. $(w_1, \ldots, w_n) \leftarrow (z_1^4, \ldots, z_n^4)$ *(vecteur de partage de $E_{(s^3)^4}(\mathcal{C})$)*
10. $(z_1, \ldots, z_n) \leftarrow \mathsf{SecMult2}((z_1, \ldots z_n), (w_1, \ldots, w_n))$ *(vecteur de partage de $E_{s^{15}}(\mathcal{C})$)*
11. $(z_1, \ldots, z_n) \leftarrow (z_1^{16}, \ldots, z_n^{16})$ *(vecteur de partage de $E_{(s^{15})^{16}}(\mathcal{C})$)*
12. $(z_1, \ldots, z_n) \leftarrow \mathsf{SecMult2}((z_1, \ldots z_n), (w_1, \ldots, w_n))$ *(vecteur de partage de $E_{s^{252}}(\mathcal{C})$)*
13. $(z_1, \ldots, z_{t+1}) \leftarrow \mathsf{transcodage}_{(\mathcal{C} \to \mathcal{C}^\star)}(\mathbf{1}_n, (z_1 y_1, \ldots, z_n y_n))$ *(vecteur de partage de $E_{s^{254}}(\mathcal{C}^\star)$)*
14. $(z_1, \ldots, z_{t+1}) \leftarrow (A(z_1) + b, A(z_2), \ldots, A(z_{t+1}))$ *(vecteur de partage de $E_{sbox(s)}(\mathcal{C}^\star)$)*
15. **Retourner** $(z_1, \ldots, z_{t+1})$

**Théoreme 4.3.3** (Transformation *SubBytes* sécurisée)**.** *La procédure décrite par l'algorithme 23 est résistante aux attaques d'ordre $t$.*

*Démonstration.* D'après les paramètres des codes linéaires utilisés dans l'algorithme 23, les conditions attendues par les théorèmes 3.4.3 et 3.4.7 du chapitre 3 sont remplies. L'implantations des procédures, notées transcodage et SecMult2 décrites respectivement par l'algorithme 15 et l'algorithme 17, peuvent alors être effectuées de manière résistante aux attaques d'ordre $t$. De plus, les procédures restantes étant effectuées comme des transformations stables, leur implantation est également résistante aux attaques d'ordre $t$. Par conséquent, la procédure décrite par l'algorithme 23 est résistante aux attaques d'ordre $t$. □

#### 4.3.3.2 Exemple : utilisation du code auto-dual XQR(3)

Afin d'expliciter la transformation *SubBytes* sécurisée de l'AES que l'on propose, on présente un exemple d'application résistant aux attaques d'ordre 1, utilisant le code défini sur $\mathbb{F}_{256}$ engendré par le code auto-dual $\mathcal{C}$ construit à partir de la matrice génératrice du

code $\texttt{XQR}(3)$ défini sur $\mathbb{F}_4 = \{0, 1, w, w+1\}$. En entrée de l'algorithme 23, on a pour chaque état $s$ de l'AES, un vecteur de partage $(x_1, x_2) \in E_s(\mathcal{C}^\star)$, où $\mathcal{C}^\star$ est le code de parité de paramètres $[3, 2]$ défini sur $\mathbb{F}_{256}$.
Comme la matrice génératrice du code auto-dual considéré a ses coefficients dans $\mathbb{F}_4$, on commence par effectuer la procédure d'élévation au carré directement sur le vecteur de partage $(x_1, x_2)$, afin que cette transformation soit stable. On obtient alors :

$$(y_1, y_2) \leftarrow (x_1^2, x_2^2) \ . \tag{4.29}$$

Les vecteurs de partage associés aux secrets $s$ et $s^2$ (*i.e.*,$(x_1, x_2)$ et $(y_1, y_2)$), sont transcodés en deux nouveaux vecteurs de partage correspondant à des mots de code de $\mathcal{C}$. Plus précisément, on obtient le vecteur de partage $(c_1, c_2, c_3) \in E_s(\mathcal{C})$ en calculant dans un premier temps :

$$\begin{array}{rcl} (x_1, r_1) \cdot G & = & (x_1, r_1, x_1 w + r_1 w + r_1, x_1 w + r_1 w + x_1) \\ (x_2, r_2) \cdot G & = & (x_2, r_2, x_2 w + r_2 w + r_2, x_2 w + r_2 w + x_2) \end{array} \tag{4.30}$$

où $r_1, r_2 \in_R \mathbb{F}_{256}$ et $G$ est la matrice génératrice de $\mathcal{C}$ donnée par (A.7) en Appendix A.2. Ensuite le vecteur de partage $(c_1, c_2, c_3)$ est obtenu en sommant les précédents vecteurs de partage entre eux :

$$\left\{ \begin{array}{rcl} c_1 & \leftarrow & r_1 + r_2 \\ c_2 & \leftarrow & [x_1 w + r_1 w + r_1] + [x_2 w + r_2 w + r_2] \\ c_3 & \leftarrow & [x_1 w + r_1 w + x_1)] + [x_2 w + r_2 w + x_2] \end{array} \right. \tag{4.31}$$

De façon similaire, le nouveau vecteur de partage $(y_1, y_2, y_3) \in E_{s^2}(\mathcal{C})$ est obtenu en calculant :

$$\left\{ \begin{array}{rcl} y_1 & \leftarrow & r_3 + r_4 \\ y_2 & \leftarrow & [x_1^2 w + r_3 w + r_3] + [x_2^2 w + r_4 w + r_4] \\ y_3 & \leftarrow & [x_1^2 w + r_3 w + x_1^2)] + [x_2^2 w + r_4 w + x_2^2] \end{array} \right. \tag{4.32}$$

où $r_3, r_4 \in \mathbb{F}_{256}$.

L'étape suivante de l'algorithme consiste à effectuer successivement :

$$\begin{array}{rcl} (z_1, z_2, z_3) & \leftarrow & \mathsf{SecMult2}((c_1, c_2, c_3), (y_1, y_2, y_3)) \\ (w_1, w_2, w_3) & \leftarrow & (z_1^4, z_2^4, z_3^4) \\ (z_1, z_2, z_3) & \leftarrow & \mathsf{SecMult2}((z_1, z_2, z_3), (w_1, w_2, w_3)) \\ (z_1, z_2, z_3) & \leftarrow & (z_1^{16}, z_2^{16}, z_3^{16}) \\ (z_1, z_2, z_3) & \leftarrow & \mathsf{SecMult2}((z_1, z_2, z_3), (w_1, w_2, w_3)) \end{array} \tag{4.33}$$

où $\mathsf{SecMult2}$ réfère à l'algorithme 17.
Enfin, l'étape 13 de l'algorithme 23 retourne directement un vecteur de partage de $E_{s^{254}}(\mathcal{C}^\star)$ :

1. En effectuant le produit de Schur entre $(z_1, z_2, z_3)$ et $(y_1, y_2, y_3)$, ce qui donne $(z_1 y_1, z_2 y_2, z_3 y_3)$

2. En transcodant $(z_1y_1, z_2y_2, z_3y_3)$ en un vecteur de partage de $\mathcal{C}^\star$ :

$$(z_1, z_2) \leftarrow \mathsf{transcodage}_{(\mathcal{C}\to\mathcal{C}^\star)}\left(\mathbf{1}_3, (z_1y_1, z_2y_2, (z_3y_3))\right) \ . \tag{4.34}$$

Finalement la transformation affine de la transformation $SubBytes$ est effectuée comme une transformation stable en appliquant sur chaque coordonnée une table pré-calculée $A$ : $A(z_1), A(z_2)$, puis en additionnant à $A(z_1)$ la constant $b$. Le vecteur de partage retourné par l'algorithme 23 est $(A(z_1) + b, A(z_2))$ qui correspond bien à un vecteur de partage de $E_{sbox(s)}(\mathcal{C}^\star)$.

#### 4.3.3.3 Résistance en pratique

Pour mettre en évidence la mise en pratique de notre proposition donnée par l'algorithme 23, on a implanté sur un composant de carte à puce ayant un langage assembleur 8051 et un co-processeur 8 bits, la transformation $SubBytes$ avec le code de paramètres $[7, 3, 4]$ (pour résister aux attaques d'ordre 1) et avec le code de Hamming étendu de paramètres $[8, 4, 4]$ (pour résister aux attaques d'ordre 2) (cf. Table 4.2).

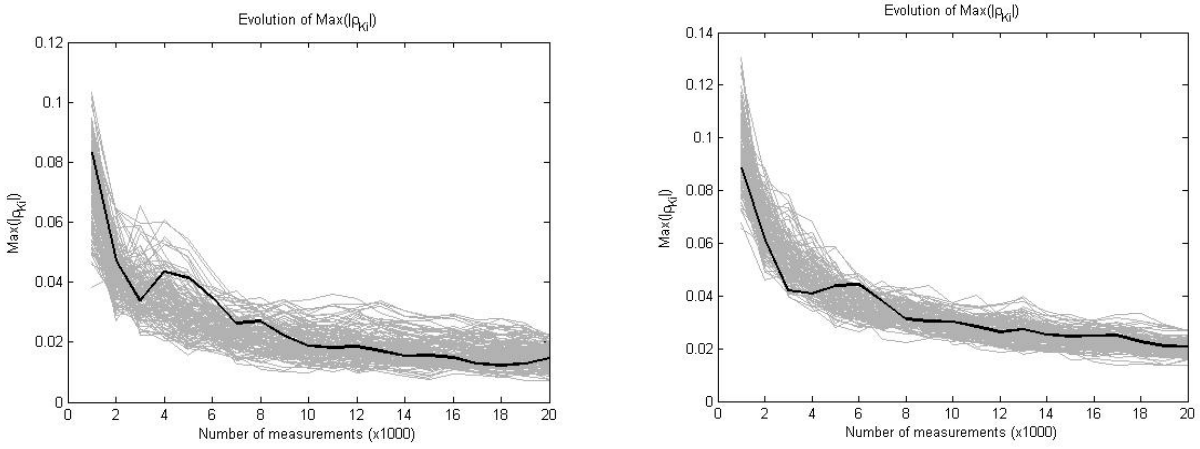



FIGURE 4.1 – Courbes de convergence résultant d'une CPA d'ordre 1 (à gauche) et d'une CPA d'ordre 2 à (droite) effectuées à partir de courbes de consommation simulées sans bruit

Ensuite, on a effectué une CPA d'ordre 1 (resp. une CPA d'ordre 2) ciblant un octet de la clé de chiffrement de l'algorithme résistant à l'ordre 1 (resp. à l'ordre 2) en utilisant des courbes de consommation simulées sans bruit supposant le modèle de fuite du composant lié au poids de Hamming. La figure 4.1 à gauche (resp. à droite) explicite la convergence des valeurs maximales obtenues en calculant le coefficient de Pearson pour les 256 hypothèses de sous-clé en fonction du nombre de courbes de consommation considérées lors d'une CPA d'ordre 1 (resp. d'ordre 2). En particulier, les courbes grises correspondent aux

255 mauvaises hypothèses de sous-clé, tandis que la courbe noire correspond à la bonne. Comme attendu, on peut voir que notre implantation utilisant le code de paramètres $[7, 3, 4]$ (resp. utilisant le code de Hamming étendu de paramètres $[8, 4, 4]$) résiste aux CPA d'ordre 1 (resp. aux CPA d'ordre 2).

## 4.4 Comparaison des différentes méthodes

Dans cette section, on compare les différentes méthodes présentées dans le chapitre 2 (cf. sous-section 2.4.2) ainsi que celles présentées dans ce chapitre pour implanter la transformation *SubBytes* de l'AES. La table 4.5 présente le coût de ces différentes méthodes pour un paramètre de sécurité $t$ fixé. Dans cette table :

- **aléa** indique le nombre d'éléments générés uniformément sur $\mathbb{F}_{256}$ (*i.e.,* le nombre d'octets générés), excepté aux lignes 2 et 4, où il est précisé le nombre de bits générés.
- **add/and** indique le nombre d'additions entre des éléments définis sur $\mathbb{F}_{256}$ (ou sur $\mathbb{F}_{2^4}$) et le nombre de multiplications bit à bit effectué sur un octet lorsque and est précisé.
- **mult** indique le nombre de multiplications entre des éléments définis sur $\mathbb{F}_{256}$ utilisant les tables pré-calculées alog, log.
- **table** indique le nombre d'accès à une table pré-calculée, comme par exemple la table qui retourne l'élévation au carré d'un élément défini sur $\mathbb{F}_{256}$.

L'implantation de l'article [CPRR13], décrite par l'algorithme 8 suggère l'utilisation de deux tables pré-calculées particulières : l'une retournant l'élévation à la puissance 3 d'un élément défini sur $\mathbb{F}_{256}$ et l'autre retournant l'élévation à la puissance 5. Cette solution permet d'éviter deux multiplications coûteuses entre des éléments définis sur $\mathbb{F}_{256}$, ce qui est donc plus efficace. Pour cette raison, on propose d'utiliser de telles tables dans l'implantation de l'algorithme 23. Les deux dernières lignes de la table 4.5 indique donc le coût de l'algorithme 23 lorsque l'on considère de telles tables pré-calculées.

D'après la table 4.5, on peut constater que les solutions décrites par l'algorithme 23 ont chacune une complexité de $\mathcal{O}(t)$ multiplications [4] contrairement aux solutions décrites par les algorithmes 8 et 9 qui ont une complexité de $\mathcal{O}(t^2)$ multiplications, et à celle décrite par l'algorithme 11 qui a une complexité de $\mathcal{O}(t^3)$ multiplications. Le coût de la solution décrite à la ligne 4 ne requiert quant à elle aucune multiplication sur $\mathbb{F}_{256}$, cependant cette dernière requiert plus d'accès à des tables pré-calculées que les autres méthodes. En supposant le coût des opérations autres que l'implantation d'une multiplication sur $\mathbb{F}_{256}$ et l'accès à des tables comme négligeables, la solution décrite par l'algorithme 23 est donc asymptotiquement plus pertinente que les solutions décrites dans [CPRR13, GPQ11b, GM11, PR11b] et la celle décrite dans la sous-section 4.2.

4. Le paramètre $n$ est linéaire en $t$.

| | aléa | add/and | mult $\mathbb{F}_{256}$ | table |
|---|---|---|---|---|
| Schéma de partage de secret basique : Alg.8+ rel.(2.44) [CPRR13] | $3t^2+3t$ | $14t^2+12t-1$ | $2t^2+4t+2$ | $4t^2+14t+10$ |
| Schéma de partage multiplicatif : Alg.9+ rel.(2.44) [GPQ11b] | $\frac{7t^2+7t}{8}$ bits $4t^2+4t$ octets | $\frac{60t^2+77t+25}{8}$ add $\frac{14t^3+28t^2+14t}{8}$ and | $2t^2+8t$ | $2t+2$ |
| Schéma de partage de Shamir : Alg.11+rel.(2.67) [GM11, PR11b] | $8t^2+4t$ | $8t^3+28t^2+20t+8$ | $8t^3+20t^2+35t+15$ | $14t+14$ |
| Variante du schéma de partage de Shamir : cf. sous-section 4.2 | $(8t^2+24t+4)$ demi-octets | $8t^3+72t^2+148t+80$ | - | $8t^3+56t^2+156t+120$ |
| Schéma construit à partir de codes auto-duaux binaires : Alg.23 + Code $\mathbb{F}_2$ | $5nt+5(k-1)t+5k-9$ | $11nt+(5L-1)t+5L-3$ | $4n$ | $7n+t+1$ |
| Schéma construit à partir de codes auto-duaux définis sur $\mathbb{F}_4$ : Alg.23 + Code $\mathbb{F}_4$ | $5nt+6(k-1)t+6k-10$ | $12nt+(6L-1)t+6L+-3$ | $4n$ | $6n+(6L'+2)t+6L'+2$ |
| Schéma construit à partir de codes auto-duaux binaires : Alg.23 + Code $\mathbb{F}_2$ (avec tables de puissance 3 et 5) | $5nt+5(k-1)t+5k-9$ | $11nt+(5L-1)t+5L-3$ | $2n$ | $9n+t+1$ |
| Schéma construit à partir de codes auto-duaux définis sur $\mathbb{F}_4$ : Alg.23 + Code $\mathbb{F}_4$ (avec tables de puissance 3 et 5) | $5nt+6(k-1)t+6k-10$ | $12nt+(6L-1)t+6L-3$ | $2n$ | $8n+(6L'+2)t+6L'+2$ |

TABLE 4.5 – Coût des implantations de la transformation *SubBytes* avec un paramètre de sécurité $t$

Pour estimer le coût des différentes solutions pour des petites valeurs de $t$ (pour $t = 1, \ldots, 6$), une estimation en cycles de cette table est présentée par la table 4.6. Pour ces estimations, on a considéré les implantations sur un processeur 8 bits où :

- la génération de 4 bits aléatoires requiert 1 cycle,
- l'addition ou la multiplication bit à bit entre deux éléments de $\mathbb{F}_{256}$ requiert 1 cycle,
- l'implantation d'une multiplication définie sur $\mathbb{F}_{256}$ requiert 20 cycles,
- et l'accès à des tables pré-calculées requiert 3 cycles.

| Choix du schéma de partage de secret | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 |
|---|---|---|---|---|---|---|
| Schéma de partage de secret basique : Alg.8+ rel.(2.44) [CPRR13] | 281 | 637 | 1137 | 1781 | 2569 | 3501 |
| Schéma de partage multiplicatif : Alg.9+ rel.(2.44) [GPQ11b] | 260 | 642 | 1168 | 1847 | 2690 | 3707 |
| Schéma construit à partir de codes auto-duaux binaires : Alg.23 + Code $\mathbb{F}_2$ (avec tables de puissance 3 et 5) | 816 | 1207 | 4598 | 5485 | 7054 | 8515 |
| Schéma construit à partir de codes auto-duaux définis sur $\mathbb{F}_4$ : Alg.23 + Code $\mathbb{F}_4$ (avec tables de puissance 3 et 5) | 402 | - | 2462 | 3487 | 7184 | - |
| Variante du schéma de partage de Shamir : cf. sous-section 4.2 | 1364 | 2972 | 5460 | 9020 | 13.844 | 20.124 |
| Schéma de partage de Shamir : Alg.11+rel.(2.67) [GM11, PR11b] | 1732 | 5010 | 11.192 | 21.286 | 36.300 | 57.242 |

TABLE 4.6 – Application numérique des transformation *SubBytes*

D'après la table 4.6, on peut voir que la solution utilisant la variante du schéma de partage de secret de Shamir (cf. ligne 5) est plus efficace que la solution proposée dans [GM11, PR11b] dès $t = 1$. Les méthodes basées sur l'utilisation de codes auto-duaux (ou faiblement auto-duaux) décrites par l'algorithme 23 (cf. lignes 3 et 4) sont quant à elles moins coûteuses que ces deux solutions. Cependant, elles sont plus coûteuse que les méthodes proposées par l'utilisation du schéma de partage de secret basique (cf. [CPRR13]) et celui multiplicatif (cf. [GPQ11b]). Néanmoins, on peut noter pour $t = 1$ que l'implantation de l'AES à l'aide du code auto-dual construit à partir de la matrice génératrice du code `XQR`(3) est compétitive comparée à ces deux solutions.

## 4.5 Conclusion

Dans ce chapitre, on a présenté deux nouvelles solutions pour implanter un AES résistant aux attaques d'ordre $t$. Ces solutions sont basées sur l'utilisation de schémas de partage de secret construits à partir de codes linéaires non MDS. La première solution décrite dans la sous-section 4.2, est une alternative au schéma de partage de secret de Shamir utilisant un code d'évaluation proche du code de Reed-Solomon. La seconde solution décrite par l'algorithme 23, repose quant à elle sur l'utilisation d'un code auto-dual (ou faiblement auto-dual) construit à partir d'une matrice génératrice à coefficient sur $\mathbb{F}_2$ ou $\mathbb{F}_4$. Ces deux solutions sont à la fois résistantes aux attaques d'ordre $t$, mais aussi plus efficaces que la solution proposée récemment dans [GM11, PR11b]. De plus, la solution basée sur l'utilisation de codes auto-duaux (ou faiblement auto-duaux) décrites par l'algorithme 23 a une complexité en $\mathcal{O}(t)$ multiplications, contrairement aux solutions décrites dans [CPRR13, GPQ11b] qui ont une complexité en $\mathcal{O}(t^2)$ multiplications. Asymptotiquement, la solution décrite à l'aide de code auto-duaux (ou faiblement auto-duaux) est donc la plus pertinente. Cependant pour des petites valeurs de $t$, cette dernière est plus coûteuse du fait du nombre conséquent d'addition et de génération de nombre aléatoire requis. Néanmoins, pour $t = 1$, l'utilisation d'un code auto-dual construit à partir d'une matrice génératrice à coefficient sur $\mathbb{F}_4$ est compétitive comparée aux solutions décrites dans [CPRR13, GPQ11b].

# Conclusion et Perspectives

De nos jours, les cryptosystèmes implantés sur carte à puce doivent faire face à deux grandes catégories d'attaques : les attaques par canaux cachés et les attaques par injection de fautes. Pour s'en prémunir, des contre-mesures sont alors élaborées, puis validées en considérant un modèle d'attaquant bien défini. Dans ce manuscrit, nous nous sommes intéressés à la protection des cryptosystèmes symétriques contre les attaques par canaux cachés et plus précisément contre les attaques statistiques d'ordre supérieur. Pour ce faire, nous avons rappelé dans un premier temps l'analogie entre les contre-mesures de masquage et les schémas de partage de secret décrits dans le contexte du calcul multi-parties. Puis notre étude a porté sur la construction de schémas de partage de secret à partir de codes linéaires, introduites par James L. Massey en 1993. En adaptant cette solution dans le contexte des attaques par canaux cachés, nous avons proposé l'élaboration d'une méthode générique de contre-mesures de masquage à l'ordre $t + 1$ construit à partir de codes linéaires, ainsi que décrit des procédures de calculs sécurisées dans ce contexte. L'originalité de cette construction repose sur les propriétés inhérentes à certains codes permettant de construire des contre-mesures de masquage adéquates en fonction des opérations à effectuer. Une telle méthode permettrait donc de déterminer les contre-mesures de masquage les plus appropriées en fonction des cryptosystèmes à implanter.

Dans notre étude, nous nous sommes focalisés sur l'utilisation de codes linéaires dans le contexte des attaques par canaux cachés. Cependant, un code linéaire est un cas particulier de code correcteur dont les objectifs sont principalement la détection et la correction d'erreurs. En considérant la propriété de détection, le code linéaire utilisé pour une contre-mesure de masquage pourrait également permettre de se prémunir contre des attaques par injection de fautes. Ainsi, une étude approfondie dans cette direction permettrait de tirer profit de la contre-mesure de masquage à la fois pour se prémunir des attaques statistiques d'ordre supérieur mais aussi des attaques par injection de fautes. Une telle solution pourrait permettre de réduire le coût imputé par l'ajout de contre-mesures qui généralement sont élaborées indépendamment en fonction de l'attaque à contrecarrer.

Dans cette même optique, un autre axe de recherche serait de tirer profit de l'utilisation de schémas de partage de secret construit à partir de codes correcteurs pour implanter des cryptosystèmes asymétriques. Par exemple l'utilisation des codes des restes

chinois permettrait de protéger les cryptosystèmes asymétriques, tels que la signature RSA ou les cryptosystèmes basés sur l'utilisation des courbes elliptiques, des attaques statistiques, mais aussi des attaques par injection de fautes. Par ailleurs, on peut noter que le code des restes chinois correspond à la représentation RNS (*Residue Number System*) qui permet une arithmétique efficace basée sur le théorème des restes chinois. Des travaux étudiant l'efficacité d'une telle solution face aux attaques statistiques ont déjà été mené, par exemple dans le cadre du RSA [CNPQ04]. En considérant la représentation RNS en termes de codes correcteurs, on pourrait envisager de tirer profit de la propriété de détection et ainsi d'utiliser une telle contre-mesure également dans le contexte des attaques par injection de fautes.

# Annexe A

# Appendice

## A.1 Code auto-dual (ou faiblement auto-dual) binaire

Dans cette section, nous donnons les matrices génératrices des codes auto-duaux (ou faiblement auto-duaux) binaires citées dans la table 4.2. Ces matrices peuvent être trouvées par exemple dans [CS90, GO03].

**Matrice génératrice du code $[7, 3]$ (avec $d^{\perp} = 3$)**

$$\left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right) \tag{A.1}$$

**Matrice génératrice du code de Hamming étendu $[8, 4, 4]$ (avec $d^{\perp} = 4$) :**

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right) \tag{A.2}$$

**Matrice génératrice du code $\mathcal{C}_{21}$ $[21, 10]$ (avec $d^{\perp} = 5$) :**

$$\left(\begin{array}{c|cccccccccc} & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ Id_{10} & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right) \tag{A.3}$$

**Matrice génératrice du code de Golay raccourci $[22, 11, 6]$ (avec $d^{\perp} = 6$) :**

$$\left(\begin{array}{c|cccccccccc} & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ Id_{11} & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{array}\right) \tag{A.4}$$

**Matrice génératrice du code de Golay $[23, 11]$ (avec $d^{\perp} = 7$) :**

$$\left(\begin{array}{c|cccccccccccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & \\ & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ Id_{11} & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right) \tag{A.5}$$

**Matrice génératrice du code de Golay étendu $[24, 12, 8]$ (avec $d^{\perp} = 8$) :**

$$\left(\begin{array}{c|cccccccccccc} & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ Id_{12} & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right) \tag{A.6}$$

## A.2 Code auto-dual (ou faiblement auto-dual) défini sur $\mathbb{F}_4$

Dans cette section nous donnons des matrices génératrices des codes auto-duaux (ou faiblement auto-duaux) définis sur $\mathbb{F}_4 = \{0, 1, w, w+1\}$ citées dans la table 4.3. Ces matrices peuvent être trouvées dans [GO03].

**Matrice génératrice du code de résidus quadratiques étendu** $\texttt{XQR}(3)$ $[4, 2, 3]$ **(avec** $d^\perp = 3$**) :**

$$\left(\begin{array}{cc|cc} 1 & 0 & w & w+1 \\ 0 & 1 & w+1 & w \end{array}\right) \tag{A.7}$$

**Matrice génératrice du code** $[11, 5]$ **(avec** $d^\perp = 5$**) :**

$$\left(\begin{array}{c|cccccc} & 1 & w & 1 & 1 & w+1 & 1 \\ & w+1 & 0 & 1 & w & w+1 & w \\ & w+1 & w & w+1 & w & 0 & 1 \\ Id_5 & 0 & w+1 & w & w+1 & w & 1 \\ & w & w+1 & 1 & 0 & w & w+1 \end{array}\right) \tag{A.8}$$

**Matrice génératrice du code de résidus quadratiques étendu** $\texttt{XQR}(11)$ $[12, 6, 6]$ **(avec** $d^\perp = 6$**) :**

$$\left(\begin{array}{c|cccccc} & 1 & w & 1 & 1 & w+1 & 1 \\ & w+1 & 0 & 1 & w & w+1 & w \\ & w+1 & w & w+1 & w & 0 & 1 \\ Id_6 & 0 & w+1 & w & w+1 & w & 1 \\ & w & w+1 & 1 & 0 & w & w+1 \\ & w & 1 & 1 & w+1 & 1 & 1 \end{array}\right) \tag{A.9}$$

**Matrice génératrice du code** $[19, 9]$ **(avec** $d^\perp = 7$**) :**

$$\left(\begin{array}{c|cccccccccc} & 1 & w & w+1 & w+1 & 0 & 1 & 0 & w & w & w+1 \\ & w+1 & 0 & 0 & 1 & w+1 & w+1 & 1 & 1 & w+1 & 0 \\ & 0 & w+1 & 0 & 0 & 1 & w+1 & w+1 & 1 & 1 & w+1 \\ & w+1 & 1 & 1 & w & 0 & w & w+1 & w & 0 & w+1 \\ Id_9 & w+1 & w & w+1 & w+1 & w & w+1 & w & w & w+1 & w \\ & w & 0 & w+1 & w & w+1 & 0 & w+1 & 1 & 1 & w \\ & w & 1 & 1 & w & w & 1 & 0 & 0 & w & 0 \\ & 0 & w & 1 & 1 & w & w & 1 & 0 & 0 & w \\ & w & w+1 & w+1 & 0 & 1 & 0 & w & w & w+1 & 1 \end{array}\right) \tag{A.10}$$

# Bibliographie

[AARR02] D. Agrawal, B. Archambeault, J.R. Rao, and P. Rohatgi. The EM Side-Channel(s). In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 29–45. Springer, 2002.

[ABG04] Mehdi-Laurent Akkar, R. Bévan, and L. Goubin. Two Power Analysis Attacks against One-Mask Method. In B. Roy and W. Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *LNCS*, pages 332–347. Springer, 2004.

[AcKKS07] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting Secret Keys Via Branch Prediction. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *LNCS*, pages 225–242. Springer, 2007.

[AFV07] F. Amiel, B. Feix, and K. Villegas. Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography – SAC 2007*, LNCS, pages 110–125. Springer, 2007.

[AG03] Mehdi-Laurent Akkar and L. Goubin. A Generic Protection against High-order Differential Power Analysis. In T. Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *LNCS*, pages 192–205. Springer, 2003.

[ANS05] ANSI X9.62–2005. *Public Key Cryptography for The Financial Service Industry : The Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standards Institute, November 2005.

[APSQ06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template Attacks in Principal Subspaces. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *LNCS*, pages 1–14. Springer, 2006.

[BCO03] É. Brier, C. Clavier, and F. Olivier. Optimal Statistical Power Analysis. Cryptology ePrint Archive, Report 2003/152, 2003.

[BCO04] É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic*

*Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[Bév04] R. Bévan. *Estimation statistique et sécurité des cartes à puce – Évaluation d'attaques DPA évoluées*. PhD thesis, Supelec, June 2004.

[BFGV12] J. Balasch, S. Faust, B. Gierlichs, and I. Verbauwhede. Theory and practice of a leakage resilient masking scheme. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 758–775. Springer, 2012.

[BGLT06] M. Bucci, L. Giancane, R. Luzzi, and A. Trifiletti. Three-Phase Dual-Rail Pre-charge Logic. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *LNCS*, pages 232–241. Springer, 2006.

[BK02] R. Bévan and E. Knudsen. Ways to Enhance Power Analysis. In P.J. Lee and C.H. Lim, editors, *Information Security and Cryptology – ICISC 2002*, volume 2587 of *LNCS*, pages 327–342. Springer, 2002.

[Bla79] G.R. Blakley. Safeguarding cryptographic keys. In *National Comp. Conf.*, volume 48, pages 313–317, New York, June 1979. AFIPS Press.

[BLV13] Alexis Bonnecaze, Pierre Liardet, and Alexandre Venelli. AES side-channel countermeasure using random tower field constructions. *Des. Codes Cryptography*, 69(3) :331–349, 2013.

[BMK04] J. Blömer, J. G. Merchan, and V. Krummel. Provably Secure Masking of AES. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *LNCS*, pages 69–83. Springer, 2004.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Symposium on Theory of Computing*, pages 1–10, 1988.

[BS99] E. Biham and A. Shamir. Power Analysis of the Key Scheduling of the AES Candidates. In *Second AES Candidate Conference – AES 2*, March 1999.

[CC06] H. Chen and R. Cramer. Algebraic Geometric Secret Sharing Schemes and Secure Multi-Party Computations over Small Fields. In S. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *LNCS*, pages 521–536. Springer, 2006.

[CCCX09] I. Cascudo, H. Chen, R. Cramer, and C. Xing. Asymptotically Good Ideal Linear Secret Sharing with Strong Multiplication over Any Fixed Finite Field. In S. Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *LNCS*, pages 466–486. Springer, 2009.

[CCD88] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. *Symposium on Theory of Computing*, pages 11–19, 1988.

[CCD00] C. Clavier, J.-S. Coron, and N. Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 252–263. Springer, 2000.

[CCG+07] H. Chen, R. Cramer, S. Goldwasser, R. de Haan, and V. Vaikuntanathan. Secure computation from random error correcting codes. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2007*, volume 7237 of *LNCS*, pages 291–310. Springer, 2007.

[CDM00] R. Cramer, I. Damgard, and U. Maurer. General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 316–334. Springer, 2000.

[CFG+11] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Square Always Exponentiation. In D. J. Bernstein and S. Chatterjee, editors, *INDOCRYPT 2011 - 12th International Conference on Cryptology in India*, volume 7107 of *LNCS*, pages 40–57. Springer, 2011.

[CGP+12a] C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In A. Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, 2012.

[CGP+12b] J.-S. Coron, C. Giraud, E. Prouff, S. Renner, M. Rivain, and P. K. Vadnala. Conversion of Security Proofs from One Model to Another : A New Issue. In Werner Schindler and Sorin Huss, editors, *Second International Workshop on Constructive Side-Channel Analysis and Secure Design – COSADE 2012*, LNCS, pages 69–81. Springer, 2012.

[CH12] Jong-Won Cho and Dong-Guk Han. Security Analysis of the Masking-Shuffling based Side-Channel Attack Countermeasures . *International Journal of Security and Its Applications*, 08(4) :207–213, October 2012.

[CJRR99a] S. Chari, C.S. Jutla, J.R. Rao, and P. Rohatgi. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In *Second AES Candidate Conference – AES 2*, March 1999.

[CJRR99b] S. Chari, C.S. Jutla, J.R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M.J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An Efficient Method for Random Delay Generation in Embedded Software. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *LNCS*, pages 156–170. Springer, 2009.

[CM04] Benoît Chevallier-Mames. Self-Randomized Exponentiation Algorithms. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *LNCS*, pages 236–249. Springer, 2004.

[CMCJ04] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost Solutions for Preventing Simple Side-Channel Analysis : Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6) :760–768, 2004.

[CNPQ04] Mathieu Ciet, Michael Neve, Eric Peeters, and Jean-Jacques Quisquater. Parallel FPGA Implementation of RSA with Residue Number Systems - Can side-channel threats be avoided ? - Extended version. Cryptology ePrint Archive, Report 2004/187, 2004.

[Cor99] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.

[CPR07] J.-S. Coron, E. Prouff, and M. Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *LNCS*, pages 28–44. Springer, 2007.

[CPR12] J.-S. Coron, E. Prouff, and T. Roche. On the use of shamir's secret sharing against side-channel analysis . In Stefan Mangard, editor, *Smart Card Research and Advanced Applications, 11th International Conference – CARDIS 2012*, LNCS. Springer, 2012.

[CPRR13] Jean-Sebastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-Order Side Channel Security and Mask Refreshing. In *Fast Software Encryption – FSE 2013*, 2013.

[CRR02] S. Chari, J.R. Rao, and P. Rohatgi. Template Attacks. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 13–29. Springer, 2002.

[CRZ13] Guilhem Castagnos, Soline Renner, and Gilles Zémor. High-order Masking by Using Coding Theory and Its Application to AES. In *IMA International Conference on Cryptography and Coding*, pages 193–212, 2013.

[CS90] J.H. Conway and N.J.A. Sloane. A new upper bound on the minimal distance of self-dual codes. *IEEE Transactions on Information Theory*, 36(6) :1319–1333, 1990.

[CZ06] Z. Chen and Y. Zhou. Dual-Rail Random Switching Logic : A Countermeasure to Reduce Side Channel Leakage. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *LNCS*, pages 242–254. Springer, 2006.

[dBLW02] B. den Boer, K. Lemke, and G. Wicke. A DPA Attack against the Modular Reduction within a CRT Implementation of RSA. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 228–243. Springer, 2002.

[DF12] S. Dziembowski and S. Faust. Leakage-resilient circuits without computational assumptions. In R. Cramer, editor, *Theory of Cryptography Conference – TCC 2012*, volume 7194 of *LNCS*. Springer, 2012.

[DKL+00] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the Timing Attack. In J.-J. Quisquater, editor, *CARDIS 1998*, volume 1820 of *LNCS*, pages 175–190. Springer, 2000.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, 2002.

[FIP77] FIPS PUB 46. *The Data Encryption Standard*. National Bureau of Standards, January 1977.

[FIP99] FIPS PUB 46-3. *Data Encryption Standard (DES)*. National Institute of Standards and Technology, October 1999.

[FIP01] FIPS PUB 197. *Advanced Encryption Standard*. National Institute of Standards and Technology, November 2001.

[FIP07] FIPS PUB 180-3. *Secure Hash Standard*. National Institute of Standards and Technology, June 2007. Draft.

[FMPR10] 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 – SAC 2010*, volume 6544 of *LNCS*, pages 262–280. Springer, 2010.

[GM11] L. Goubin and A. Martinelli. Protecting AES with Shamir's Secret Sharing Scheme. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*, pages 79–94. Springer, 2011.

[GMO01] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis : Concrete Results. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GO03] P. Gaborit and A. Otmani. Experimental constructions of self-dual codes. *Finite fields and their applications-Elsevier*, July 2003.

[GP99] L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.

[GPQ10] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Secure multiplicative masking of power functions. In J. Zhou and M. Yung, editors, *Applied Cryptography and Network Security - 8th International Conference, ACNS 2010*, volume 6123 of *LNCS*, pages 200–217. Springer, 2010.

[GPQ11a] L. Genelle, E. Prouff, and M. Quisquater. Montgomery's Trick and Fast Implementation of Masked AES. In A. Nitaj and D. Pointcheval, editors, *AFRICACRYPT 2011*, volume 6737 of *LNCS*, pages 153–169. Springer, 2011.

[GPQ11b] L. Genelle, E. Prouff, and M. 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*, volume 6917 of *LNCS*, pages 240–255. Springer, 2011.

[GRR98] R. Gennaro, M. Rabin, and T. Rabin. Simplifed vss and fact-track multiparty computations with applications to threshold cryptography. *Symposium on Principles of Distributed Computing*, pages 101–111, 1998.

[GSF13] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. Multiparty Computation : How Large Is the Gap for AES ? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *LNCS*, pages 400–416. Springer, 2013.

[HH98] Helena Handschuh and Howard M. Heys. A timing attack on rc5. In *Selected Areas in Cryptography*, pages 306–318, 1998.

[HP06] Helena Handschuh and Bart Preneel. Blind Differential Cryptanalysis for Enhanced Power Attacks. In Eli Biham and Amr Youssef, editors, *Selected Areas in Cryptography – SAC 2006*, LNCS, pages 158–169. Springer, 2006.

[ISO97] ISO/IEC 7816-3. *Information Technology – Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 3 : Electronic Signals and Transmission Protocols*, 1997.

[ISO98] ISO/IEC 7816-1. *Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 1 : Physical Characteristics*, 1998.

[ISO99] ISO/IEC 7816-2. *Information Technology – Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 2 : Dimensions and Location of the Contacts*, 1999.

[ISO03] ISO/IEC 7810. *Identification Cards – Physical Characteristics*, troisième edition, November 2003.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits : Securing Hardware against Probing Attacks. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

[JPS05] M. Joye, P. Paillier, and B. Schoenmakers. On Second-order Differential Power Analysis. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 293–308. Springer, 2005.

[JY02] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer, 2002.

[KHL11] H. Kim, S. Hong, and J. 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*, volume 6917 of *LNCS*, pages 95–107. Springer, 2011.

[KJJ98] P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc., 1998.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M.J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[Koc96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[LH05] Jiqiang Lv and Yongfei Han. Enhanced DES Implementation Secure against High-order Differential Power Analysis in Smartcards. In Colin Boyd and Juan Manuel González Nieto, editors, *Information Security and Privacy - 10th Australasian Conference – ACISP 2005*, volume 3574 of *LNCS*, pages 195–206. Springer, 2005.

[Liu] Chung Liu. Introduction to Combinatorial Mathematics. page 1968.

[Man02] S. Mangard. A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion. In P.J. Lee and C.H. Lim, editors, *Information Security and Cryptology – ICISC 2002*, volume 2587 of *LNCS*, pages 343–358. Springer, 2002.

[Man04] Stefan Mangard. Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *LNCS*, pages 222–235. Springer, 2004.

[Mas93] J. Massey. Minimal codewords and secret sharing. In *Sixth Joint Swedish-Russian Workshop on Information Theory*, pages 246–249, 1993.

[MDS99a] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *the USENIX Workshop on Smartcard Technology (Smartcard '99)*, pages 151–161, 1999.

[MDS99b] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcard. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *LNCS*, pages 144–157. Springer, 1999.

[MDS02] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Examining Smart-Card Security under the Threat of Power Analysis Attacks. *IEEE Transactions on Computers*, 51(5), May 2002.

[Mes00a] T.S. Messerges. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*. PhD thesis, University of Illinois, 2000.

[Mes00b] T.S. Messerges. Securing the AES Finalists against Power Analysis Attacks. In B. Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *LNCS*, pages 150–164. Springer, 2000.

[Mes00c] T.S. Messerges. Using Second-order Power Analysis to Attack DPA Resistant Software. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000.

[Mon87] P.L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48 :243–264, 1987.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks – Revealing the Secrets of Smartcards*. Springer, 2007.

[MR02] S. Murphy and M. Robshaw. Essential algebraic structure within the AES. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *LNCS*, pages 1–16. Springer, 2002.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*. North-Holland Publishing Co., 1977. North-Holland Mathematical Library, Vol. 16.

[MS81] Robert J. McEliece and Dilip V. Sarwate. On Sharing Secrets and Reed-Solomon Codes. volume 24, pages 583–584, 1981.

[MvOV97] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[Nov02] R. Novak. SPA-Based Adaptive Chosen-Ciphertext Attack on RSA Implementation. In D. Naccache and P. Paillier, editors, *Public Key Cryptography – PKC 2002*, volume 2274 of *LNCS*, pages 252–262. Springer, 2002.

[OM07] Elisabeth Oswald and Stefan Mangard. Template Attacks on Masking—Resistance is Futile. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *LNCS*, pages 243–256. Springer, 2007.

[OMHT06] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical Second-order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *LNCS*, pages 192–207. Springer, 2006.

[OMP04] Elisabeth Oswald, Stefan Mangard, and Norbert Pramstaller. Secure and Efficient Masking of AES – A Mission Impossible ? Cryptology ePrint Archive, Report 2004/134, 2004.

[OS06] Elisabeth Oswald and Kai Schramm. An Efficient Masking Scheme for AES Software Implementations. In J. Song, T. Kwon, and M. Yung, editors, *WISA 2005*, volume 3786 of *LNCS*, pages 292–305. Springer, 2006.

[Osw02] Elisabeth Oswald. Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 82–97. Springer, 2002.

[OT04] K. Okeya and T. Takagi. Security Analysis of CRT-based Cryptosystems. In J. Zhou, editor, *Applied Cryptography and Network Security – ANCS 2004*, volume 3089 of *LNCS*, pages 383–397. Springer, 2004.

[Pag02] D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Cryptology ePrint Archive, Report 2002/169, 2002.

[PM05] Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-charge Logic : DPA-Resistance Without Routing Constraints. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 175–186. Springer, 2005.

[PR08] Emmanuel Prouff and Matthieu Rivain. A Generic Method for Secure SBox Implementation. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *WISA 2007*, volume 4867 of *LNCS*, pages 227–244. Springer, 2008.

[PR11a] Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the aes using secure multi-party computation protocols. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*, pages 63–78. Springer, 2011.

[PR11b] Emmanuel Prouff and Thomas Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. *IACR Cryptology ePrint Archive*, 2011 :413, 2011.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Régis Bévan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Comput.*, 58(6) :799–811, 2009.

[PRR13] Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. On the Practical Security of a Leakage Resilient Masking Scheme. *IACR Cryptology ePrint Archive*, 2013 :396, 2013.

[PSQ07] E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Power and Electromagnetic Analysis : Improved Model, Consequences and Comparisons. *Integration*, 40(1) :52–60, 2007.

[QS01] Jean-Jacques Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA) : Measures and Countermeasures for Smart Cards. In I. Attali and T. Jensen, editors, *Smart Card Programming and Security – E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.

[RBJ+01] Atri Rudra, Pradeep K. Bubey, Charanjit S. Jutla, Vijay Kumar, Josyula Rao, and Pankay Rohatgi. Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 171–184. Springer, 2001.

[RDP08] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In T. Baignères and S. Vaudenay, editors, *Fast Software Encryption – FSE 2008*, volume 5086 of *LNCS*, pages 127–143. Springer, 2008.

[RO04] Christian Rechberger and Elisabeth Oswald. Practical Template Attacks. In C. H. Lim and M. Yung, editors, *WISA 2004*, volume 3325 of *LNCS*, pages 443–457. Springer, 2004.

[RP10] M. Rivain and E. Prouff. Provably Secure Higher-order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order Masking and Shuffling for Software Implementations of Block Ciphers. Cryptology ePrint Archive, 2009.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2) :120–126, 1978.

[Sha79] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11) :612–613, November 1979.

[Sha00] A. Shamir. Protecting smart cards from passive power analysis with detached power supplies. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 71–77. Springer, 2000.

[SMBY04] Danil Sokolov, Julian Murphy, Alex Bystrov, and Alex Yakovlev. Improving the Security of Dual-Rail Circuits. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 282–297. Springer, 2004.

[SMTM01] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact rijndael hardware architecture with s-box optimization. In E. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 239–254. Springer, 2001.

[SP06] Kai Schramm and Christof Paar. Higher Order Masking of the AES. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225. Springer, 2006.

[Sti95] D.R. Stinson. *Cryptography – Theory and Practice*. CRC Press, 1995.

[TB07] Michael Tunstall and Olivier Benoît. Efficient Use of Random Delays in Embedded Software. In Damien Sauveron, Konstantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, editors, *Information Security Theory and Practices – WISTP 2007*, volume 4462 of *LNCS*, pages 27–38. Springer, 2007.

[TSS+03] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. In C.D. Walter, Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 62–77. Springer, 2003.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks : A comprehensive study with cautionary note. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 740–757. Springer, 2012.

[WW04] J. Waddle and D. Wagner. Toward Efficient Second-order Power Analysis. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 1–15. Springer, 2004.

[Yao82] A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

[Yao86] A. C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.