



Implémentation d'une contre-mesure face à une attaque par analyse de la consommation de puissance d'un circuit intégré de chiffrement AES

Travail de fin d'étude présenté par
Thomas ANIZET
En vue de l'obtention du diplôme de
Master en Sciences de l'Ingénieur Industriel orientation Electronique

Abstract

Imaginez que de simples analyses de paramètres physiques (température, consommation de puissance, rayonnement électromagnétique) permettent de révéler les secrets de votre carte d'identité ou votre carte bancaire ? Ceci est aujourd'hui possible depuis la découverte fin des années 1990 d'un nouveau type d'attaque ciblant les algorithmes de chiffrement : les attaques par canaux auxiliaires. À ce jour, il existe une multitude d'algorithmes de chiffrement. L'algorithme *AES (Advanced Encryption Standard)* est sans nul doute le plus réputé et le plus répandu, de nos jours, dans la majeure partie des systèmes embarqués comportant des applications en sécurité. L'attaque par *l'analyse de la consommation de puissance* est un exemple parmi d'autres d'attaques par canaux auxiliaires d'un système implémentant l'algorithme AES. Cette attaque analyse la consommation de puissance du device cryptographique lorsque celui-ci chiffre les données. En effet, la puissance consommée par le circuit intégré reflète directement ses activités internes (instructions exécutées et données manipulées). Depuis ces découvertes, la sécurité matérielle a pris une tournure particulière. C'est désormais sur un nouvel axe de recherche, s'écartant des sentiers traditionnels, que se porte la problématique de protection des données sensibles.

Étant un acteur majeur dans le domaine de la Défense, Thales collabore étroitement avec de nombreux gouvernements. La cybersécurité est au centre des préoccupations actuelles. Le besoin en implantations sécurisées se fait de plus en plus ressentir. Il est nécessaire de se prémunir contre les attaques classiques mais également contre les attaques plus bas niveau exploitant des caractéristiques physiques. L'attaque ciblée dans ce *Travail de Fin d'Étude* (TFE) concerne l'analyse de la consommation de puissance. Mon objectif est le suivant : dans un premier temps, réaliser une série de démonstrateurs permettant de mettre en évidence l'impact des failles non traitées. Dans un second temps, développer une contre-mesure et justifier son gain en sécurité.

Remerciements

Je tiens à remercier toutes les personnes qui ont permis à ce travail de voir le jour, mais qui m'ont également soutenu tout au long de ces 5 années d'étude à l'ECAM.

Je remercie *Thales Belgium Communications & Security* de m'avoir permis de réaliser mon stage et mon TFE.

Je remercie tout particulièrement Monsieur Liran Lerman, mon promoteur, de m'avoir proposé ce projet très intéressant et qui fut très enrichissant. Je le remercie pour son support précieux, sa disponibilité mais également pour ses conseils et ses encouragements quotidiens.

Je remercie Monsieur François Durvaux, pour ses précieux conseils et ses connaissances avancées dans le domaine des attaques par canaux auxiliaires. Il m'a permis d'avancer rapidement dans mon travail.

Je remercie Monsieur Arnaud, pour son temps précieux consacré à m'apprendre de nouveaux concepts sur la programmation orientée hardware.

Je remercie également mademoiselle Clémence Flémal, ma tutrice, qui m'a accompagné dans la réalisation de ce travail et qui m'a consacré son temps et son attention durant la période de stage et de TFE.

Cahier des charges relatif au travail de fin d'études de

Thomas ANIZET, inscrit en 2^{ème} Master, orientation électronique

- Année académique : 2018-2019

- Titre provisoire : Contre-mesure pour les attaques par canaux cachés

- Objectifs à atteindre :

Étant un acteur majeur dans le domaine de la *Défense*, Thales collabore étroitement avec de nombreux gouvernements. La *cybersécurité* est au centre des préoccupations actuelles. Le besoin en implémentations sécurisées se fait donc de plus en plus ressentir. L'objectif est de se prémunir contre des attaques classiques et des attaques plus bas niveau exploitant des caractéristiques physiques. Afin de se familiariser avec ce domaine, l'étudiant devra réaliser une recherche bibliographique de règles de bonnes pratiques pour de la programmation sécurisée hardware. Après en avoir étudié les tenants et aboutissants, l'étudiant réalisera *(i) une série de démonstrateurs permettant de mettre en évidence l'impact des failles non traitées et (ii) le gain en sécurité dû aux contre-mesures.*

- Principales étapes :

- Recherches bibliographiques de règles de bonnes pratiques.
- Implémentation de l'algorithme AES sur FPGA.
- Réalisation d'une attaque CPA (*Correlation Power Analysis*).
- Étude et choix de métrique(s) pour l'analyse de contre-mesures.
- Développement d'une contre-mesure.
- Analyse et conclusion sur la contre-mesure développée.

Fait en trois exemplaires à Tubize, le 22 Novembre 2018

L'étudiant

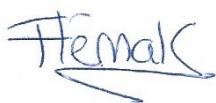
Nom – Prénom :
ANIZET Thomas

Signature :



Le tuteur

Nom – Prénom :
FLÉMAL CLÉMENCE
Département/Unité :
Électronique
Signature :



Le promoteur

Nom – Prénom :
LERMAN Liran
Société :
Thales
Signature :

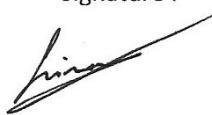


Table des matières

Abstract	I
Remerciements	III
Cahier des charges	IV
1 Introduction	1
1.1 Présentation du problème	1
1.2 Structure du travail	3
I Contexte théorique	4
2 Cryptologie	5
2.1 Concepts et définitions	5
2.1.1 Chiffrement symétrique	6
2.1.2 Chiffrement asymétrique	7
2.2 Algorithme de chiffrement	8
2.2.1 Principe	8
2.2.2 <i>Device cryptographique</i>	8
2.2.3 Algorithme <i>AES (Advanced Encryption Standard)</i>	9
3 Consommation de puissance	14
3.1 Technologie CMOS	14
3.1.1 Consommation de puissance des circuits en technologie CMOS	15
3.1.2 Composition des traces de puissance	18
3.2 Modèles de puissance	18
3.2.1 Modèle de poids de Hamming	18
3.2.2 Modèle de distance de Hamming	19
4 Attaques par canaux auxiliaires	20

4.1	Introduction	20
4.2	Analyse de la consommation de puissance	22
4.2.1	Analyse simple de la consommation	22
4.2.2	Analyse différentielle de la consommation	22
4.2.3	Analyse de la consommation par corrélation	22
5	Contre-mesures	25
5.1	Contre-mesures Hiding	25
5.2	Contre-mesures Masking	25
5.3	Contre-mesures Faking	25
II	Acquis pratiques	26
6	Configuration de l'attaque	27
6.1	Méthodologie	27
6.2	Configuration du FPGA	27
6.3	Configuration de l'oscilloscope	27
6.4	Configuration de l'attaque CPA	27
7	Mise en application d'une attaque CPA (<i>Correlation Power Analysis</i>)	28
7.1	Résultats de simulations	28
7.2	Résultats expérimentaux	28
7.3	Conclusion	28
8	Implémentation de la contre-mesure de type <i>faking</i>	29
8.1	Développement de la contre-mesure	29
8.1.1	Résultats expérimentaux	29
8.2	Conclusion	29
9	Évaluation des performances de la contre-mesure	30
9.1	Évaluation par métriques	30
9.1.1	Évaluation par critères de performances	30
10	Conclusion	31
11	Consommation de puissance	32
11.0.1	Simulations sur MATLAB	35
11.0.2	Contre-mesures	42
11.1	Conclusion	48
Crédits		50
Références		50

Table des figures

1.1	Matériel nécessaire pour réaliser une attaque par analyse de la consommation de puissance.	2
2.1	Concepts de chiffrement et de déchiffrement.	6
2.2	Chiffrement symétrique : Une seule clé est utilisée pour chiffrer et déchiffrer les messages.	7
2.3	Chiffrement asymétrique : Un clé publique est utilisée pour chiffrer le message et une clé privée est utilisée pour le déchiffrer.	7
2.4	Concept d'un algorithme de chiffrement.	8
2.5	Fonctionnement général de l'algorithme AES.	9
2.6	Les 3 matrices utilisées par l'algorithme AES.	10
2.7	Opération <i>AddRoundKey</i> entre la matrice STATE et la matrice clé.	10
2.8	Opération <i>SubBytes</i> exécutée sur la matrice STATE.	11
2.9	Opération <i>ShiftRows</i> exécutée sur la matrice STATE.	11
2.10	Opération <i>MixColumns</i> exécutée sur la matrice STATE.	12
2.11	Exemple de l'opération <i>MixColumns</i> exécutée sur la deuxième colonne (i.e colonne 1) de la matrice STATE.	12
3.1	Schéma de l'inverseur CMOS.	15
4.1	Les 2 grandes familles d'attaques physiques possibles. La suite de ce rapport se concentre essentiellement sur les attaques physiques dites passives.	21
4.2	Les différentes façons d'attaquer passivement un device cryptographique en vue de casser l'algorithme de chiffrement qu'il exploite.	21
11.1	Principe de mesure à l'oscilloscope.	32
11.2	Principe général d'une attaque CPA.	33
11.3	Schéma-bloc permettant de comprendre la simulation d'un point d'une trace.	35
11.4	Schéma-bloc permettant de comprendre la seconde phase de l'exercice 1.	35
11.5	Schéma-bloc permettant de visualiser la taille des différentes matrices employées pour la simulation.	36
11.6	Calcul du coefficient de corrélation entre un point d'une trace simulée et le poids de Hamming.	37

11.7 Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 10 traces.	37
11.8 Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 100 traces.	38
11.9 Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 1000 traces.	38
11.10Graphe présentant la valeur du coefficient de corrélation en fonction du nombre de traces. Plus le nombre de traces est élevé, plus la valeur de la clé secrète employée pour le chiffrement des données se distingue des autres clés. La vraie clé est 200.	39
11.11Schéma-bloc permettant de comprendre le principe de fonctionnement du deuxième exercice.	40
11.12Les contre-mesures de type hiding sont utilisées pour rendre aléatoire ou égale la consomma- tion de puissance du device cryptographique.	44
11.13On voit qu'au plus le nombre de traces augmente au plus une clé se distingue des 255 autres. Il s'agit de la fausse clé (en rouge). La vraie clé (en vert) ne laisse fuiter aucune information.	48

Liste des tableaux

1.1	Conséquences de l'implémentation d'une contre-mesure.	3
2.1	Les trois variantes de l'algorithme AES.	9
3.1	Table de vérité pour la fonction NON (inverseur CMOS).	14
3.2	Type de puissance consommée par une cellule CMOS en fonction des 4 transitions d'état de sa sortie.	16

Introduction

1.1 Présentation du problème

Le monde dans lequel nous vivons aujourd’hui est un monde où tout doit être connecté. Notre smartphone, notre montre, notre TV, notre frigo, notre tondeuse, ... Tout système d’électronique embarquée est potentiellement connectable ! Les transferts d’informations sont par conséquent de plus en plus gourmands. Les technologies ne cessent d’évoluer. Le meilleur exemple est le développement de l’IoT (*Internet of Things*) ou encore le développement du réseau 5G, prévu entre-autre pour augmenter les débits de données. Cependant, une question a le droit d’être posée : est-ce que toutes ces données qui voyagent de système en systèmes sont protégées ? Est-ce que les données qui transitent entre les différents appareils connectés de ma maison peuvent être captées par des pirates informatiques ? La réponse à ces questions reste bien souvent assez vague. Tentons de cerner le sujet.

Depuis toujours, l’homme cherche à dissimuler des informations à ses tiers mais depuis toujours, l’homme cherche également à accéder à des informations dont il n’a pas le droit. Cette lutte symbolise le combat entre la **cryptographie** et la **cryptanalyse**. Rappelons brièvement que la cryptographie a pour objectif de crypter de l’information, c’est-à-dire qu’elle vise à élaborer des procédés assurant la confidentialité de données sensibles. La cryptanalyse quant à elle vise à décrypter des informations sensibles auxquelles elle n’a normalement pas accès. Ainsi, lorsqu’il s’agit de manipuler des informations secrètes, la question d’intégrité et de confidentialité des informations manipulées se pose inévitablement. C’est pour cette raison que des **algorithmes de chiffrement** sont développés et par la suite, la cryptanalyse tente d’y déceler des failles. À ce jour, il existe une multitude d’algorithmes de chiffrement, chacun possédant ses avantages et ses inconvénients. Celui étudié dans ce travail est l’**algorithme AES** (*Advanced Encryption Standard*).

Initialement, la cryptanalyse se basait essentiellement sur la compréhension des procédés mathématiques utilisés dans les algorithmes en vue d’y déceler des failles. Devant ces attaques, les ingénieurs renforçèrent la complexité des algorithmes. Par ailleurs, fin du millénaire précédent, le *NIST* (*National Institute of Standards and Technology*) a encouragé le développement d’un nouveau standard d’algorithme de chiffrement difficilement envisageable à casser : l’algorithme *AES*. Sa robustesse aux attaques classiques lui permet d’être considéré aujourd’hui comme l’algorithme le plus connu au monde et implémenté dans la majeure partie des systèmes embarqués comportant des applications en sécurité. Cependant, fin des années 1990, de nouvelles recherches dans le domaine de la cryptanalyse ont été réalisées. Ces recherches ont abouties sur un nouveau type d’attaque : les **attaques par canaux auxiliaires**. Une attaque par canal auxiliaire désigne une attaque informatique qui, sans remettre en cause la robustesse théorique des méthodes et procédures de sécurité, recherche et exploite des failles dans leur implantation logicielle ou matérielle.

Il existe différents types d'attaques par canaux auxiliaires cependant ce travail se concentre sur un type d'attaque en particulier : les **attaques par analyse de la consommation de puissance**. Comme son nom l'indique, cette attaque analyse la consommation de puissance d'un *système cryptographique* en cours de chiffrement. En effet, la puissance consommée par le circuit intégré reflète directement ses activités internes (instructions exécutées et données manipulées). Cette puissance consommée peut être capturée à l'aide d'un instrument de mesure tel que l'oscilloscope. Les mesures prélevées à l'oscilloscope portent le nom de *traces de puissance*. Lorsque la clé d'un algorithme de chiffrement est révélée, le contenu censé rester confidentiel devient accessible à quiconque qui le souhaite. Ainsi, comme nous le verrons au chapitre X, dès que des traces de puissance sont acquises, il est possible pour un attaquant, via divers calculs, de retrouver la clé utilisée pour le chiffrement des données. Attirons l'attention sur le fait que l'attaque exploite les faiblesses du système cryptographique, c'est-à-dire du système implémentant l'algorithme de chiffrement (l'AES en l'occurrence). En aucun cas, l'attaque exploite les failles des principes mathématiques mis en place.

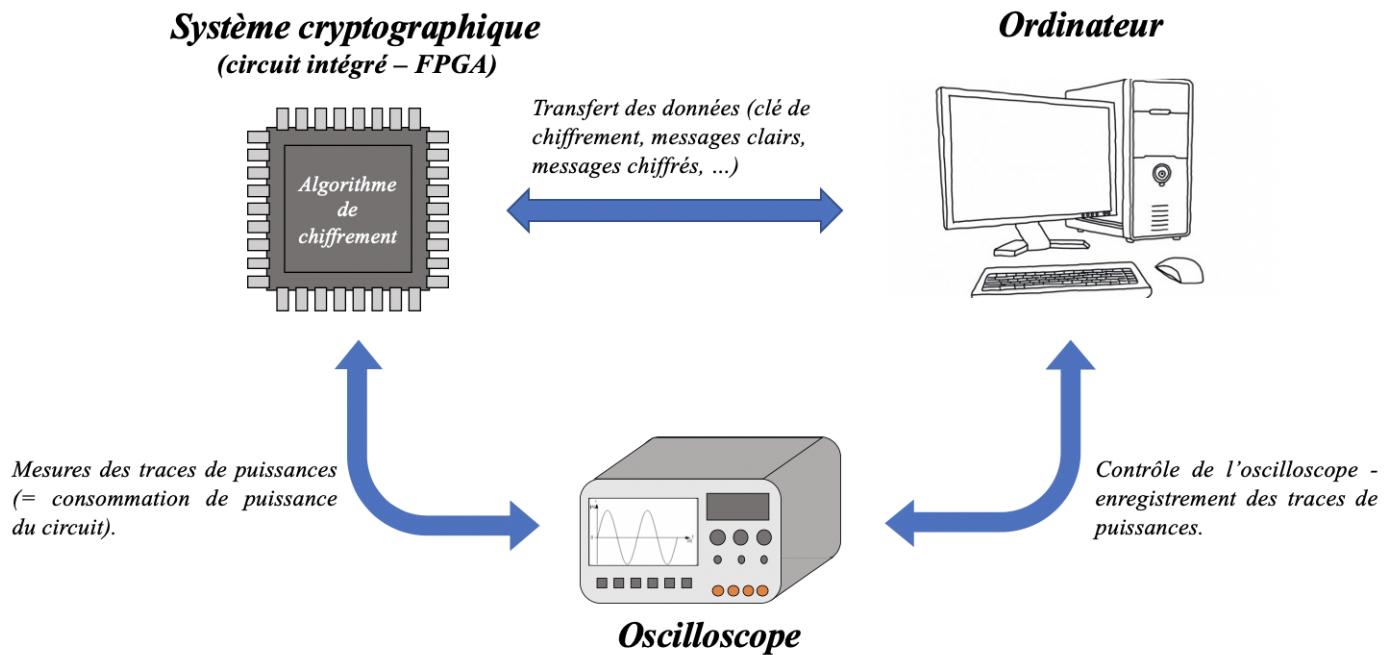


Figure 1.1 : Matériel nécessaire pour réaliser une attaque par analyse de la consommation de puissance.

Dès que ces attaques se sont manifestées, une série de contre-mesures a été développée. Pour obtenir une contre-mesure efficace contre ces attaques par analyse de la consommation de puissance, il est nécessaire de bien appréhender le fonctionnement de l'attaque. Ainsi, retenons que lors de l'exécution d'un algorithme de chiffrement, différentes opérations sont exécutées conduisant à différentes **valeurs intermédiaires calculées**. Si aucune protection n'est mise en place, la consommation de puissance du device cryptographique dépend des données intermédiaires manipulées par ce dernier (par l'algorithme précisément). Par conséquent, l'attaquant peut potentiellement retrouver la clé secrète en analysant la consommation de puissance du système. Dès lors, l'objectif général d'une contre-mesure est de rendre la consommation de puissance du device cryptographique indépendante des valeurs intermédiaires calculées par l'algorithme de chiffrement.

Deux grands types de contre-mesures sont à distinguer : les contre-mesures de type **Masking** et les contre-mesures de type **Hiding**. Brèvement, le principe d'une contre-mesure de type *Masking* est de modifier les caractéristiques de l'algorithme de chiffrement alors que le principe d'une contre-mesure de type *Hiding* est de modifier les caractéristiques physiques du système cryptographique. La contre-mesure développée dans ce travail est une contre-mesure apparue assez récemment : la contre-mesure de type *Faking*. Le principe de fonctionnement de cette contre-mesure se rapproche de celui de contre-mesure de type *Masking*. Cependant, comme nous le verrons au chapitre X, il existe quelques subtilités. Le tableau 1.1 résume les constats qui peuvent être tirés selon qu'une contre-mesure soit implantée ou non. Ces contre-mesures seront détaillées plus en profondeur au chapitre y faisant référence (chapitre X).

Avec/Sans contre-mesure	Conséquences
Sans contre-mesure	La consommation de puissance dépend directement des valeurs intermédiaires calculées par l'algorithme de chiffrement.
Avec contre-mesure type Hiding	Modification des caractéristiques de consommation : La consommation de puissance du device cryptographique est identique (ou aléatoire) pour chaque instruction exécutée et chaque donnée manipulée.
Avec contre-mesure type Masking	Modification de l'algorithme : Les valeurs intermédiaires calculées par l'algorithme de chiffrement sont masquées par des valeurs aléatoires. La consommation de puissance est alors modifiée (les traces capturées à l'oscilloscope sont insensées).
Avec contre-mesure type Faking	Modification de l'algorithme : La consommation de puissance du device cryptographique permet de trouver une valeur de clé secrète. Cependant, la clé obtenue est volontairement faussée en appliquant un masque sur la vraie clé.

Table 1.1 : Conséquences de l'implémentation d'une contre-mesure.

Enfin, pour juger de l'**efficacité de la contre-mesure implémentée**, différentes analyses peuvent être réalisées. Ainsi, des métriques telles que le *SNR* (*Signal-Noise to Ratio*) peuvent être utilisées afin de savoir si les traces capturées à l'oscilloscope laissent fuiter de l'information ou non. Outre les métriques, une analyse des performances du système sera également établie. Les principales performances étudiées sont : le temps d'exécution de l'algorithme, la consommation de puissance du device cryptographique, la place mémoire de l'implémentation protégée de l'algorithme sur le device cryptographique, ...

1.2 Structure du travail

Ce travail est scindé en deux parties :

1. La première partie aborde tous les concepts théoriques nécessaires afin de comprendre le fondement des attaques par canaux auxiliaires. Un premier chapitre expose les concepts et définitions de la cryptologie et définit en particulier le fonctionnement de l'algorithme AES. Le second chapitre est consacré à la consommation de puissance des circuit en technologie CMOS. Cet analyse de la consommation de puissance pour ce type de circuit en particulier permet de comprendre le principe de fonctionnement des attaques par analyse de la consommation de puissance. Le troisième chapitre définit le sujet principal de ce travail, à savoir les attaques par canaux auxiliaires. Enfin, le dernier chapitre étudie les différentes contre-mesures existantes afin d'assurer une meilleure protection des données sensibles.
2. La seconde partie aborde les notions et compétences acquises durant la période de TFE. Un premier chapitre définit la configuration du matériel nécessaire pour implémenter l'attaque par analyse de la consommation de puissance. Un descriptif succinct des langages et matériels utilisés est ainsi détaillé. Le second chapitre reprend les résultats obtenus et leurs conclusions pour la mise en application d'une attaque CPA (*Correlation Power Analysis*). Le troisième chapitre détaille l'implémentation de la contre-mesure de type *faking*. Le quatrième chapitre discute de l'efficacité de la contre-mesure implémentée. Enfin, un cinquième chapitre est consacré à la conclusion de ce rapport.

Première partie

Contexte théorique

Chapitre 2

Cryptologie

Ce chapitre introduit au lecteur les fondamentaux en matière de cryptologie. Les différents concepts et définitions abordés ont pour but d'apporter au lecteur les connaissances nécessaires et suffisantes pour la bonne compréhension des chapitres suivants. Ce chapitre est scindé en deux parties. Dans un premier temps, il abordera les définitions importantes en matière de cryptologie et plus spécifiquement en matière de cryptographie. La cryptographie symétrique sera ainsi distinguée de la cryptographie asymétrique. Dans un second temps, ce chapitre détaillera le principe de fonctionnement général d'un algorithme de chiffrement. Une attention particulière sera finalement consacrée à la compréhension de l'algorithme AES.

2.1 Concepts et définitions

Il y a plus de 2000 ans naissait le concept de **cryptologie** avec la montée au pouvoir de Jules César. Général romain très puissant, César marqua le monde et l'histoire universelle au travers de ses nombreuses conquêtes. Tant adulé par son peuple, César s'était néanmoins fait plus d'un ennemis durant ses conquêtes. Le 15 mars 44 avant J-C, il fut tragiquement assassiné par une conspiration de sénateurs romains dont son fils, Marcus Junius Brutus. Jules César avait pris pour habitude de se méfier de ses ennemis. Arrivé à un stade de méfiance extrême, il n'avait plus confiance en ses propres messagers. C'est pour cette raison qu'il envoyait des messages chiffrés où il décalait chaque lettre d'un mot de 3 lettres dans l'alphabet. Seul le destinataire, au courant de la règle de chiffrement, pouvait alors déchiffrer les messages. Ce fut le premier procédé de chiffrement de l'histoire, appelé *Chiffre de César* (voir annexe A.1).
C'est ainsi que tout commença...

Étymologiquement, le mot "*cryptologie*" vient du grec et signifie littéralement la *science du secret* (*kryptos* pour caché et *logos* pour science). La cryptologie regroupe deux champs d'études diamétralement opposés : la cryptographie et la cryptanalyse.

Définition 2.1.1. Cryptographie : *La cryptographie est la science qui utilise les mathématiques en vue de chiffrer et déchiffrer de l'information secrète.*

Définition 2.1.2. Cryptanalyse : *La cryptanalyse est la science qui analyse les procédés mathématiques mis en place par la cryptographie en vue de retrouver de l'information secrète.*

Six termes couramment employés en cryptologie doivent également être définis :

Texte clair : Il s'agit de données brutes, considérées comme confidentielles, compréhensibles sans intervention spécifique.

Texte chiffré : Il s'agit de données modifiées, considérées comme confidentielles et devant le rester. Ces données sont incompréhensibles, inintelligibles pour quiconque qui tenterait de les comprendre.

Chiffrement (cryptage) : Procédé utilisé pour dissimuler le texte clair. Le chiffrement transforme donc le texte clair en texte chiffré.

Déchiffrement (décryptage) : Procédé inverse au chiffrement. Le déchiffrement transforme donc le texte chiffré en texte clair, c'est-à-dire en texte d'origine.

Algorithme de chiffrement : Fonctions mathématiques utilisées lors du processus de chiffrement. Les fonctions mathématiques inverses sont également définies pour le processus de déchiffrement.

Clé de chiffrement : Paramètre associé à l'algorithme de chiffrement pour le cryptage des textes clairs ou pour le décryptage des textes chiffrés.

La figure 2.1 illustre cinq des six termes fondamentaux énoncés. Le sixième, l'algorithme de chiffrement, pourrait être confondu avec avec les étapes de *chiffrement* et *déchiffrement* cependant, une explication plus détaillée est donnée à la section 2.2.

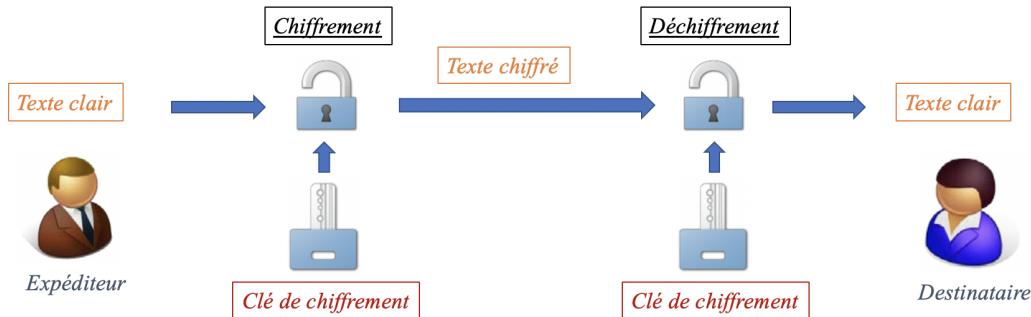


Figure 2.1 : Concepts de chiffrement et de déchiffrement.

Revenons sur la notion de cryptographie. La cryptographie doit remplir trois conditions majeures qui sont :

- **Confidentialité** : La confidentialité spécifie que seules les personnes autorisées à accéder à une certaine information ont la possibilité de l'atteindre.
- **Intégrité** : L'intégrité spécifie que l'information, lors de son traitement ou de sa transmission, ne subit aucun modification volontaire ou accidentelle. Autrement dit, il s'agit d'une garantie que l'information conserve son état d'origine.
- **Disponibilité** : La disponibilité spécifie que l'information est en tout temps disponible pour une personne autorisée à y accéder et en demandant l'accès.

Ainsi, la cryptographie a pour objectif de chiffrer des informations tout en assurant la confidentialité, l'intégrité et la disponibilité de ces informations. La cryptographie se divise en deux catégories distinctes : la cryptographie symétrique et la cryptographie asymétrique.

2.1.1 Chiffrement symétrique

Définition 2.1.3. *Chiffrement symétrique* : Le chiffrement est dit symétrique lorsque la clé utilisée pour le chiffrement et le déchiffrement des données est la même. La clé est alors appelée clé secrète.

Par convention, ce type de chiffrement permet à la fois de chiffrer et de déchiffrer des messages à partir d'une seule et unique clé : la clé secrète. Le désavantage de ce type de chiffrement est que si une personne parvient à subtiliser la clé, elle sera en mesure de déchiffrer tout message qu'elle intercepte.

Exemple : L'algorithme AES. Une explication plus détaillée de cet algorithme est reprise à la section 2.2.3.

La figure 2.2 (page suivante) présente le principe de fonctionnement du chiffrement symétrique. Comme le présente la figure, si un attaquant intercepte un message, ce dernier sera chiffré. Il doit donc être en possession de la clé secrète s'il souhaite comprendre le message envoyé par l'expéditeur.

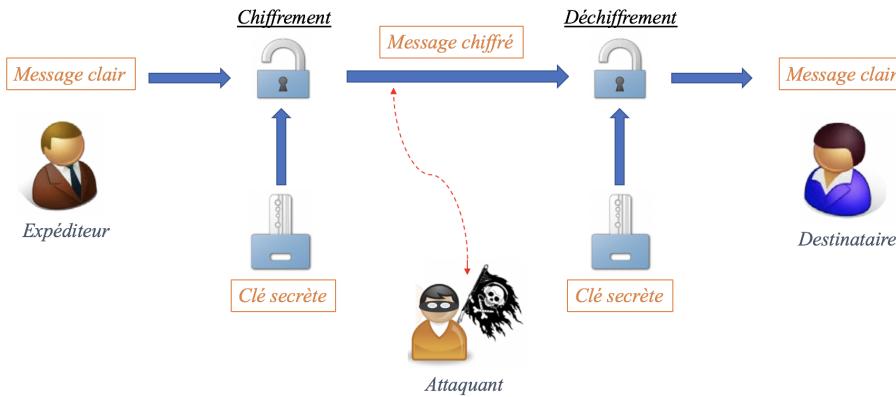


Figure 2.2 : Chiffrement symétrique : Une seule clé est utilisée pour chiffrer et déchiffrer les messages.

2.1.2 Chiffrement asymétrique

Définition 2.1.4. Chiffrement asymétrique : Le chiffrement est dit asymétrique lorsqu'il existe une clé publique utilisée pour le chiffrement des textes clairs et une clé privée utilisée pour le déchiffrement des textes chiffrés.

Par convention, la clé publique est la clé de chiffrement du message clair, elle peut être communiquée sans aucune restriction tandis que la clé privée est la clé de déchiffrement du message chiffré, elle ne doit être communiquée sous aucun prétexte. Le fonctionnement est le suivant : Avec une clé publique, l'expéditeur envoie un message chiffré selon un algorithme de chiffrement préalablement défini. Ce message, une fois transmis, ne pourra être déchiffré que par le destinataire, détenteur de la clé privée.

Exemple : L'algorithme RSA (Rivest Shamir Adleman).

La figure 2.3 ci-dessous présente le principe de fonctionnement du chiffrement asymétrique.

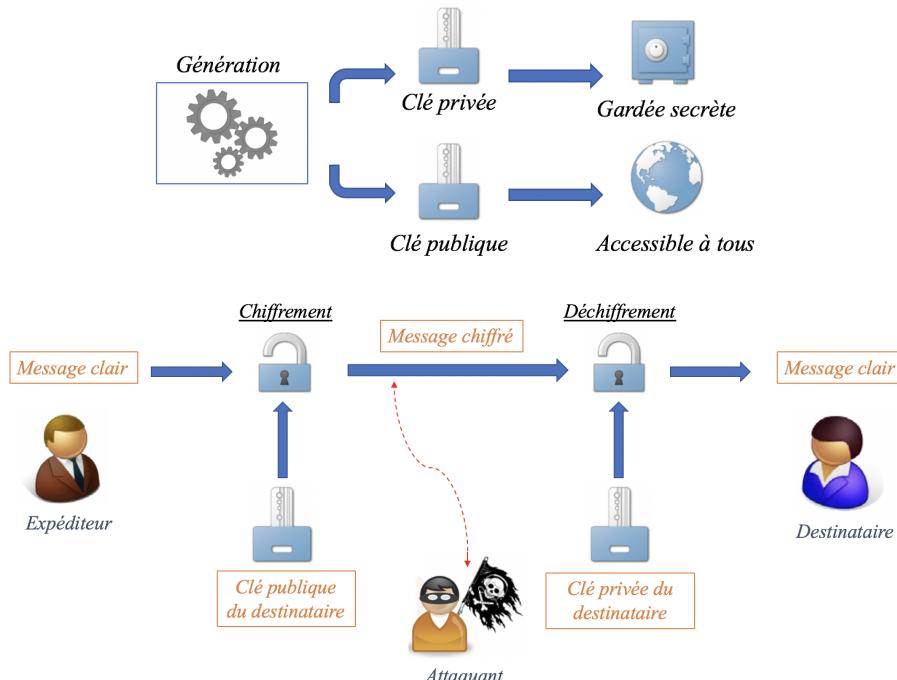


Figure 2.3 : Chiffrement asymétrique : Un clé publique est utilisée pour chiffrer le message et une clé privée est utilisée pour le déchiffrer.

2.2 Algorithme de chiffrement

2.2.1 Principe

Définition 2.2.1. *Algorithme de chiffrement* : Un algorithme de chiffrement est un ensemble de fonctions mathématiques utilisé lors du processus de chiffrement et de déchiffrement de données sensibles. Une clé publique ou privée est associée à l'algorithme de chiffrement selon qu'il s'agisse d'un chiffrement symétrique ou asymétrique.

Les systèmes de sécurité modernes utilisent des algorithmes de chiffrement pour assurer la confidentialité et l'intégrité de données sensibles. Ces algorithmes prennent typiquement :

- 2 paramètres en entrée : un *message clair* et une *clé de chiffrement*.
- 1 paramètre en sortie : le *message chiffré*.

Comme décrit précédemment, le procédé transformant les données claires en entrée en données chiffrées en sortie est appelé chiffrement. Ce procédé est réalisé grâce à un algorithme de chiffrement manipulant une clé de chiffrement, un message clair et diverses opérations mathématiques. Avec des clés différentes, le résultat du cryptage variera également. La figure 2.4 ci-dessous présente le principe de fonctionnement d'un algorithme de chiffrement.

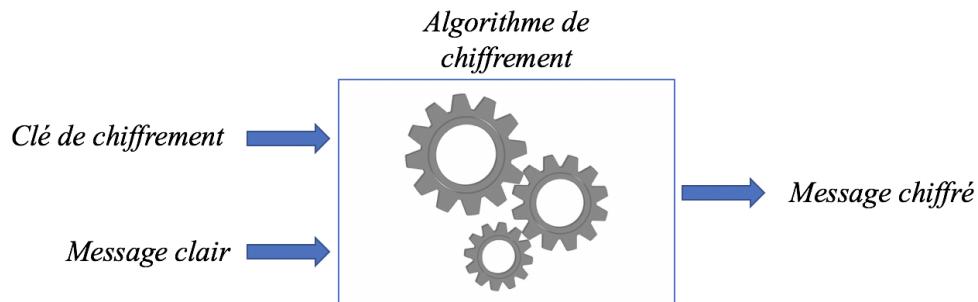


Figure 2.4 : Concept d'un algorithme de chiffrement.

Il est important de préciser que tous les détails décrivant le fonctionnement d'un algorithme cryptographique sont disponibles publiquement, seule la clé de chiffrement doit rester secrète. En effet, un principe fondamental de la cryptographie est le principe de Kerckhoffs. Ce principe stipule que la sécurité de tout algorithme de chiffrement ne doit pas reposer sur la connaissance du fonctionnement de ce dernier. Autrement dit, la sécurité offerte par un algorithme de chiffrement ne doit pas dépendre du secret de son implémentation mais doit uniquement reposer sur la protection de la clé, ou plus généralement d'un secret. Par conséquent, un bon algorithme est un algorithme dont on ne parviendra pas à déchiffrer les données chiffrées, même en connaissant son fonctionnement interne. Lorsque la clé d'un algorithme est trouvée, le déchiffrement des données confidentielles devient possible. On dit que l'algorithme de chiffrement est *cassé*.

2.2.2 Device cryptographique

Que ce soit pour un algorithme de chiffrement symétrique ou asymétrique, la clé de chiffrement tout comme l'algorithme en lui-même doivent être stockés sur un support physique. Ce support, appelé *device cryptographique*, doit être suffisamment sécurisé que pour contenir la clé de manière protégée. Par la suite, les messages clairs seront envoyés au device cryptographique qui se chargera de les chiffrer selon l'algorithme implémenté et la clé utilisée. Un exemple de device cryptographique utilisé est le *FPGA* (*Field-Programmable Gate Array*).

Définition 2.2.2. *Device cryptographique* : Un device cryptographique est un device qui implémente des algorithmes de chiffrement et qui stocke des clés de chiffrement.

2.2.3 Algorithmes AES (*Advanced Encryption Standard*)

En 1997, le NIST (*National Institute of Standards and Technology*) décida qu'il était temps de développer un nouveau standard d'algorithme de chiffrement. Ce nouveau standard, nommé **AES** (pour *Advanced Encryption Standard*), était appelé à remplacer l'ancien standard de chiffrement, l'algorithme DES (pour *Data Encryption Standard*). Pour ce faire, le NIST organisa un concours cryptographique. Les chercheurs du monde entier furent invités à soumettre leurs propositions. En Octobre 2000, Le NIST annonça le vainqueur du concours : l'algorithme de Rijndael, du nom de ses concepteurs Joan Daemen et Vincent Rijmen, tous deux de nationalité belge.

L'algorithme de Rijndael, désormais plus connu sous le nom d'algorithme AES, est un **algorithme de chiffrement symétrique par blocs**. Par *blocs* signifie que les données sont traitées par blocs de 128 bits. La clé secrète peut posséder différentes tailles : 128 bits (AES-128), 192 bits (AES-192) ou encore 256 bits (AES-256). À noter qu'en théorie, plus la taille de la clé est élevée, moins il y a de chance de casser l'algorithme. Cela est dû au fait que, plus la taille de la clé est grande, plus il existe un nombre important de possibilités de clé à tester avant de retrouver celle qui correspond à la valeur exacte. L'algorithme AES n'a ainsi, à ce jour, jamais été cassé par des méthodes d'attaques classiques. Cependant, comme nous le verrons par la suite avec les attaques par canal auxiliaire (section 4.1), le problème peut vite être contourné. Pour ce travail, il m'a été demandé d'attaquer une implémentation de l'algorithme AES-256. La description qui suit est dans un premier temps basée sur l'algorithme AES-128. Dans un second temps, une adaptation sera insérée afin de comprendre le fonctionnement de l'algorithme AES-256.

Avant de détailler le fonctionnement de l'algorithme AES-128, précisons quelques caractéristiques principales. L'algorithme AES est caractérisé par une série de tours (*rounds* en anglais) dépendant de la taille de la clé. Pour une clé dont la taille est 128 bits, on dénombre 10 tours. Pour une clé de taille 192 bits, on dénombre 12 tours et pour une clé de taille 256 bits, on dénombre 14 tours. Un *round* est défini par 4 opérations appliquées succinctement sur une matrice de données. Ces 4 opérations sont : *AddRoundKey*, *SubBytes*, *ShiftRows* et *MixColumns*. Elles sont appliquées à divers instants dans l'exécution de l'algorithme AES. Le tableau 2.1 présente un récapitulatif des principales caractéristiques des trois variantes de l'algorithme AES. La figure 2.5 permet de visualiser l'ordre d'exécution chronologique des 4 opérations opérées par l'algorithme. Les annexes A.2 et A.3 respectivement, reprennent le code réalisé pour l'exécution de l'algorithme AES-128 et l'algorithme AES-256.

	Taille de la clé (bits)	Taille du bloc de données (bits)	Nombre de <i>rounds</i>
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

Table 2.1 : Les trois variantes de l'algorithme AES.

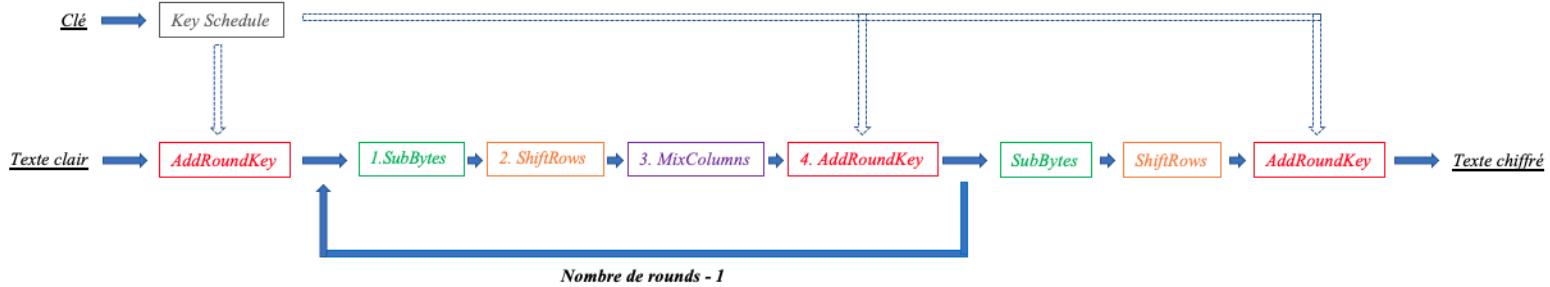


Figure 2.5 : Fonctionnement général de l'algorithme AES.

L’AES-128 a donc pour rôle de chiffrer des blocs de données de 128 bits avec une clé de 128 bits. Les données et la clé sont représentées par une matrice où chaque élément de la matrice correspond à un byte (un octet, i.e 8 bits). Étant donné que 128 bits correspondent à 16 bytes, la matrice de données, au même titre que la matrice de clé, correspond à une matrice de 4 lignes et 4 colonnes (formant ainsi les 4x4 soit 16 bytes). Une matrice particulière (de taille 4x4 également) appelée STATE contient l’ensemble des résultats intermédiaires résultant des diverses opérations que subissent les données (depuis leur état initial). La figure 2.6 présente les 3 matrices qui viennent d’être citées : la matrice de donnée (message clair initial de 128 bits), la matrice STATE (qui va contenir les résultats intermédiaires des données suite aux différentes opérations) et la matrice clé (clé de 128 bits).

d_0	d_4	d_8	d_{12}
d_1	d_5	d_9	d_{13}
d_2	d_6	d_{10}	d_{14}
d_3	d_7	d_{11}	d_{15}

Matrice de données

S_0	S_4	S_8	S_{12}
S_1	S_5	S_9	S_{13}
S_2	S_6	S_{10}	S_{14}
S_3	S_7	S_{11}	S_{15}

Matrice STATE

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

Matrice clé

Figure 2.6 : Les 3 matrices utilisées par l’algorithme AES.

Remarque : En pratique, la matrice de données est directement confondue avec la matrice STATE. Autrement dit, les premiers éléments à être placés dans la matrice STATE représentent les bytes de données. Ainsi, on n’utilise que deux matrices durant le fonctionnement de l’algorithme AES : la matrice STATE et la matrice clé.

Fonctionnement AES-128 :

Initialement, deux matrices vont être utilisées : la matrice STATE, contenant les données claires, et la matrice clé, contenant la clé secrète initiale. Comme on peut le voir sur la figure 2.5, la première opération à être appliquée sur ces deux matrices est l’opération *AddRoundKey*. Cette opération réalise un XOR (symbole \oplus) entre chaque élément de la matrice STATE et chaque élément respectif de la matrice clé. Le résultat est ré-écrit dans la matrice STATE. La figure 2.7 ci-dessous présente le principe de fonctionnement de cette première opération :

12	e0	13	28
04	ab	f8	d3
23	19	69	26
e5	b9	7a	4c

\oplus

ab	88	23	2a
10	fe	a3	6c
fe	2c	39	75
17	b1	39	05

=

b9	68	30	02
14	55	5b	bf
dd	35	50	53
f2	08	43	49

Matrice STATE

Matrice clé

Nouvelle matrice STATE

Figure 2.7 : Opération *AddRoundKey* entre la matrice STATE et la matrice clé.

Ensuite, toujours selon la figure 2.5, une série de quatre opérations se répétant neuf fois (car AES-128) est exécutée. Ces quatres opérations sont appliquées dans l’ordre chronologique suivant : *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*.

Enfin, une fois les neufs rounds exécutés, trois opérations terminent le chiffrement : *SubBytes*, *ShiftRows* et *AddRoundKey*. À la fin de ces trois dernières opérations, une matrice de taille 4x4 (la matrice STATE) présente le message chiffré de 128 bits.

Description des 4 opérations :

1. SubBytes : La figure 2.8 ci-dessous présente le principe de fonctionnement de l'opération *SubBytes*. Le principe de fonctionnement présenté dans ce cas-ci est simple : il repose sur une table de substitution, appelée Sbox et présentée en annexe A.4. La matrice STATE avant l'exécution de l'opération contient 16 bytes. Chacun de ses 16 bytes (notés $S_{i,j}$) va fournir une nouvelle valeur de byte (noté $S'_{i,j}$) en fonction de la table de substitution. Un exemple est donné afin de comprendre le principe : Si le byte $S_{1,2}$ vaut 53 (en hexadécimal) alors, selon la table Sbox, la valeur du byte résultant $S'_{1,2}$ vaut ed (en hexadécimal).

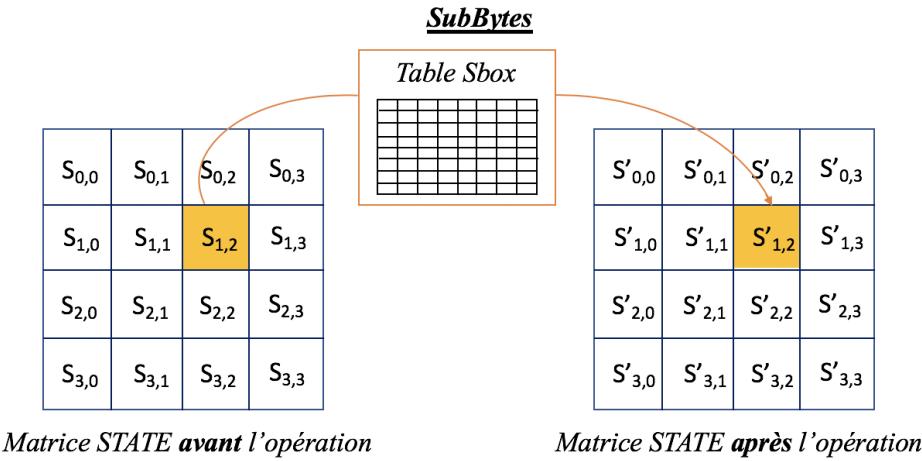


Figure 2.8 : Opération *SubBytes* exécutée sur la matrice STATE.

2. ShiftRows : Comme son nom l'indique, cette opération concerne les lignes de la matrice STATE. Cette opération réalise une permutation cyclique des octets sur les lignes de la matrice STATE. Plus précisément, **pour la i -ième ligne, on décalera chaque élément de la matrice STATE de i positions vers la gauche**, en considérant que la première ligne a pour indice 0. La figure 2.9 ci-dessous présente le principe de fonctionnement de l'opération *ShiftRows* :

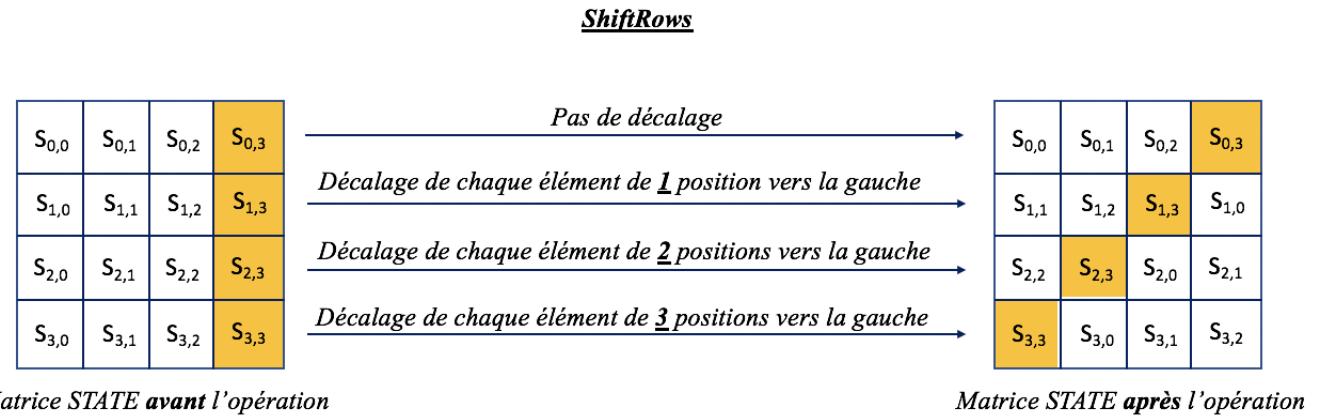


Figure 2.9 : Opération *ShiftRows* exécutée sur la matrice STATE.

3. MixColumns : Comme son nom l'indique, cette opération concerne les colonnes de la matrice STATE. Cette opération réalise un produit matriciel entre une matrice fixée (taille 4x4) définie ci-dessous (figure 2.11) et un vecteur colonne (taille 4x1) de la matrice STATE. Il en résulte un nouveau vecteur colonne (taille 4x1) permettant de définir la nouvelle matrice STATE. La figure 2.10 ci-dessous présente le principe de fonctionnement de l'opération *MixColumns*. Un exemple est ensuite donné à la figure 2.11.

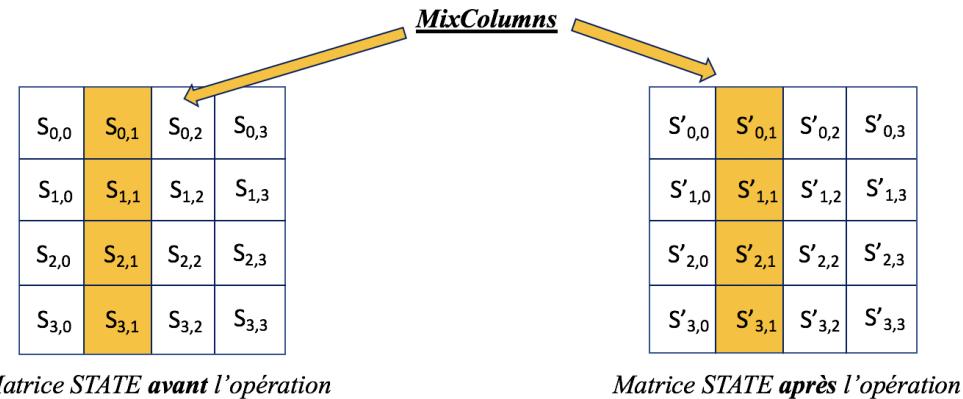


Figure 2.10 : Opération *MixColumns* exécutée sur la matrice STATE.

$$\begin{array}{c}
 \left[\begin{array}{cccc} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{array} \right] \quad \left[\begin{array}{c} S_{0,1} \\ S_{1,1} \\ S_{2,1} \\ S_{3,1} \end{array} \right] = \left[\begin{array}{c} S'_{0,1} \\ S'_{1,1} \\ S'_{2,1} \\ S'_{3,1} \end{array} \right]
 \end{array}$$

Matrice fixée Colonne 1 de la matrice STATE avant opération Colonne 1 de la matrice STATE après opération

Exemple : 1^{er} élément du vecteur colonne

$$S'_{0,1} = (02 * S_{0,1}) + (03 * S_{1,1}) + (01 * S_{2,1}) + (01 * S_{3,1})$$

Figure 2.11 : Exemple de l'opération *MixColumns* exécutée sur la deuxième colonne (i.e colonne 1) de la matrice STATE.

4. AddRoundKey : Comme précisé précédemment dans le fonctionnement initial, l'opération *AddRoundKey* réalise un XOR entre la matrice STATE et la matrice de clé. Le résultat de ce XOR est placé dans la nouvelle matrice STATE (voir figure 2.7).

L'opération *KeySchedule* :

Il persiste une opération qui n'a pas encore été expliquée : l'opération *KeySchedule*. Cette opération s'exécute uniquement sur la matrice clé. Par conséquent, la gestion des clés se fait au travers des deux fonctions qui sont les fonctions *AddRoundKey* et *KeySchedule*. Nous connaissons le principe de fonctionnement de l'opération *AddRoundKey*, reste à comprendre l'utilité de l'opération *KeySchedule*.

Jusqu'à présent, nous avons vu que la matrice STATE était modifiée après chaque opération exécutée. Ce qu'on ne sait pas encore, c'est que la matrice clé est également modifiée à chaque round afin d'éviter d'utiliser toujours la même valeur de clé. Cela permet ainsi de complexifier le chiffrement des données. On l'aura deviné, l'opération qui modifie la matrice clé initiale en une nouvelle matrice clé est appelée ***KeySchedule***. Cette opération génère une nouvelle clé sur base de la clé précédemment employée. Ainsi, pour générer la première nouvelle clé, l'opération *KeySchedule* emploiera la clé initialement donnée. Étant donné que pour l'AES-128, nous avons un total de dix rounds à exécuter, cela signifie que l'opération *KeySchedule* s'exécute dix fois afin de générer dix nouvelles clés de 128 bits. En rajoutant la clé initiale, nous avons alors onze clés de chiffrement différentes permettant d'exécuter les onze opérations *AddRoundKey* présentes dans l'algorithme AES-128. Ainsi, comme présenté à l'annexe A.2, l'opération *KeySchedule* s'exécute en premier lieu dans le code afin de générer dix nouvelles clés. En comptant la clé initialement donnée, il y a donc onze clés. Chacune de ces onze clés sera utilisée lors de l'appel de la fonction *AddRoundKey*.

L'annexe A.5 permet de comprendre le principe de fonctionnement de l'opération *KeySchedule*, utilisée afin de générer les nouvelles clés. Plus précisément, cette annexe présente un exemple pour générer la première nouvelle clé (taille de 128 bits). Le fonctionnement est le suivant : l'opération *KeySchedule* s'exécute colonne par colonne sur la matrice clé de taille 4x4. On va d'abord générer la première colonne de la nouvelle clé, ensuite on générera les colonnes 2, 3 et 4. C'est la première colonne qui est la plus compliquée à générer. Les 3 autres colonnes réalisent simplement un XOR pour générer la nouvelle colonne. Ainsi, pour obtenir la 1ère colonne de la nouvelle clé (soit les quatre premiers bytes), on va prendre trois vecteurs colonnes (taille 4x1), à savoir :

1. Prendre la 4ème colonne de la clé initiale.
 - (a) Décaler chaque élément de cette colonne d'une position vers le haut.
 - (b) Appliquer l'opération *SubBytes* sur chaque élément de la colonne.
2. Prendre la 1ère colonne de la clé initiale.
3. Prendre la 1ère colonne de la matrice RCON.

Une fois ces trois vecteurs colonnes obtenus, on réalise un XOR entre eux. Le résultat de ce XOR représente la 1ère colonne de la nouvelle matrice clé. Pour les colonnes 2, 3 et 4, le principe est le suivant : la colonne i de la nouvelle clé est obtenue en réalisant un XOR entre la colonne i de l'ancienne clé et la colonne $i-1$ de la nouvelle clé (avec $i \in \{2; 4\}$). L'annexe A.5 donne un exemple concret et didactique de l'opération *KeySchedule*.

Adaptation pour l'AES-256 :

En se basant sur le tableau 2.1 et la figure 2.5, nous constatons que les deux différences majeures entre l'algorithme AES-128 et l'algorithme AES-256 sont :

- La taille de la clé : on passe de 128 bits à 256 bits.
- Le nombre de rounds : on passe de 10 rounds à 14 rounds.

Ainsi, l'opération *KeySchedule* s'exécute 14 fois. Cependant, cette opération ne génère pas 14 nouvelles clés de 256 bits mais seulement 7 nouvelles clés de 256 bits. La raison est la suivante : les 256 bits de la clé sont scindés en deux parties égales, c'est-à-dire en 128 bits, afin que les 4 autres opérations décrites précédemment (*SubBytes*, *ShiftRows*, *MixColumns* et *AddRoundKey*) puissent s'exécuter sans problème. En effet, la matrice STATE ayant toujours une taille de 128 bits, la matrice clé doit obligatoirement avoir une taille équivalente (128 bits) afin que les quatre opérations puissent être exécutées comme décrit pour l'algorithme AES-128. Le premier round utilisera donc les seize premiers bytes de la clé initiale ; La second round utilisera les seize derniers bytes de la clé initiale (les trente-deux bytes de la clé initiale ont ainsi été utilisés) ; Le troisième round utilisera les seize premiers bytes de la seconde clé, générée par l'opération *KeySchedule* ; et ainsi de suite jusqu'au quatorzième et dernier round.

Chapitre 3

Consommation de puissance

Ce chapitre introduit au lecteur une technologie bien répandue dans les systèmes informatiques modernes : la technologie CMOS. C'est en étudiant le fonctionnement de cette technologie que les attaques par analyse de la consommation de puissance sont rendues praticables. Le principe de fonctionnement de cette technologie constitue donc une étape nécessaire dans l'apprentissage du lecteur. En effet, ce dernier sera ainsi en mesure de comprendre pourquoi une analyse de la consommation de puissance d'un circuit en technologie CMOS permettra plus tard (chapitre 4) de retrouver la clé d'un algorithme de chiffrement. De cette façon, ce chapitre se scinde en deux parties. Dans un premier temps, l'objectif est de comprendre le fonctionnement de la technologie CMOS ainsi que ses répercussions sur les différentes puissances intervenant. Dans un second temps, l'objectif est d'assimiler le concept de *modèle de puissance*. Pour ce faire, deux modèles seront judicieusement développés.

3.1 Technologie CMOS

La **technologie CMOS** (pour *Complementary Metal Oxyde Semiconductor*) est la technologie la plus répandue parmi toutes les technologies de semi-conducteurs. En effet, on la retrouve dans la majorité des systèmes informatiques modernes. En 2001, elle couvrait 86% de la production mondiale des circuits intégrés. Pour cette raison, nous nous intéressons à leur conception afin de détecter des anomalies qui pourraient se révéler être utiles pour la cryptanalyse.

Le nom de cette technologie vient du fait que toutes les fonctions logiques (portes OR, AND, etc.) peuvent être réalisées moyennant l'utilisation d'une paire de transistors MOS complémentaires (N-MOS et P-MOS) associés symétriquement et fonctionnant en régime de commutation. Cela signifie que lorsqu'un des deux transistors MOS conduit, l'autre est par conséquent fermé. Grâce à ce principe, une porte logique CMOS ne consomme de l'énergie qu'au moment de la commutation. Cette caractéristique permet de distinguer le CMOS de toutes les autres technologies.

Pour expliquer le fonctionnement de cette technologie, on peut prendre un exemple simple : **l'inverseur CMOS**. Un inverseur CMOS est une fonction "NON" dont voici la table de vérité (table 3.1) :

<i>Entrée</i>	<i>Sortie</i>
0	1
1	0

Table 3.1 : Table de vérité pour la fonction NON (inverseur CMOS).

La figure 3.1 ci-dessous (schéma a) présente le schéma de l'inverseur CMOS :

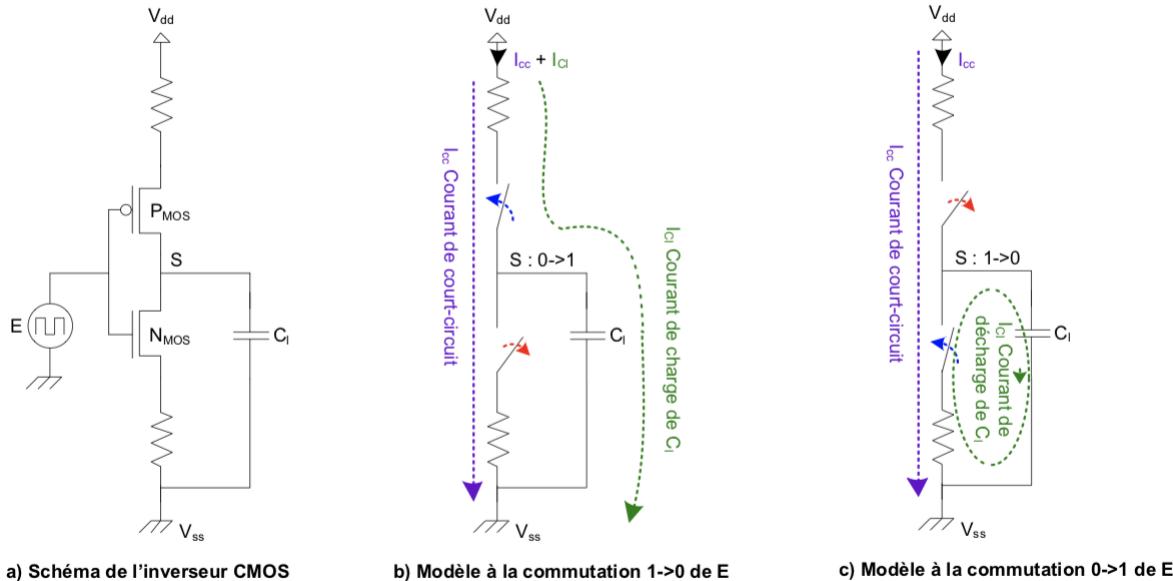


Figure 3.1 : Schéma de l'inverseur CMOS.

Si on applique à l'entrée (E) un état bas, le transistor N est bloqué et le P est passant (schéma b). On place ainsi la sortie au potentiel Vdd (la tension d'alimentation), c'est-à-dire à l'état haut. Inversement, quand on met l'entrée à l'état haut, le transistor P est bloqué et le N est passant (schéma c). La sortie est donc à l'état bas. On a donc bien réalisé une fonction inversion. Dans la suite de ce travail, on considérera toujours que les circuits attaqués sont réalisés en technologie CMOS.

3.1.1 Consommation de puissance des circuits en technologie CMOS

Il est évident que les circuits digitaux modernes consomment de la puissance lorsqu'ils exécutent des instructions. Dans notre cas, c'est un FPGA sur lequel est implanté l'algorithme AES qui consomme de la puissance. Dans le domaine de la cryptanalyse, cette puissance va être mesurée et analysée afin de déterminer si le device cryptographique laisse fuiter des informations. Si tel est le cas, nous verrons comment ces informations peuvent être utilisées afin de retrouver le clé de chiffrement de l'algorithme AES (chapitre 4). Pour ce faire, nous devons tout d'abord étudier les différentes puissances intervenant dans les circuits réalisés en technologie CMOS. Pour rappel, cette technologie est très répandue et couvre la plupart des circuits digitaux modernes.

La consommation totale de puissance d'un circuit CMOS peut être obtenue en sommant les consommations de puissance respectives de chaque cellule logique (cellule conçue pour remplir une certaine fonction logique) du circuit CMOS. De cette façon, la consommation de puissance totale dépend essentiellement du nombre de cellules logiques dans le circuit CMOS.

En prenant comme exemple de cellule logique CMOS, l'inverseur CMOS (expliqué à la section 3.1), nous allons tenter de comprendre quand et pour quelles raisons ces cellules CMOS dissipent de la puissance. Pour ce faire, il faut savoir que la consommation de puissance est essentiellement divisée en deux parties :

- La puissance statique (notée P_{stat}) : C'est la puissance qui est consommée lorsqu'il n'y a pas de commutation dans une cellule (c'est-à-dire dans l'inverseur). Autrement dit, c'est la puissance qui est consommée lorsque l'inverseur est en fonctionnement normal mais ne commute pas.
- La puissance dynamique (notée P_{dyn}) : C'est la puissance qui est consommée par une cellule si la sortie de celle-ci commute.

Ainsi, la puissance totale consommée par une cellule vaut la somme de ces deux composantes, soit :

$$P_{total} = P_{stat} + P_{dyn} \quad (3.1)$$

Mais de façon plus précise, que vaut la puissance statique ? De même, comment peut-on exprimer la puissance dynamique ?

Puissance statique :

Les cellules CMOS sont toujours construites de façon à ce que les deux transistors complémentaires ne soient jamais passants au même moment. En effet, si on reprend l'exemple de l'inverseur CMOS (3.1), lorsqu'on met le signal d'entrée à GND alors le transistor P1 est passant et N1 est bloqué. Par contre, lorsqu'on met le signal d'entrée à V_{DD} , le transistor P1 devient bloqué tandis que le transistor N1 devient passant. Ainsi, en théorie, un seul transistor fonctionne et laisse passer du courant. Cependant, en pratique, lorsqu'un transistor MOS est bloqué, le courant qui le traverse n'est jamais entièrement nul. En effet, une très petite valeur de courant circule dans le canal du transistor. Ce courant, que l'on appelle *courant de fuite* et que l'on note I_{fuite} , produit une consommation de puissance **statique** pouvant être calculée de la façon suivante :

$$P_{stat} = I_{fuite} \cdot V_{DD} \quad (3.2)$$

Ainsi, on peut conclure que la consommation de puissance statique des circuits CMOS correspond à la puissance qui est consommée par le circuit lorsqu'il n'y a pas de commutation dans une cellule. Cette puissance est typiquement très faible et sera, en pratique, négligée.

Puissance dynamique :

La consommation de puissance dynamique apparaît typiquement lors d'une commutation des transistors. Une commutation est le passage d'un état haut à un état bas ou d'un état bas à un état haut. En réalité, il existe 4 transitions d'état possibles. Ces 4 possibilités sont reprises dans le tableau 3.2 ci-dessous.

<i>Transitions</i>	<i>Type de puissance consommée</i>
$0 \rightarrow 0$	Statique
$0 \rightarrow 1$	Statique + Dynamique
$1 \rightarrow 0$	Statique + Dynamique
$1 \rightarrow 1$	Statique

Table 3.2 : Type de puissance consommée par une cellule CMOS en fonction des 4 transitions d'état de sa sortie.

On constate que pour chaque transition possible, il y a présence de puissance statique. Cependant, il n'y a présence de puissance dynamique que dans le cas d'une commutation, c'est-à-dire lors des deux transitions : $0 \rightarrow 1$ et $1 \rightarrow 0$. En toute logique, la consommation de puissance totale dépend du type de cellule et de la technologie employée. Cependant, en général, on constate que :

- **Lorsqu'il n'y a pas de commutation** (transitions $0 \rightarrow 0$ et $1 \rightarrow 1$), la puissance totale reste plus ou moins constante. En effet, la puissance dynamique étant nulle, on ne retrouve dans le calcul de puissance totale que la puissance statique. Autrement dit : $P_{total} = P_{stat}$.
- **Lorsqu'il y a une commutation** (transitions $0 \rightarrow 1$ et $1 \rightarrow 0$), la puissance totale augmente. En effet, en plus de la puissance statique déjà présente, vient s'ajouter la puissance dynamique. Autrement dit : $P_{total} = P_{stat} + P_{dyn}$.

Ainsi, on peut conclure que la consommation de puissance dynamique des circuits CMOS correspond à la puissance qui est consommée par le circuit lorsqu'il y a une commutation dans la cellule. Cette puissance constitue un facteur dominant dans la consommation de puissance totale. Il est donc primordial de pouvoir la calculer.

Calcul de la puissance dynamique : Le calcul de la consommation de puissance dynamique se divise en deux parties. Pour mieux comprendre pourquoi, reprenons l'exemple de l'inverseur CMOS.

1. **Puissance moyenne de chargement de la capacité :** La figure 3.1 présente le schéma de l'inverseur CMOS lorsqu'il y a une commutation de sa sortie de l'état 0 à l'état 1 (schéma b) et lorsqu'il y a une commutation de sa sortie de l'état 1 à l'état 0 (schéma c). Pour rappel, le fonctionnement de l'inverseur est le suivant : Lorsque le signal d'entrée est à 1 (ou 0), le transistor du dessous est passant (ou bloqué) alors que celui du haut est bloqué (ou passant), la sortie est donc à 0 (ou 1). Maintenant, il faut regarder dans le cas d'une commutation à la sortie de l'inverseur CMOS. Si il y a une commutation de l'état 0 à l'état 1 en sortie (schéma b), l'inverseur dessine un courant provenant de l'alimentation (V_{DD}) et venant charger le condensateur C_L . Ce courant est appelé *courant de charge*. À contrario, lors d'une commutation de l'état 1 à l'état 0, l'inverseur décharge le courant du condensateur C_L vers la masse (GND). Ainsi, on constate bien que la commutation à la sortie de l'inverseur génère un courant qui produira une partie de la consommation de puissance dynamique. La consommation de puissance moyenne de chargement de la capacité durant un temps T peut être calculée de la façon suivante (3.3) :

$$P_{chrg} = \frac{1}{T} \int_0^T p_{chrg}(t) dt = \alpha \cdot f \cdot C_L \cdot V_{DD}^2 \quad (3.3)$$

Où :

- $p_{chrg}(t)$ représente la consommation de puissance de chargement instantanée de la cellule.
- α est le facteur d'activité. Il correspond au nombre moyen de transitions (0-1) en sortie de la cellule à chaque coup de clock.
- f représente la fréquence de clock.
- C_L représente la valeur de capacité du condensateur.
- V_{DD} représente la tension positive de l'alimentation.

2. **Puissance moyenne causée par les courants de court-circuit :** En plus de la puissance moyenne de chargement de la capacité, il existe lors d'une commutation un bref instant, durant lequel les deux transistors conduisent le courant. Cela a pour effet de créer un court-circuit entre V_{DD} en GND . Ce court-circuit va dissiper, le temps de son passage, de la puissance. La consommation de puissance moyenne qui est causée par les courants de court-circuit dans une cellule durant un temps T peut être calculée de la façon suivante (3.4) :

$$P_{cc} = \frac{1}{T} \int_0^T p_{cc}(t) dt = \alpha \cdot f \cdot C_L \cdot V_{DD} \cdot I_{fuite} \cdot t_{cc} \quad (3.4)$$

Où :

- $p_{cc}(t)$ représente la puissance de court-circuit consommée par la cellule.
- α est le facteur d'activité. Il correspond au nombre moyen de transitions (0-1) en sortie de la cellule à chaque coup de clock.
- f représente la fréquence de clock.
- V_{DD} représente la tension positive de l'alimentation.
- I_{fuite} représente le courant de fuite causé par le court-circuit.
- t_{cc} représente le temps durant lequel le court-circuit se produit.

En conclusion, les devices cryptographiques modernes (comme le FPGA) possèdent deux composantes en puissance :

- Une **puissance statique** de faible valeur, requise pour garder le device en fonctionnement continu. Elle dépend du nombre de transistors dans la cellule. Elle est **négligée**.
- Une **puissance dynamique** de haute valeur, qui apparaît lors d'une commutation. Elle dépend des opérations exécutées et des données manipulées. Elle n'est **pas négligée**.

Ceci nous conduit donc à l'équation suivante (3.5) :

$$P_{total} = P_{stat} + P_{dyn} \cong P_{dyn} = P_{chrg} + P_{cc} \quad (3.5)$$

3.1.2 Composition des traces de puissance

Les attaques basées sur l'analyse de la consommation de puissance exploitent le fait que la consommation de puissance d'un device cryptographique dépend des **opérations qu'il exécute** et des **données qu'il manipule**. Ces deux informations vont ainsi permettre de définir différentes propriétés intéressantes. Pour chaque point analysé dans une trace de puissance, on notera :

- P_{op} la **composante dépendante de l'opération exécutée** ;
- P_{data} la **composante dépendante de la donnée manipulée**.

De plus, une troisième composante doit également être prise en compte. Cette composante fait référence au **bruit électrique** (aussi appelé bruit de fond) et sera notée P_{noise} . En effet, un signal est toujours affecté de petites fluctuations plus ou moins importantes. Ces fluctuations, dont les origines peuvent être diverses, sont appelées "bruit électrique" (ou simplement bruit). Le bruit est considéré comme un élément parasite aléatoire, c'est-à-dire qu'on ne sait pas le déterminer à l'avance. Au plus cette composante est élevée et au plus l'analyse de la consommation de puissance est difficile.

Ainsi, chaque point d'une trace de puissance peut être modélisé comme la somme des 3 composantes définies ci-dessus, soit : $P_{total} = P_{op} + P_{data} + P_{noise}$. Par ailleurs, en reprenant l'équation (3.5) définie dans la section 3.1.1, nous pouvons conclure que (équation 3.6) :

$$P_{total} = P_{op} + P_{data} + P_{noise} = P_{stat} + P_{dyn} \cong P_{dyn} = P_{chrg} + P_{cc} \quad (3.6)$$

En conclusion, la puissance consommée par un device cryptographique reflète directement ses activités internes. En effet, on remarque que cette puissance dépend aussi bien des opérations exécutées que des données manipulées, auxquelles il faut également ajouter la présence d'un bruit parasite inconnu (aléatoire).

3.2 Modèles de puissance

Dans une attaque par analyse de la consommation de puissance, l'attaquant doit utiliser ce que l'on appelle un **modèle de puissance** afin de prédire la consommation de puissance du device cryptographique attaqué. Différents modèles de prédictions existent. Chacun de ces modèles se base sur les valeurs de bits dans des set de données. La qualité du modèle employé a un impact important sur l'efficacité de l'attaque. Deux modèles sont généralement définis et utilisés : Il s'agit des modèles de **Poids de Hamming** (*Hamming Weight - HW*) et de **Distance de Hamming** (*Hamming Distance - HD*). Il faut noter que ces deux modèles restent généraux, c'est-à-dire qu'ils ne requièrent presque aucune connaissance à propos du design du circuit et sont par conséquent parfois imprécis. Une fois les prédictions obtenues, celles-ci sont comparées selon diverses méthodes (voir chapitre 4) aux mesures réelles de consommation de puissance du device capturées à l'oscilloscope (traces de puissance).

3.2.1 Modèle de poids de Hamming

Le poids de Hamming est le modèle de consommation de puissance le plus élémentaire. C'est celui le plus utilisé par un attaquant lorsqu'il s'agit d'estimer la consommation de puissance d'un circuit dont on ne connaît pas certaines valeurs intermédiaires consécutives calculées durant l'exécution de l'algorithme. Ce modèle considère qu'un 0 ne mène à aucune quantité significative de consommation de puissance tandis qu'un 1 implique une quantité significative de puissance consommée. Ainsi, pour ce modèle, on assume que la consommation de puissance prédictive est proportionnelle au nombre de bits à 1 d'une donnée traitée. Dit vulgairement, le poids de Hamming calcule le nombre de bits à 1 présents dans un nombre binaire. Mathématiquement, cela se traduit par l'expression 3.7 :

$$HW(b_0) = \sum_{i=0}^{N-1} b_{0,i} \quad (3.7)$$

Où b_0 est un mot de N bits. Par conséquent, $b_{0,i}$ est le i^{me} bit du mot binaire b_0 .
Exemple : $HW(100110) = 3$.

3.2.2 Modèle de distance de Hamming

La distance de Hamming est un modèle de consommation de puissance proposé par Brier et Al. Il est basé sur la relation entre la consommation de puissance et l'activité de commutation dans les circuits en technologie CMOS. En effet, comme indiqué en conclusion de la section 3.1.1, la consommation de puissance d'un device en technologie CMOS est principalement d'ordre dynamique, c'est-à-dire due aux activités de commutation des cellules dans le circuit. Ainsi, ce modèle assume que la puissance totale consommée par un circuit CMOS (définie selon l'équation 3.5) est équivalente à la consommation de puissance lors de commutations (transitions $0 \rightarrow 1$ et $1 \rightarrow 0$). Ce modèle est donc proportionnel au nombre de transitions $0 \rightarrow 1$ et $1 \rightarrow 0$. La distance de Hamming entre 2 nombres binaires, b_1 et b_2 se calcule en comptant le nombre de transitions ($0 \rightarrow 1$ et $1 \rightarrow 0$) entre ces 2 nombres, soit l'expression 3.8 :

$$HD(b_1, b_2) = HW(b_1 \oplus b_2) \quad (3.8)$$

Exemple : $HD(110110,100100) = 2$.

Chapitre 4

Attaques par canaux auxiliaires

Ce quatrième chapitre est sans conteste celui le plus attendu par le lecteur. Tout d'abord, une introduction générale est donnée au lecteur afin que ce dernier connaisse les potentialités des différents attaques par canaux auxiliaires existantes. Ensuite, l'attention est plus particulièrement portée sur trois types d'attaques par analyse de la consommation de puissance. La première, l'attaque par l'analyse simple de la consommation, permet de comprendre aisément le concept des attaques. La seconde, l'attaque par l'analyse différentielle de la consommation, est plus complexe mais apporte déjà de meilleurs taux de succès pour l'attaquant. Enfin, la troisième, l'attaque par l'analyse de la consommation par corrélation, est sans nul doute celle présentant le plus grand taux de succès parmi les trois attaques citées. Pour cette raison, ce sera celle mise en oeuvre afin d'attaquer l'implémentation de l'algorithme AES sur un FPGA. Les résultats de cette attaque seront présentés au chapitre X (partie 2).

4.1 Introduction

À la fin des années 1990, une nouvelle contrainte pour la conception de systèmes informatiques a vu le jour : la sécurité matérielle. Bien souvent, la sécurité d'un système informatique s'appuie plus sur les concepts software que hardware. Cependant, un nouveau mode d'attaque s'est développé. Il s'agit d'attaques physiques, c'est-à-dire d'attaques réalisées sur le circuit électronique lui-même. En général, le but d'une attaque est de retrouver la clé de chiffrement utilisée par l'algorithme afin de déchiffrer des données sensibles. Deux grandes familles d'attaques sont recensées :

- **Attaques actives** : Une attaque est dite active lorsque les entrées et/ou l'environnement du device cryptographique sont manipulés par l'attaquant en vue de produire un comportement anormal du device. La clé secrète est révélée en exploitant les données issues de ce comportement anormal. Cela peut être une variation de la tension du device, une injection de glitch d'horloge, etc... On distingue deux types d'attaques actives :
 - Les attaques actives **irréversibles** qui conduisent à la destruction du device cryptographique. Ce type d'attaque est souvent réalisé pour connaître la conception physique d'un device. *Exemple* : Découpage laser d'un circuit intégré.
 - Les attaques actives **pseudo-réversibles** qui n'entraînent pas forcément la destruction du device cryptographique, mais qui sont souvent tout de même invasives puisqu'elles nécessitent la préparation du circuit (découpe partielle du boîtier du circuit intégré par exemple). Un exemple typique de ce type d'attaque est ce qu'on appelle les *attaques en fautes*. Le principe est d'introduire volontairement des fautes dans le circuit (exemple : Injection de rayon laser, injection de glitch d'horloge, etc.). Les fautes ainsi créées peuvent entraîner le circuit dans des modes de fonctionnement conduisant à des erreurs. Ces erreurs peuvent ensuite être exploitées pour déterminer la clé.
- **Attaques passives** : Une attaque est dite passive lorsque l'attaquant exploite l'analyse, en fonctionnement normal, d'informations s'échappant d'un device cryptographique. Cela peut être l'analyse de

la consommation de puissance, l'analyse temporelle, l'analyse par rayonnement électromagnétique, etc... C'est ce type d'attaque qui sera détaillé tout au long de ce stage et durant la réalisation du TFE.

La figure 4.1 ci-dessous résume les différents types d'attaques physiques possibles.

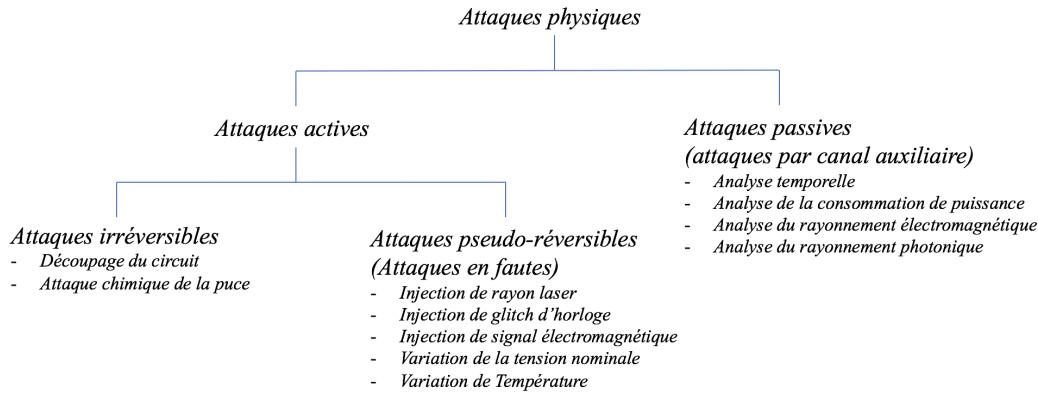


Figure 4.1 : Les 2 grandes familles d'attaques physiques possibles. La suite de ce rapport se concentre essentiellement sur les attaques physiques dites passives.

Les attaques passives sont globalement beaucoup plus simples à mettre en oeuvre que les attaques actives. Comme définit précédemment, ces attaques consistent à analyser des données issues de canaux auxiliaires au device cryptographique (lorsque ce dernier est en état de fonctionnement normal). Ces canaux auxiliaires sont des canaux présents physiquement sur le circuit attaqué et le long desquels de l'information s'échappe (sous différentes formes : rayonnement électromagnétique, rayonnement photonique, consommation de puissance, etc.). C'est là qu'intervient la notion de *side-channel attacks* ou en français *l'attaque par canal auxiliaire*. En effet, les fonctions cryptographiques, bien que pouvant être extrêmement robustes théoriquement (c'est-à-dire mathématiquement) sont très sensibles aux fuites d'informations. C'est-à-dire qu'une quantité très faible d'informations peut être exploitée pour casser un algorithme cryptographique très fort. C'est ce que les attaques par canaux auxiliaires exploitent.

La figure 4.2 ci-dessous présente les différentes façons possibles d'attaquer **passivement** un device.

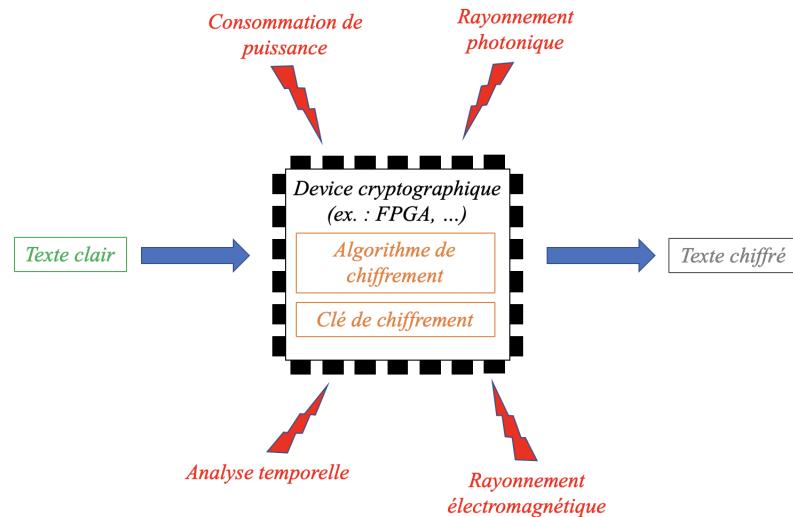


Figure 4.2 : Les différentes façons d'attaquer passivement un device cryptographique en vue de casser l'algorithme de chiffrement qu'il exploite.

Dans la suite de cet ouvrage, on se concentrera sur un type précis d'attaque passive : l'attaque sur **l'analyse de la consommation de puissance**. Les sections suivantes décrivent le principe de

fonctionnement de ce type d'attaque.

4.2 Analyse de la consommation de puissance

4.2.1 Analyse simple de la consommation

4.2.2 Analyse différentielle de la consommation

4.2.3 Analyse de la consommation par corrélation

Coefficient de corrélation

Par définition, le **coefficient de corrélation** est un coefficient statistique permettant de mettre en évidence une liaison entre deux types de séries de données statistiques. La valeur du coefficient de corrélation est comprise entre -1 et 1. Le coefficient de corrélation se calcule de la façon suivante (4.1) :

$$r(X; Y) = \frac{\sum(X - \bar{X}).(Y - \bar{Y})}{\sqrt{\sum(X - \bar{X})^2} \cdot \sqrt{\sum(Y - \bar{Y})^2}} \quad (4.1)$$

Où

- X et Y sont deux séries de données statistiques.
- \bar{X} et \bar{Y} sont les moyennes respectives des variables X et Y .

Sur base de la valeur du coefficient de corrélation, on peut conclure que :

- Si la valeur absolue du coefficient de corrélation est élevée (proche de 1) alors c'est qu'il y a une forte liaison entre les deux séries analysées.
- Si la valeur absolue du coefficient de corrélation est faible (proche de 0) alors c'est qu'il y a une très faible liaison, voire aucun lien entre les deux séries analysées.

L'attaque CPA en concret

En 1999, Paul Kocher a révolutionné le monde de la sécurité matérielle en publiant une nouvelle attaque sur canaux cachés exploitant la mesure de la consommation de puissance d'un circuit en cours de chiffrement. La faille utilisée est la corrélation, dans les circuits électroniques de technologie CMOS, entre la consommation de puissance dynamique et le nombre de transistors qui commutent (de 0-1 et de 1-0). Comme précisé dans la section 11, il existe différents types d'attaques par analyse de la consommation de puissance. Celle définie par Kocher en 1999 est une attaque de type DPA (elle ne sera pas abordée dans cet ouvrage). En 2004, cette attaque DPA fut améliorée par des chercheurs qui décidèrent de lui donner un nouveau nom : **Attaque CPA** pour *Correlation Power Analysis*. Ce nom se justifie par le fait que cette attaque calcule la corrélation entre la consommation de puissance mesurée à l'oscilloscope et la consommation de puissance simulée par logiciels informatiques (et plus précisément pas des modèles de puissance, comme définit à la section 3.2). L'annexe B.1 présente le principe de fonctionnement d'une attaque CPA.

Pour réaliser une attaque CPA, nous avons principalement besoin de deux éléments :

Un oscilloscope : L'oscilloscope est utilisé pour enregistrer des traces de puissance. C'est-à-dire des mesures de tension en fonction du temps représentant la consommation de puissance du device cryptographique lorsqu'il chiffre une donnée (un *plaintext*). Plusieurs plaintexts mènent donc à plusieurs traces.

Un ordinateur : L'ordinateur est utilisé pour réaliser toute une série de calculs. Plus précisément, il réalise 3 grands types de calculs : calculs de simulation de l'algorithme de chiffrement, calculs de simulation du modèle de puissance et calculs des coefficients de corrélation.

Pour la suite de la description, on utilisera l'algorithme AES-128 comme algorithme de chiffrement et le modèle poids de Hamming comme modèle de puissance.

Principe de fonctionnement :

Comme le montre l'annexe B.1, avant de pouvoir calculer la corrélation, deux études doivent être réalisées :

1. Bloc 1 = oscilloscope : Une étude qui concerne la prise de mesures afin d'analyser la consommation de puissance du device cryptographique lorsqu'il chiffre différents plaintexts.
2. Bloc 2 = ordinateur : Une étude qui concerne la simulation de l'algorithme AES ainsi que la simulation de la puissance consommée.

Le bloc 1 est simple à réaliser. À l'aide d'un oscilloscope, on va enregistrer différentes traces représentant la consommation de puissance du device lorsqu'il chiffre différents plaintexts. Ainsi, pour N plaintexts, on enregistre N traces de puissance.

Le bloc 2 est plus complexe. Il est composé de deux parties importantes :

1. Simulation de l'algorithme AES : Comme vu à la section ??, l'algorithme AES-128 exécute quatre opérations, à savoir *AddRoundKey*, *SubBytes*, *ShiftRows* et *MixColumns*, répétées dix fois en raison des dix rounds. Parmi ces quatre opérations, trois sont des fonctions linéaires (*AddRoundKey*, *ShiftRows* et *MixColumns*) et une est une fonction non-linéaire (*SubBytes*). Comme le montre l'annexe B.1, l'attaque CPA exploite les valeurs intermédiaires en sortie de l'opération *SubBytes* lors du premier round. En pratique, l'attaque CPA peut exploiter les valeurs intermédiaires en sortie des trois autres opérations. Cependant, de part sa propriété de fonction non-linéaire, l'opération *SubBytes* nécessite moins de traces pour retrouver la valeur de la clé. C'est pourquoi, il ne faut simuler que les deux premières opérations de l'algorithme AES-128 (*AddRoundKey* - *SubBytes*) pour réaliser une attaque CPA.

Rappelons que les opérations de l'AES-128 s'exécutent byte par byte sur les 16 bytes (128 bits) de la matrice STATE et de la matrice clé. Ainsi, on peut étudier chaque byte de façon séparée. Prenons le cas simple de l'étude du premier byte, i.e que l'on ne considère que le premier byte de notre plaintext et que le premier byte de notre clé. La problématique consiste alors à retrouver la valeur du premier byte de la clé connaissant la valeur du premier byte des différents plaintexts. Étant donné qu'un byte correspond à 8 bits, il existe 2^8 soit 256 valeurs possibles pour le premier byte de la clé. De cette façon, lors de la simulation, si l'on souhaite retrouver le premier byte de la clé, on va devoir simuler pour chaque plaintext, 256 valeurs de clé.

2. Simulation de la puissance consommée : Comme vu à la section 3.2, l'attaquant doit utiliser ce qu'on appelle un modèle de puissance afin de prédire la consommation de puissance du device cryptographique attaqué. Pour une attaque CPA, le modèle de puissance privilégié est le poids de Hamming. Ainsi, comme le montre l'annexe B.1, on calcule le poids de Hamming des valeurs intermédiaires obtenues en sortie de l'opération *SubBytes*. Ce poids de Hamming représente la puissance consommée par le device lorsqu'il chiffre un plaintext. Il sera ensuite comparé (par calcul de corrélation) à la puissance réellement consommée et mesurée à l'aide de l'oscilloscope.

Une fois les traces de puissance sauvegardées et les calculs de simulation du bloc 2 obtenus, nous pouvons calculer la corrélation (bloc 3) entre les données résultantes du bloc 1 et les données résultantes du bloc 2. Plus précisément, ce calcul de coefficient de corrélation va permettre de mettre en évidence la liaison qui existe entre les traces de puissance et les poids de Hamming.

Reprendons le cas simple où l'on cherche uniquement à retrouver le premier byte de la clé. Pour un plaintext donné, nous allons obtenir une trace de puissance d'une part et un calcul de poids de Hamming d'autre part. En pratique, la trace de puissance est présentée sous forme d'une matrice de taille NxS où N représente le nombre de plaintexts et S représente le nombre d'échantillons dans la trace (voir annexe B.2 pour visualiser la situation). Parallèlement, le résultat du calcul du poids de Hamming est présenté sous forme d'une matrice de taille 1x256 où '1' représente le premier byte du 'premier' plaintext et '256' représente les '256' valeurs de clé possibles à tester. Ainsi, pour N plaintexts, le résultat HW sera une matrice de taille Nx256. Le calcul du coefficient de corrélation va alors être réalisé pour chaque résultat du poids de Hamming (256 résultats différents) avec chaque trace enregistrée. Le résultat de cette corrélation sera représenté sous forme d'une matrice de taille Sx256. En effet, Si on a 500 échantillons dans une trace alors le résultat de la

corrélation sera de taille 500x256. On peut voir ce résultat comme la corrélation entre les 256 valeurs de clé testées pour chacune des traces caractérisées par 500 points (échantillons).

Chapitre **5**

Contre-mesures

5.1 Contre-mesures Hiding

5.2 Contre-mesures Masking

5.3 Contre-mesures Faking

Deuxième partie

Acquis pratiques

Chapitre **6**

Configuration de l'attaque

6.1 Méthodologie

6.2 Configuration du FPGA

6.3 Configuration de l'oscilloscope

6.4 Configuration de l'attaque CPA

Chapitre **7**

Mise en application d'une attaque CPA (*Correlation Power Analysis*)

7.1 Résultats de simulations

7.2 Résultats expérimentaux

7.3 Conclusion

Chapitre **8**

Implémentation de la contre-mesure de type *faking*

8.1 Développement de la contre-mesure

8.1.1 Résultats expérimentaux

8.2 Conclusion

Chapitre **9**

Évaluation des performances de la contre-mesure

9.1 Évaluation par métriques

9.1.1 Évaluation par critères de performances

Chapitre 10

Conclusion

Consommation de puissance

Introduction aux attaques par analyse de la consommation de puissance

Nous allons donc étudier un cas précis d'attaque passive : l'attaque par **l'analyse de la consommation de puissance**. Comme son nom l'indique, ce type d'attaque analyse la consommation de puissance du device cryptographique attaqué pour retrouver des informations sensibles. En effet, la consommation de courant d'un circuit électronique dépend de deux facteurs :

- Les opérations qui sont exécutées.
- Les données qui sont manipulées.

Ainsi, en mesurant un certain nombre de fois la consommation de puissance d'un circuit, il est possible de retrouver certaines informations telles que les opérations exécutées (afin d'identifier un algorithme par exemple) ou les informations secrètes (clé de chiffrement). Pour ce faire, un oscilloscope est utilisé afin de capturer et d'enregistrer des données, appelées *traces*, mesurées à partir des canaux auxiliaires du circuit électronique (dans notre cas, un FPGA). Pour réaliser la mesure, une résistance est placée en série avec le canal (la PIN) connecté à la tension d'alimentation du device cryptographique (V_{DD}). L'oscilloscope est alors en mesure d'enregistrer une différence de potentiel (notée $V(t)$) aux bornes de la résistance. Étant donné que les courants circulant dans le device cryptographique sont de valeurs très faibles (μA), la tension $V(t)$ est également très faible. Ainsi, un amplificateur est utilisé afin d'amplifier cette différence de potentiel. La figure 11.1 ci-dessous présente le principe de mesure à l'oscilloscope.

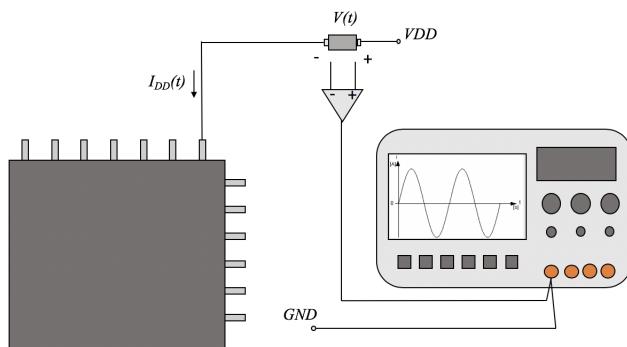


Figure 11.1 : Principe de mesure à l'oscilloscope.

On parle d'attaque par analyse de la consommation de puissance or avec un oscilloscope, on mesure une tension et non une puissance. Cependant, comme le démontre l'équation 11.2, la puissance consommée ($p(t)$) est proportionnelle à la tension consommée ($V(t)$). Il ne s'agit donc pas d'une erreur de parler de consommation de puissance. En effet, en supposant que la tension d'alimentation V_{DD} est constante et par

simple application de la loi d'Ohm, on a :

$$\begin{cases} u(t) = V_{DD} \\ i(t) = \frac{V(t)}{R} \end{cases} \quad (11.1)$$

En reprenant la définition de la puissance consommée et en y remplaçant les termes $u(t)$ et $i(t)$, on a :

$$p(t) = u(t).i(t) = V_{DD} \cdot \frac{V(t)}{R} \quad (11.2)$$

En pratique, il existe différents types d'attaques par analyse de la consommation de puissance : Les attaques SPA (*Simple Power Analysis*), les attaques DPA (*Differential Power Analysis*), les attaques CPA (*Correlation Power Analysis*), les attaques par template, etc.

Nous allons en définir une plus précisément, l'attaque CPA. C'est ce type d'attaque qui a été mis en place durant le stage afin de tenter de casser l'algorithme AES. Ainsi, en considérant les attaques par analyse de la consommation de puissance et en prenant le cas particulier d'une attaque dite CPA, nous pouvons énoncer le problème étudié de la façon suivante :

En supposant connus les messages clairs envoyés au device cryptographique et en supposant que ce dernier implémente l'algorithme AES, nous simulerons sur ordinateur le poids de Hamming de chaque donnée binaire obtenue en sortie de l'opération *SubBytes*. Ensuite, nous calculerons les différentes valeurs des coefficients de corrélation entre les traces de puissance capturées à l'oscilloscope et le poids de Hamming obtenu par simulation (sur ordinateur). Sur base de cette étude de la corrélation, nous serons (en principe) capable de déterminer la clé secrète, c'est-à-dire casser l'algorithme AES et ainsi exploiter les données confidentielles. La figure 11.2 présente de façon synthétique le principe général d'une attaque CPA.

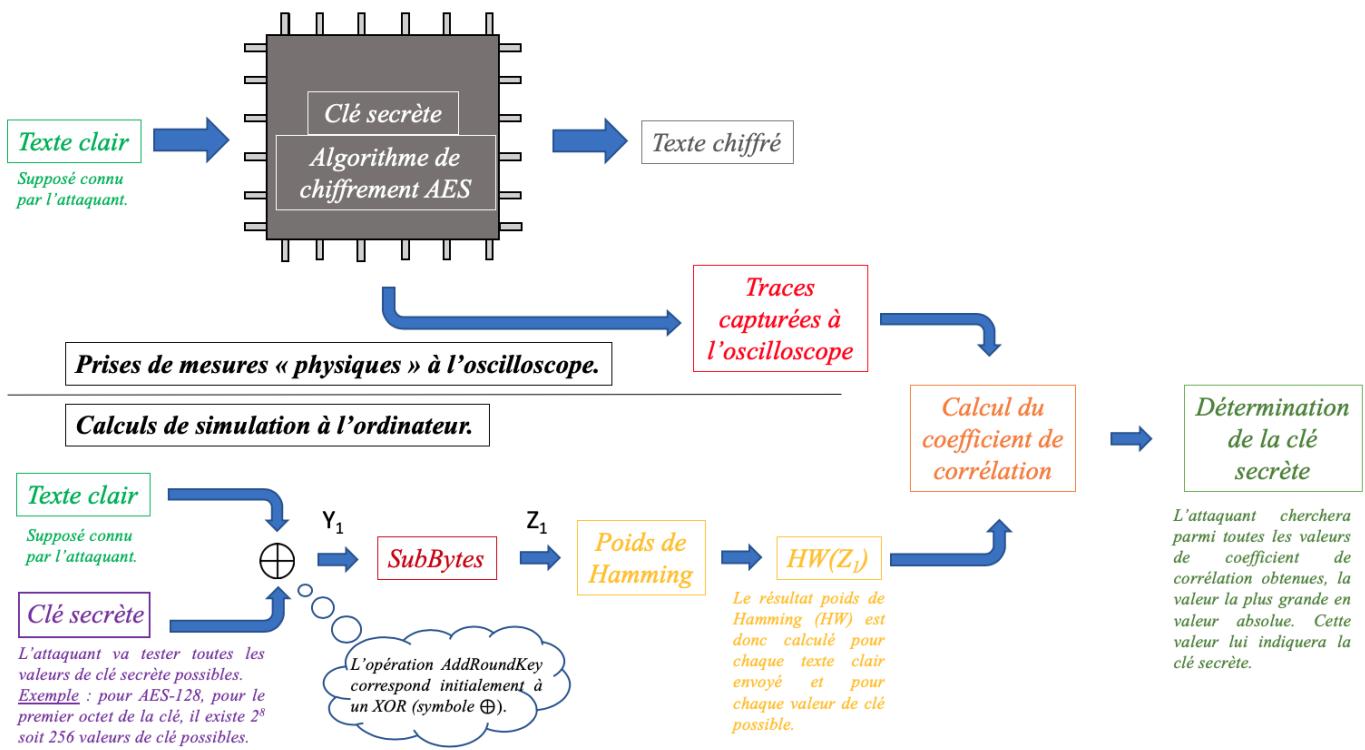


Figure 11.2 : Principe général d'une attaque CPA.

Le principe d'une attaque par consommation de puissance et plus particulièrement d'une attaque CPA, ayant été introduit de manière générale, nous allons maintenant revenir sur différentes notions citées ci-dessus afin de mieux les définir et ainsi mieux comprendre le raisonnement qui se cache derrière une attaque par calcul de corrélation (CPA). La section 4.2.3 définit toutes les notions élémentaires pour réaliser une attaque CPA.

Afin de mieux appréhender le fonctionnement d'une attaque CPA, la section 11.0.1 présente deux exemples de simulation sur MATLAB.

11.0.1 Simulations sur MATLAB

Afin de bien assimiler une attaque par canal auxiliaire, il m'a été demandé de tester, par des simulations sur le logiciel MATLAB, les notions théoriques développées précédemment. Deux exercices différents ont ainsi été réalisés.

- Simuler un point d'une trace et ensuite réaliser une attaque CPA sur ce point.** La première phase de la simulation a pour objectif de générer un point particulier d'une trace sur base de l'algorithme AES-128. La seconde phase de la simulation a pour but de réaliser une attaque par canal auxiliaire. Plus précisément, il s'agit d'une attaque CPA. Pour cette raison, seules les 2 premières étapes de l'algorithme AES-128 sont nécessaires et seront donc simulées (*AddRoundKey*, *SubBytes*). À noter que, pour simplifier, cette attaque n'est réalisée que sur un seul byte de données et donc aussi un seul byte de clé.
- Réaliser une attaque CPA à partir de traces réelles.** Dans ce cas de figure, on connaît 4 paramètres : les messages clairs envoyés (plaintexts), les traces capturées à l'oscilloscope, le nombre de traces ainsi que le nombre d'échantillons. Ainsi, sur base des traces qui nous sont fournies, l'objectif est de tenter de retrouver la clé secrète en réalisant une attaque CPA. La différence majeure avec l'exercice précédent est que l'on étudie une trace selon l'ensemble de points (les échantillons) qui la caractérise. Cet ensemble de traces étant par ailleurs fourni sur base de mesures réalisées à l'oscilloscope.

Exercice 1 : La figure 11.3 ci-dessous représente le schéma-bloc de la première phase de l'exercice 1, c'est-à-dire qu'elle présente les différentes étapes à réaliser pour simuler un point d'une trace.

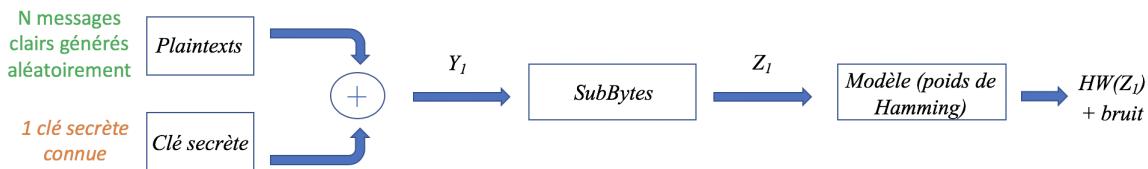


Figure 11.3 : Schéma-bloc permettant de comprendre la simulation d'un point d'une trace.

Sur le schéma-bloc, nous percevons deux entrées : la clé secrète (connue) utilisée pour le chiffrement ainsi que les N messages clairs devant être chiffrés. Nous réalisons ensuite les deux premières étapes de l'algorithme AES, à savoir les opérations *AddRoundKey* (correspondant à un *XOR*) et *SubBytes* respectivement. Le résultat obtenu à la sortie de l'opération *AddRoundKey* est noté Y_1 alors que celui obtenu à la sortie de l'opération *SubBytes* est noté Z_1 . Ensuite, un modèle de puissance est utilisé afin d'imiter au mieux la consommation de puissance du circuit. Ce modèle de puissance est le *poids de Hamming*. Le but est donc de compter le nombre de bits à '1' pour chaque octet de données Z_1 . Le résultat obtenu est noté $HW(Z_1)$. Enfin, une fois que le poids de Hamming a été calculé, on ajoute du bruit afin de rendre la simulation plus réelle. En effet, lors d'une prise de mesure en situation réelle, un élément parasite vient toujours s'ajouter au signal que l'on étudie, il s'agit de bruit électronique.

La figure 11.4 ci-dessous représente le schéma-bloc de la seconde phase de l'exercice 1. Il présente ainsi les différentes étapes à réaliser qui serviront *in fine* à réaliser l'attaque CPA.

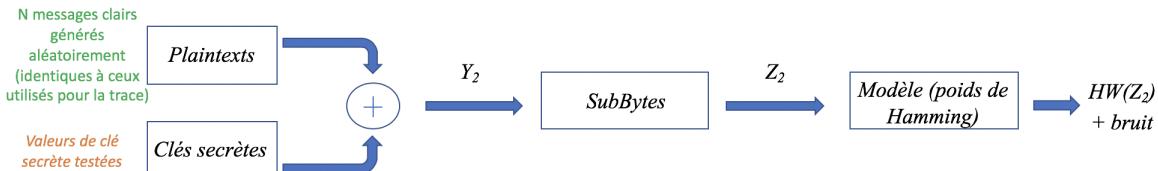


Figure 11.4 : Schéma-bloc permettant de comprendre la seconde phase de l'exercice 1.

La figure 11.4 est quasi identique à la figure 11.3. La seule différence concerne l'une des deux entrées. En effet, le but de l'attaque CPA est de retrouver la clé secrète utilisée dans la figure 11.3 pour chiffrer nos N messages. Ainsi, si on ne s'intéresse qu'à un seul octet de données et donc à un seul octet de clé, on

va tester les 256 (2^8) valeurs de clé possibles. En notant respectivement Y_2 et Z_2 les résultats obtenus en sortie des opérations *AddRoundKey* et *SubBytes*, il ne nous restera plus qu'à calculer le poids de Hamming et à ajouter du bruit pour ensuite tenter de retrouver la clé secrète par calcul du coefficient de corrélation.

Concrètement, sur MATLAB, tous ces calculs vont être opérés sur des matrices. La figure 11.5 présente les différentes tailles de matrices utilisées pour réaliser la simulation.

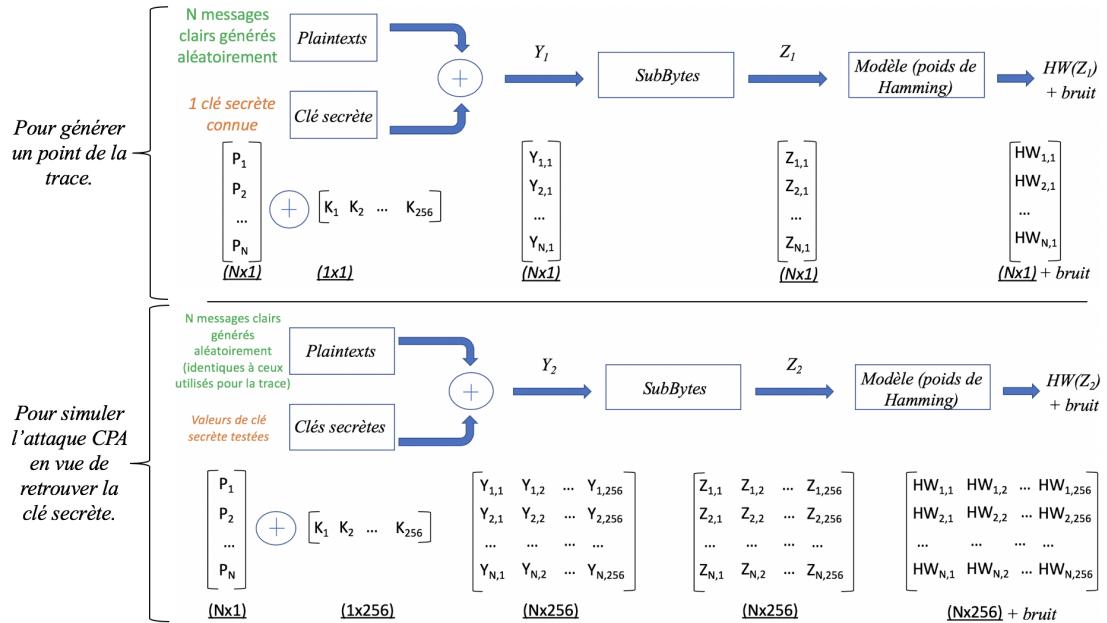


Figure 11.5 : Schéma-bloc permettant de visualiser la taille des différentes matrices employées pour la simulation.

Comme vu à la section ??, l'algorithme AES implémente deux matrices en entrée : la matrice STATE et la matrice clé. Ces deux matrices contiennent 16 éléments correspondants aux 16 octet de données (pour la matrice STATE) et aux 16 octets de clé (pour la matrice clé). Autrement dit, ces deux matrices sont de taille 4x4. Pour le test, nous allons simplifier le procédé. En effet, en pratique les opérations s'exécutent octet par octet. Dans notre cas, on ne va s'intéresser qu'au premier octet de la matrice STATE et donc, par la même occasion, au premier octet de la matrice clé. **Le but final de la simulation sera donc de retrouver le premier octet de la clé connaissant le premier octet des N messages clairs devant être chiffrés.** Ainsi :

- La matrice STATE est représentée par une matrice de taille Nx1. N représente le nombre de messages clairs envoyés et "1" représente le nombre d'octets analysés, i.e 1 octet (sur les 16).
- La matrice clé, dans le cas de la génération d'un point de la trace, est représentée par une matrice de taille 1x1. Un seul octet de l'unique clé secrète est en effet utilisé.
- La matrice clé, dans le cas de la simulation par ordinateur, est représentée par une matrice de taille 1x256. En effet, le but étant de retrouver la valeur du premier octet de la clé secrète, il existe 2^8 soit 256 valeurs possibles.
- Y_1 et Y_2 sont des matrices de taille Nx1 et Nx256 respectivement. En effet, pour Y_1 , une seule clé est utilisée alors que pour Y_2 , 256 valeurs de clé sont utilisées.
- Z_1 et Z_2 sont des matrices de taille Nx1 et Nx256 respectivement. En effet, l'opération *SubBytes* ne modifie pas la taille des données obtenues précédemment (Y_1 et Y_2).
- Enfin, $HW(Z_1)$ et $HW(Z_2)$ sont des matrices de taille Nx1 et Nx256 pour les mêmes raisons que celles évoquées précédemment.

Pour rappel, le but final de la simulation est de retrouver le premier octet de la clé connaissant un point de la trace (obtenu par simulation) et connaissant les N messages clairs envoyés. Cela est rendu possible en calculant le coefficient de corrélation pour chaque valeur de clé possible. Le figure 11.6 ci-dessous présente le calcul final qui permettra de retrouver le premier octet de la clé secrète.

$$\text{corrélation} \begin{bmatrix} \text{HW}_{1,1} & \text{HW}_{1,1} \text{ HW}_{1,2} \dots \text{HW}_{1,256} \\ \text{HW}_{2,1} & ; \quad \text{HW}_{2,1} \text{ HW}_{2,2} \dots \text{HW}_{2,256} \\ \dots & \dots \quad \dots \quad \dots \\ \text{HW}_{N,1} & \text{HW}_{N,1} \text{ HW}_{N,2} \dots \text{HW}_{N,256} \end{bmatrix} = \begin{bmatrix} \text{Corr}_{1,1} \text{ Corr}_{1,2} \dots \text{Corr}_{1,256} \\ \text{Corr}_{2,1} \text{ Corr}_{2,2} \dots \text{Corr}_{2,256} \\ \dots \quad \dots \quad \dots \\ \text{Corr}_{N,1} \text{ Corr}_{N,2} \dots \text{Corr}_{N,256} \end{bmatrix}$$

(Nx1) (Nx256) (Nx256)

Figure 11.6 : Calcul du coefficient de corrélation entre un point d'une trace simulée et le poids de Hamming.

Les graphes 11.7, 11.8 et 11.9 ci-dessous présentent les résultats obtenus. Il s'agit de graphes indiquant les valeurs des coefficients de corrélation pour chacune des 256 valeurs de clé testées. À noter que la clé secrète utilisée pour chiffrer les données lors de la simulation vaut 200 (en décimal). Comme vu à la section 4.2.3, en toute logique, le coefficient de corrélation est maximum (en valeur absolue) pour la clé réellement utilisée pour le chiffrement des données. Cependant, on sait qu'en fonction du nombre de traces analysées, les valeurs du facteur de corrélation fluctuent dans un intervalle plus ou moins grand. Ainsi, au plus le nombre de traces sera élevé, au plus l'intervalle des valeurs de corrélation sera faible et au plus ce sera facile de repérer la clé de chiffrement. En effet, si on observe les trois graphes ci-dessous, on peut remarquer que pour 10 traces, le coefficient de corrélation varie entre -0,66 et 0,73 ; pour 100 traces, le coefficient de corrélation varie entre -0,33 et 0,24 ; pour 1000 traces, le coefficient de corrélation varie entre -0,22 et 0,18. En observant ces 3 figures, on remarque que 10 traces ne suffisent pas à retrouver la valeur exacte de la clé secrète (indiquée à 125). Par contre, avec 100 traces et 1000 traces, la valeur du coefficient de corrélation maximum culmine à 0,54 et à 0,58 respectivement et permet de retrouver la bonne clé secrète utilisée pour le chiffrement (200).

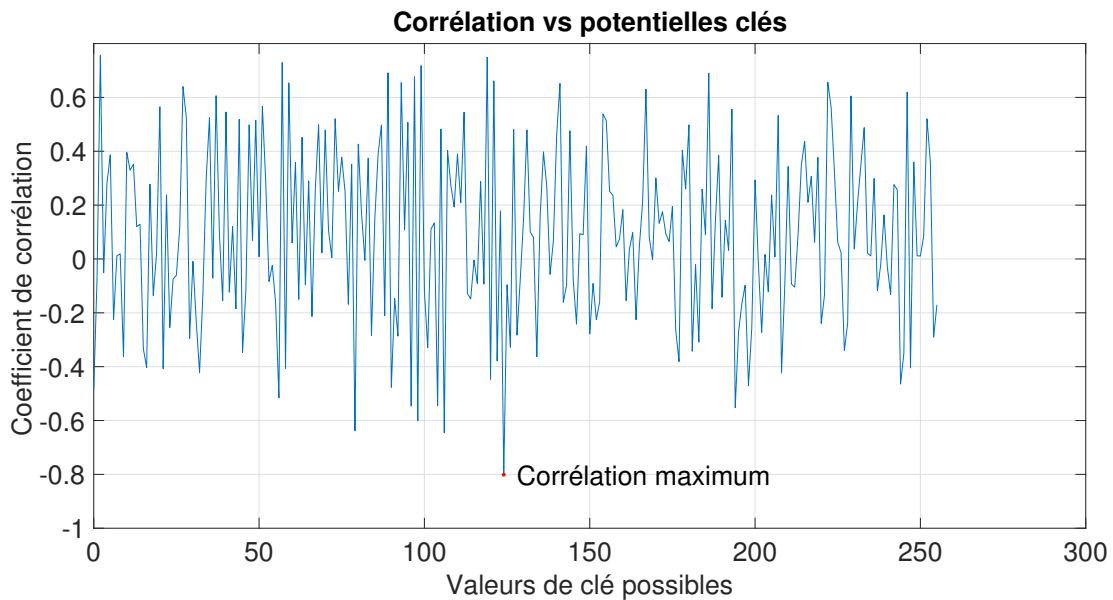


Figure 11.7 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse **10** traces.

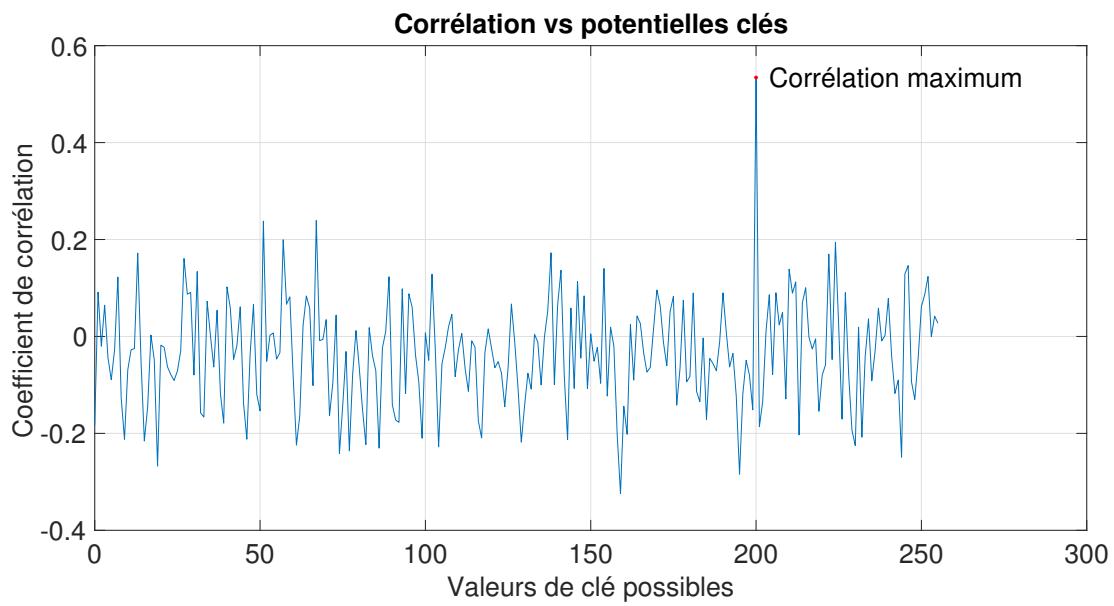


Figure 11.8 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse **100** traces.

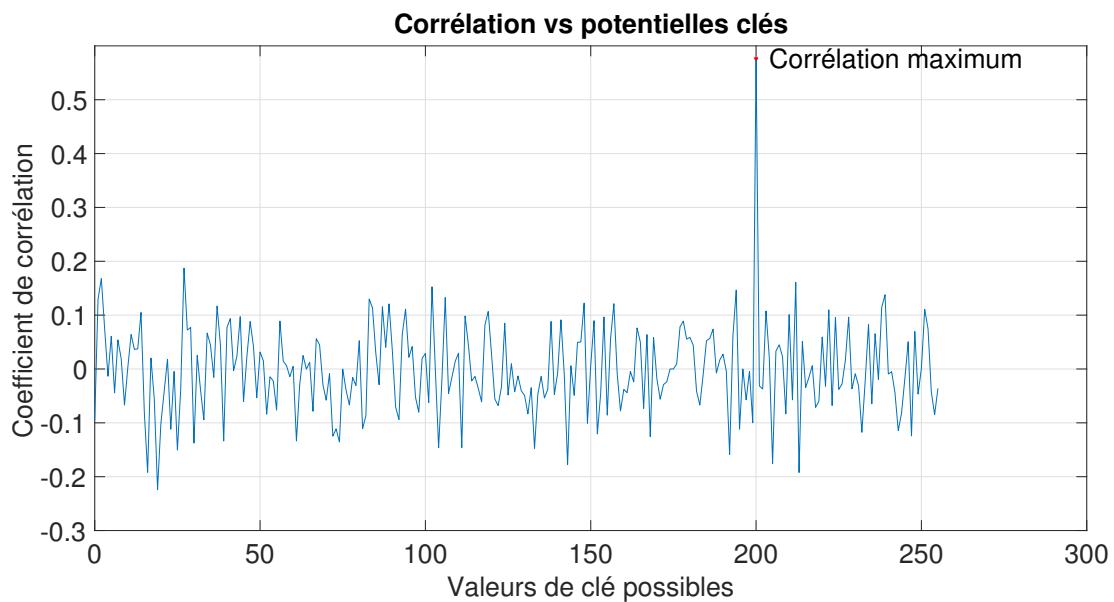


Figure 11.9 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse **1000** traces.

En général, pour montrer que l'augmentation du nombre de traces permet de trouver plus facilement la valeur de la clé secrète, on utilise un graphe comme celui présenté ci-dessous (figure 11.10).

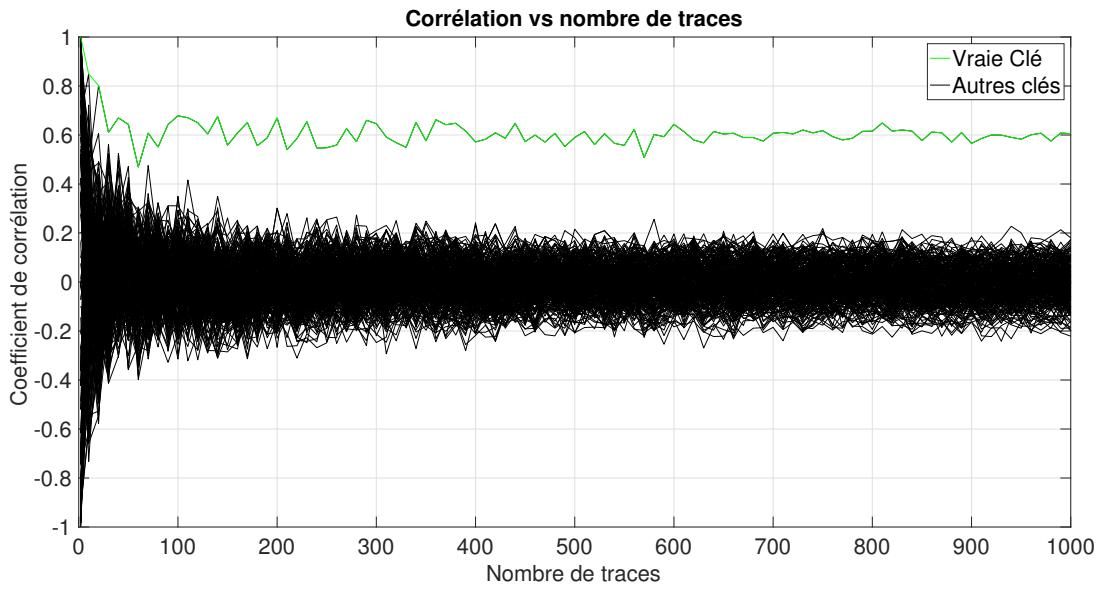


Figure 11.10 : Graphe présentant la valeur du coefficient de corrélation en fonction du nombre de traces. Plus le nombre de traces est élevé, plus la valeur de la clé secrète employée pour le chiffrement des données se distingue des autres clés. La vraie clé est 200.

Exercice 2 : La figure 11.11 ci-dessous représente le schéma-bloc de l'exercice 2. Dans cet exercice, on confronte les mesures obtenues à l'oscilloscope avec le poids de Hamming calculé à partir des messages clairs et pour chaque valeur de clé. À noter que dans cet exercice, on opère toujours sur le premier octet de données et le premier octet de la clé.

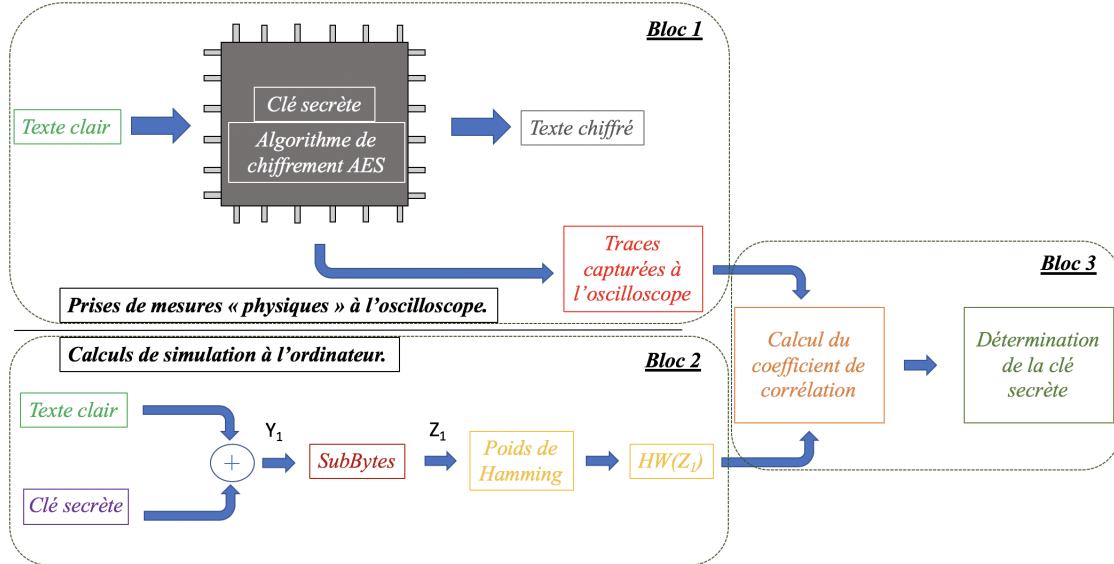


Figure 11.11 : Schéma-bloc permettant de comprendre le principe de fonctionnement du deuxième exercice.

On distingue 3 grands blocs sur le schéma ci-dessus :

- Premièrement, la prise de mesures à l'oscilloscope. Cette prise de mesures nous fournit un ensemble de traces. Plus précisément, l'oscilloscope a enregistré N traces différentes.
- Deuxièmement, des calculs de simulation de la consommation de puissance à partir du modèle du *poids de Hamming*. Pour chaque texte clair envoyé et pour chaque clé possible, on va calculer le poids de Hamming du résultat obtenu en sortie de l'opération *SubBytes*. Ce bloc est identique à la figure 11.4.
- Troisièmement, le calcul du coefficient de corrélation. Cette corrélation est effectuée pour chaque trace capturée avec chaque résultat du poids de Hamming.

Voici les données dont nous disposons pour l'exercice :

- On connaît le nombre de traces (N) enregistrées à l'oscilloscope. Il y en a 2380.
- On connaît le nombre d'échantillons pris dans une seule trace, i.e le nombre de points présents dans une trace. Il y en a 16384.
- On connaît tous les textes clairs envoyés au device cryptographique en vue d'être chiffrés. On a donc 2380 textes clairs connus. Chaque texte possédant une taille de 128 bits, soit 16 octets. Néanmoins, comme précisé précédemment, on ne s'intéresse qu'au premier octet.
- On connaît toutes les traces mesurées à l'oscilloscope. 2380 traces ont été enregistrées. Chaque trace contient 16384 points (échantillons).

Les 4 figures suivantes permettent de mieux comprendre, en pratique, l'objectif de l'exercice :

1. L'annexe B.3 présente les quatre premières traces, parmi les 2380, mesurées à l'oscilloscope.
2. L'annexe B.4 présente les valeurs des coefficients de corrélation obtenues pour quatre clés possibles, à savoir les clés 128, 129, 130 et 131. Chaque valeur de corrélation est calculée pour chacun des 16384 points (échantillons) des 2380 traces.
3. L'annexe B.5 présente en trois dimensions les valeurs des coefficients de corrélation obtenues pour les 256 valeurs de clé possibles. On remarque qu'il existe un coefficient de corrélation maximum, marqué par une étoile rouge sur la figure. Ce coefficient correspond à la clé 8 (en décimal). Cela signifie que le premier octet de la clé secrète vaut 8.

-
4. L'annexe **B.6** présente les valeurs des coefficients de corrélation obtenues pour une valeur de clé correspondant à 8. Cette valeur présente le plus grand coefficient de corrélation et correspond donc au premier octet de la clé secrète. Le coefficient de corrélation maximum vaut effectivement 0,5099.

11.0.2 Contre-mesures

Une fois que les attaques par analyse de la consommation de puissance ont été reconnues comme fonctionnelles, certaines entreprises (comme les banques par exemple) se devaient de trouver des moyens de contrer ces attaques afin d'assurer la confidentialité des données sensibles qu'elles manipulaient. Par conséquent, une série de contre-mesures a été développée à partir du début du deuxième millénaire. Cette section a pour objectif de présenter ces contre-mesures.

Pour rappel, les attaques par analyse de la consommation de puissance étudient des traces dont l'allure dépend essentiellement de deux facteurs : les opérations exécutées et les données manipulées. Pour protéger un appareil cryptographique de telles attaques, il faut donc casser les relations entre la consommation de puissance et les données sensibles manipulées et entre la consommation de puissance et les opérations exécutées. Les contre-mesures existantes sont divisées en deux catégories différentes :

1. **Les contre-mesures de type *Hiding*** : Le principe des contre-mesures de type Hiding est de rendre la consommation de puissance du device cryptographique indépendante des opérations exécutées et des données manipulées. Pour ce faire, on verra (section 11.0.2) qu'il existe deux approches différentes possibles.
2. **Les contre-mesures de type *Masking*** : Le principe des contre-mesures de type Masking est de générer des valeurs intermédiaires aléatoires. Ainsi, on accepte que la consommation de puissance du device cryptographique dépende des données manipulées. Cependant, on modifie (on masque) ces valeurs intermédiaires afin de fausser les traces de puissance obtenues à l'oscilloscope. On va donc construire des devices cryptographiques dont la consommation de puissance est liée aux données manipulées mais ces données sont faussées (modifiées) volontairement.

Il faut noter que les traces de puissance sont en pratique caractérisées par des traces en tension (voir section 11). Ainsi, sur un graphe, une trace de puissance est représentée par une courbe indiquant la tension mesurée en fonction du temps. Les contre-mesures développées pour les attaques par analyse de la consommation de puissance ont donc pour objectif de modifier l'allure de ces traces de puissances afin de compliquer la tâche de l'attaquant. Il existe deux façons de modifier une trace de puissance :

- En agissant sur *l'amplitude* de la trace, on parlera *d'intensité du leakage*.
- En agissant sur *la position dans le temps* de la trace, on parlera *d'instant du leakage*.

Avant de détailler chacune des deux catégories de contre-mesures, rappelons la définition de rapport signal à bruit (SNR : *Signal to Noise Ratio* en anglais). Le rapport signal à bruit ou SNR est un indicateur de performance. Plus précisément, son objectif est de mesurer la qualité de la transmission d'une information. Sa formulation mathématique est reprise à l'équation (11.3). Elle peut également être exprimée en dB (11.4) :

$$SNR = \frac{P_{signal}}{P_{noise}} \quad (11.3)$$

$$SNR_{dB} = 10 \cdot \log_{10} \frac{P_{signal}}{P_{noise}} \quad (11.4)$$

Où :

- SNR représente l'indicateur de performance de la transmission de l'information (sans unité/dB).
- P_{signal} représente la puissance du signal (en Watt).
- P_{noise} représente la puissance du bruit (en Watt).

Ainsi, le calcul du SNR permet d'évaluer si le signal de transmission que l'on étudie est fortement bruité ou non. Selon l'équation (11.3), cet indicateur SNR est d'autant plus élevé que la puissance du signal est élevée ou est d'autant plus élevé que la puissance du bruit est faible. Dans une transmission idéale, on désirera toujours un SNR très grand, signifiant que la trace obtenue représente majoritairement le signal et minoritairement le bruit (qui pour rappel est un élément parasite aléatoire).

Pour une trace de puissance, si le SNR d'une opération est élevé, cela signifie que la puissance du signal est plus élevée que la puissance du bruit, c'est-à-dire qu'il est plus facile de détecter des fuites d'information. Idéalement, il faut donc que le SNR soit proche de 0. De cette façon, le bruit recouvre tellement le signal qu'il est impossible de détecter du *leakage* (fuite). En pratique, cela peut être réalisé en diminuant la variance du signal vers 0 ou en augmentant la variance du bruit vers l'infini.

- Pour réduire la variance du signal, la consommation de puissance a besoin d'être exactement égale pour toutes les opérations exécutées et données manipulées. En pratique, cela se traduira par de petites valeurs de variances pour le signal.

-
- Pour augmenter la variance du bruit, l'amplitude du bruit a besoin d'être augmentée de façon significative laissant croire à l'attaquant l'existence de commutations sur les cellules du device. En pratique, cela se traduira par de grandes valeurs de variances pour le bruit.

Contre-mesures *Hiding*

Comme précisé ci-avant, le principe des contre-mesures de type Hiding est de rendre la consommation de puissance du device cryptographique indépendante des opérations exécutées et des données manipulées. De cette façon, deux approches sont possibles :

1. Faire en sorte que la consommation de puissance du device cryptographique soit **aléatoire**. Cela signifie qu'à chaque coup de clock, une certaine quantité aléatoire de puissance est consommée. Le but est donc de **modifier l'instant du leakage** ou bien de **modifier l'intensité du leakage** dans la trace de puissance.
 - *La modification de l'instant du leakage peut être réalisée par un désalignement des traces.* Cela va compliquer la tâche de l'attaquant. En effet, celui-ci doit, dans un premier temps, procéder à l'alignement de ses traces pour pouvoir les analyser correctement. Si celles-ci ne sont pas alignées, il ne pourra rien en tirer de concret. Ce désalignement des traces peut s'obtenir de différentes façons : utiliser des horloges de fréquences différentes, utiliser des interruptions aléatoires lors de l'exécution du programme, changer l'ordre des instructions, etc.
 - *La modification de l'intensité du leakage peut être réalisée par une modification du rapport signal à bruit.* Nous avons vu que pour modifier le SNR, il suffisait d'augmenter le bruit ou de diminuer le signal. Dans ce cas-ci, nous allons augmenter la variance du bruit. En effet, en ajoutant du bruit de façon indépendante à l'exécution de l'algorithme, on va diminuer le SNR, ce qui va avoir pour conséquence de diminuer le leakage d'une opération : L'attaquant aura dès lors plus de difficultés à retrouver la clé secrète. On peut, pour ce faire, utiliser un filtre afin de ne laisser passer que certaines composantes de puissance ou encore utiliser ce qu'on appelle des "*noise engines*", systèmes fonctionnant en parallèle du device cryptographique et ayant pour but de générer du bruit.
2. Faire en sorte que la consommation de puissance du device cryptographique soit **identique**. Cela signifie qu'à chaque coup de clock, une quantité égale de puissance est consommée pour toutes les opérations exécutées et pour toutes les données manipulées. Le but est donc de **modifier l'intensité du leakage**. Autrement dit, on souhaite uniformiser l'amplitude de la trace. *Pour ce faire, on va à nouveau modifier le rapport signal à bruit.* L'idéal étant d'avoir un SNR proche de 0, si on n'augmente pas la variance du bruit, la deuxième approche consiste à diminuer la variance du signal. Pour ce faire, on va modifier de façon physique les cellules CMOS. En effet, le but est de faire en sorte que chaque cellule logique consomme la même consommation de puissance pour toute opération demandée.

La figure 11.12 ci-dessous présente les possibilités de contre-mesures mises en oeuvre selon le type d'approche suivi tandis que l'annexe C.1 reprend l'ensemble des contre-mesures de type hiding.

<i>Consommation de puissance équivalente</i>	<i>Consommation de puissance aléatoire</i>
<i>Intensité du leakage</i>	<i>Instant du leakage</i>
	<i>Intensité du leakage</i>

Figure 11.12 : Les contre-mesures de type hiding sont utilisées pour rendre aléatoire ou égale la consommation de puissance du device cryptographique.

Sur base de l'annexe C.1, nous pouvons expliquer le principe de certains exemples de contre-mesures :

- Concernant la manipulation d'horloge, nous pouvons :
 - Ignorer certains coups de clock : ce qui aura pour effet de retarder l'exécution du prochain cycle, ce qui engendrera donc un désalignement des traces.
 - Changer aléatoirement la fréquence de clock : À l'aide d'un oscillateur et de nombres aléatoires, la fréquence d'horloge est changée à intervalles fréquents, provoquant des vitesses d'exécution différentes et par conséquent un désalignement des traces.
- Concernant les interruptions : on va générer aléatoirement certaines interruptions lors de l'exécution de l'algorithme, ce qui va *casser* (couper) les traces provoquant ainsi un désalignement.
- Concernant la manipulation d'instructions :

-
- Le mélange des instructions : on va mélanger l'ordre des opérations (seules les opérations qui peuvent être déplacées ou interverties). Ainsi, pour AES par exemple, il est possible de réaliser l'opération de subBytes sur les 16 bytes de la matrice, dans n'importe quel ordre, sans nuire au déroulement de l'algorithme.
 - Les instructions inutiles : en insérant des opérations inutiles, on va injecter des quantités d'informations inutiles dans chaque trace. Cela provoquera un désalignement des traces.
 - Le choix des instructions : toutes les instructions ne "fuitent" pas la même quantité d'information sur les opérandes. Il pourrait être bon de choisir les instructions qui révèlent le moins d'informations utiles à un attaquant ou de remplacer certaines instructions par d'autres instructions équivalentes pour que la fuite d'information ne soit pas toujours la même.
 - Concernant la diminution du SNR :
 - On peut diminuer le SNR en augmentant la variance du bruit à l'aide de *noise engines*. Il s'agit de composants hardware qui travaillent en parallèle à l'exécution cryptographique. Autrement dit, ces composants hardware vont générer du bruit, ce qui va avoir pour conséquence de diminuer le SNR. En effet, la consommation mesurée sera la somme de la consommation de l'exécution de l'algorithme cryptographique et des composants hardware.
 - On peut diminuer le SNR en diminuant la variance du signal. En pratique, il existe deux possibilités pour diminuer la variance du signal :
 - Une première approche pourrait concerner la cellule logique CMOS en elle-même. On sait (section 3.1.1) que la consommation de puissance totale d'un device cryptographique est la somme des puissances consommées par chaque cellule. Ainsi, si chaque cellule consomme une puissance constante, la puissance totale est constante. Il faut donc construire des cellules qui consomment des quantités de puissance constantes.
 - Une seconde approche plus complexe concerne le filtrage. Il s'agit de filtrer la puissance consommée par le device cryptographique. Le but est alors de supprimer, via ce filtre, toutes les composantes de la trace de puissance qui dépendent des données manipulées et des opérations exécutées.

Contre-mesures *Masking*

Le principe des contre-mesures de type masking est de générer des valeurs intermédiaires aléatoires. Autrement dit, lors de l'exécution d'un algorithme, différentes opérations sont exécutées conduisant à différentes valeurs intermédiaires calculées. Si aucune protection n'est mise en place, la consommation de puissance du device cryptographique dépendra des données intermédiaires qui sont manipulées par le device cryptographique (par l'algorithme précisément). L'attaquant peut alors potentiellement retrouver la clé secrète en analysant la consommation de puissance du device cryptographique. Par contre, si chaque valeur intermédiaire est dissimulée sous une nouvelle valeur intermédiaire aléatoire dite **masquée**, alors l'attaquant pourra toujours analyser la consommation de puissance, les traces qu'il obtiendra fourniront des informations faussées (par le masque). En d'autres mots, ce type de contre-mesure accepte que la consommation de puissance du device cryptographique dépend des données manipulées et des opérations exécutées. Cependant, on va modifier les valeurs intermédiaires de façon aléatoire pour que les traces mesurées à l'oscilloscope n'aient plus de sens. Un avantage de cette approche est qu'elle peut être implémentée au niveau de l'algorithme, c'est-à-dire sans changer les caractéristiques de consommation de puissance du device cryptographique (ce qui est plus complexe) comme c'est le cas pour les contre-mesures de type *Hiding*.

Définition du masque : Une valeur intermédiaire masquée v_m est une valeur intermédiaire v cachée par une valeur aléatoire m . On obtient donc la relation suivante (11.5) :

$$v_m = v * m \quad (11.5)$$

Ne connaissant pas la valeur aléatoire m , l'attaquant ne peut pas retrouver la valeur intermédiaire v . Ainsi, la consommation de puissance du device cryptographique dépend des valeurs intermédiaires masquées v_m qui ne fournissent aucune information sur les valeurs intermédiaires v . Autrement dit, un masque cache les valeurs intermédiaires et il n'est donc plus possible de retrouver la clé de chiffrement. Bien évidemment, les masques ont besoin d'être supprimés à la fin des opérations algorithmiques afin d'obtenir *in fine* le vrai message chiffré.

L'opération $*$ (dans (11.5)) représente le type d'opération réalisé pour appliquer le masque (sur les valeurs intermédiaires). Il peut s'agir d'une fonction booléenne ou d'une fonction arithmétique. Ainsi, on définit :

1. ***Un masque booléen*** : l'opération $*$ est alors remplacée par la fonction booléenne XOR (\oplus). Dans ce cas, le masque devient : $v_m = v \oplus m$.
2. ***Un masque arithmétique*** : l'opération $*$ est alors remplacée par une addition modulaire ($+$) ou par une multiplication modulaire (\times). Dans ce cas, le masque devient : $v_m = v + m \text{ (mod } n\text{)}$ ou $v_m = v \times m \text{ (mod } n\text{)}$ où *modulo n* est défini en fonction de l'algorithme de chiffrement.

Contre-mesures *Faking*

Après diverses recherches sur *Internet*, j'ai constaté qu'un nouveau type de contre-mesure a vu le jour : les contre-mesures de type ***Faking***. Les articles définissant ce type de contre-mesure ont été rédigés fin 2016 au plus tôt, il s'agit donc d'une contre-mesure assez récente. Cette contre-mesure s'appuie sur une partie du concept de contre-mesure par *Masking*.

L'objectif de la contre-mesure est le suivant : Dès le départ, on va venir masquer la vraie valeur de clé (Key_{Real}) en réalisant un XOR avec un masque (Key_{Mask}) produisant ainsi une fausse clé (Key_{Fake}). On a donc la relation (11.6) :

$$Key_{Fake} = Key_{Real} \oplus Key_{Mask} \quad (11.6)$$

À partir de cette fausse clé, on va exécuter l'algorithme AES de façon normale (tel que décrit à la section ??) : C'est-à-dire qu'on va tout d'abord appliquer l'opération *KeySchedule* générant ainsi d'autres fausses clés et ensuite on va exécuter les quatre opérations élémentaires (*AddRoundKey*, *SubBytes*, *ShiftRows*, *MixColumns*) afin de chiffrer les données. On constate ainsi que c'est la fausse clé qui est utilisée pour chiffrer les données et non la vraie clé. De cette manière, si un attaquant opère une attaque de type CPA (ou autre), en supposant qu'il ne connaisse pas la valeur du masque (Key_{Mask}) appliqué sur la vraie clé, il sera capable de retrouver une valeur de clé cependant, il s'agira de la fausse clé générée au départ. En conclusion, l'attaquant ne pourra, à partir de la fausse clé, déchiffrer les données.

Dans les contre-mesures de type *Masking*, lorsqu'on applique un masque sur les données intermédiaires, il faut ensuite le retirer de façon à obtenir *in fine* le *vrai* texte chiffré. Pour le cas des contre-mesures de type *Faking*, le principe est identique. Pour ce faire, on va utiliser un co-processeur dont le rôle est d'annuler l'emploi du masque afin d'obtenir les vraies données intermédiaires à chaque fin de round et donc *in fine* les vraies données chiffrées. L'annexe C.2 présente le principe de fonctionnement d'une contre-mesure de type Faking. On y retrouve les opérations élémentaires de l'algorithme AES (*AddRoundKey*, *SubBytes*, *ShiftRows*, *MixColumns*) ainsi que deux opérations spéciales exécutées par le co-processeur (*SubBytesTrans*, *MixCol*). *SubBytesTrans* est définie de façon à annuler l'utilisation de la fausse clé sur les données alors que *MixCol* réalise la même opération que *MixColumns*.

À l'aide de MATLAB, j'ai réalisé une simulation dans laquelle le device cryptographique est protégé par une contre-mesure de type *Faking*. Ensuite, j'ai simulé une attaque de type CPA sur ce device. Le résultat, observable à la figure 11.13, montre clairement qu'un attaquant est capable de retrouver la clé de chiffrement. Néanmoins, il s'agit de la fausse clé ! La vrai clé quant à elle ne laisse fuiter aucune information !

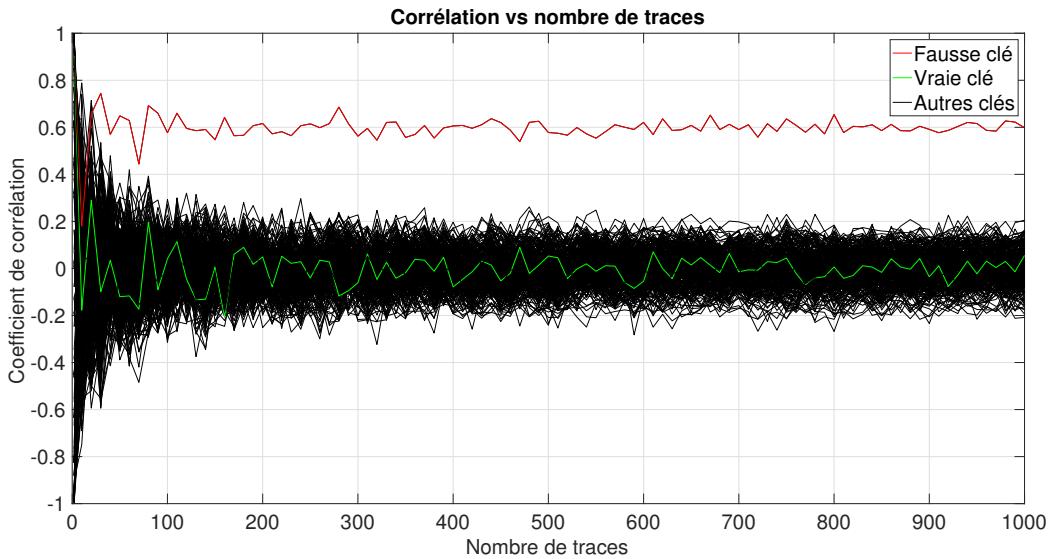


Figure 11.13 : On voit qu'au plus le nombre de traces augmente au plus une clé se distingue des 255 autres. Il s'agit de la fausse clé (en rouge). La vraie clé (en vert) ne laisse fuite aucune information.

11.1 Conclusion

Nous vivons dans une société de consommation où les technologies évoluent sans cesse. Il y a 40 ans, la gestion des signaux relevait de la science-fiction. Il y a 25 ans, les GSM constituaient une innovation majeure. Aujourd’hui, tout le monde se demande comment il était possible de vivre à cette époque sans toutes ces nouvelles technologies ! Le sujet qu'il m'a été proposé d'étudier durant ces 6 semaines de stage en immersion fut, à première vue, aussi complexe qu'inconnu. En effet, ce sujet, les attaques par canaux cachés, est encore assez méconnu à ce jour dans le domaine de l'ingénierie. Tenter d'expliquer à quiconque qu'il est possible de subtiliser des données sensibles précautionneusement chiffrées par divers algorithmes, à partir de moyens simples (oscilloscope, ordinateurs, ...) et de calculs élémentaires, semble au premier abord, difficilement concevable.

Depuis ces découvertes à la fin des années 90, la sécurité matérielle a pris une tournure particulière pour les grandes industries telles que Thales. C'est désormais sur un nouvel axe de recherche, s'écartant des sentiers traditionnels, que se porte la problématique de protection de données sensibles. Imaginez qu'un algorithme utilisé pour chiffrer des données bancaires puisse être cassé par simple application d'une attaque par canaux cachés ? Il n'est donc pas anormal de voir certaines industries développer leurs propres contre-mesures pour s'affranchir contre ce type d'attaque. Certes, avant de développer des contre-mesures, il est nécessaire, dans un premier temps, de bien comprendre les fondements des attaques par canaux cachés. Certes, la compréhension des notions théoriques et leurs mises en pratique est une tâche ardue qui nécessite la compréhension et l'intégration de concepts très spécifiques. Néanmoins, je suis convaincu que ce type d'attaque va, au fil des années, devenir une piste d'enseignement sérieuse et nécessaire aux futurs ingénieurs en électronique.

À titre personnel, je tire un bilan très positif de ce stage d'immersion. Il fut très enrichissant. Ces 6 semaines de stage sont passées à la vitesse de l'éclair : plongeant à certains moments dans les articles scientifiques et livres de références rédigés sur le sujet ; à d'autres dans les simulations MATLAB afin de mieux assimiler les concepts ; ou encore à discuter avec les étudiants stagiaires afin de partager notre expérience de stage. 6 semaines, c'est assez court mais déjà suffisant pour une première introduction sur les attaques par canaux cachés et les contre-mesures qu'il est possible d'implémenter. L'ensemble des objectifs fixés avant le début de stage (énoncés à la section ??) a pu être traité et atteint. Étant désormais convaincu de la puissance de ce type d'attaque, développer des contre-mesures me paraît constituer un challenge intéressant et très utile dans le cadre de la protection des données sensibles. Je vais donc désormais pouvoir m'attaquer à la phase suivante, consécutive à ce stage, mon Travail de Fin d'Étude. Ce TFE consistera à

développer une contre-mesure contre les attaques par canaux cachés.

Crédits

- Figure ?? provenant du site internet : <http://monipag.com/victoria-petitier/wp-content/uploads/sites/1363/Thales-Group-1.png>
- Figure 3.1 provenant du site internet : <https://hal.archives-ouvertes.fr/hal-00753215/document>
- Annexe A.4 provenant du site internet de J.M. Dutertre "*Synthèse AES 128*" : https://www.emse.fr/~dutertre/documents/synth_AES128.pdf
- Annexe C.2 provenant du site internet UPCCommons "*Faking Countermeasure Against Side-Channel Attacks*" : https://upcommons.upc.edu/bitstream/handle/2117/112973/Advances_in_Microelectronics_V_1_chapter19.pdf?sequence=1

Bibliographie

- [1] Thomas Popp Stefan Mangard, Elisabeth Oswald. *Power Analysis Attacks*. Springer, 2007.
- [2] Sri Parameswaran Jude Ambrose, Alexandar Ignjatovic. *Power Analysis Side Channel Attacks*. VDM Verlag Dr. Müller, 2010.
- [3] Eric Peeters. *Advanced DPA Theory and Practise*. Springer, Septembre 2012.
- [4] J.M. Dutertre. *Synthèse AES 128*. https://www.emse.fr/~dutertre/documents/synth_AES128.pdf, 2011.
- [5] Lilian BOSSUET. *Approche didactique pour l'enseignement de l'attaque DPA ciblant l'algorithme de chiffrement AES*. <https://hal.archives-ouvertes.fr/hal-00753215/document>, 25 Octobre 2012.
- [6] Stephane Fernandes Medeiros. *Attaques par canaux auxiliaires : nouvelles attaques, contre-mesures et mises en oeuvre*. <https://dipot.ulb.ac.be/dspace/bitstream/.../12a25345-b54f-4169-9c32-e7d2cce5af65.txt>, 2017.
- [7] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. Cryptology ePrint Archive, Report 2015/536, 2015. <https://eprint.iacr.org/2015/536>.
- [8] Francois Durvaux Francois-Xavier Standaert A. Adam Ding, Liwei Zhang and Yunsi Fei. Towards sound and optimal leakage detection procedure. In the proceedings of CARDIS 2017, Lecture Notes in Computer Science, November 2017. <https://perso.uclouvain.be/fstandae/PUBLIS/196.pdf>.
- [9] MICHAL VARCHOLA MAREK REPKA, JOZEF TOMECEK. Correlation hamming distance power analysis of 16-bit integer multiplier in fpga. <http://www.wseas.us/e-library/conferences/2014/Istanbul/TELEDU/TELEDU-06.pdf>, 2014.
- [10] Mariano López-García Rubén Lumbiarres-López and Enrique Cantó-Navarro. Faking countermeasure against side-channel attacks. https://upcommons.upc.edu/bitstream/handle/2117/112973/Advances_in_Microelectronics_V_1_chapter19.pdf;jsessionid=62B833C8C8C140B9EE46F0279D3E0B61?sequence=1, Décembre 2017.
- [11] Douglas Carson Owen Lo, William J. Buchanan. Power analysis attacks on the aes-128 s-box using differential power analysis (dpa) and correlation power analysis (cpa). <https://www.tandfonline.com/doi/pdf/10.1080/23742917.2016.1231523>, Septembre 2016.
- [12] Mohsen Machhou Rached Tourki Hassen Mestiri, Noura Benhadjyoussef. A comparative study of power consumption models for cpa attack. <http://www.mecs-press.net/ijcnis/ijcnis-v5-n3/IJCNIS-V5-N3-3.pdf>, 2013.
- [13] Correlation power analysis. https://wiki.newae.com/Correlation_Power_Analysis.
- [14] Abdelaziz Elaabid. *Attaques par canaux cachés : expérimentations avancées sur les attaques template*. <https://tel.archives-ouvertes.fr/tel-00937136/document>, 2014.
- [15] 6.1.2 *Distance de Hamming et poids d'un mot de code*. https://icwww.epfl.ch/~chappeli/it/courseFR/I2subsec_Hdist.php.
- [16] Jean-Philippe Muller. *Le bruit dans les systèmes électroniques*. <http://www.ta-formation.com/acrobat-cours/bruit.pdf>, Juillet 2002.

- [17] Renaud Dumont. *Cryptographie et Sécurité informatique INFO0045-2*. <http://www.montefiore.ulg.ac.be/~dumont/pdf/crypto09-10.pdf>, 2010.
- [18] Thales Group. *Historique*. <https://www.thalesgroup.com/fr/global/groupe/historique>, 2018.
- [19] Thales. *Wikipedia*. <http://fr.wikipedia.org/w/index.php?title=Thales&action=history>, Octobre 2018.

Annexe **A**

Liste des annexes du chapitre 2

- Annexe **A.1** : Chiffre de César.
- Annexe **A.2** : Code algorithme AES-128 - Ordre d'exécution des opérations.
- Annexe **A.3** : Code algorithme AES-256 - Ordre d'exécution des opérations.
- Annexe **A.4** : Sbox - Table de substitution.
- Annexe **A.5** : Opération *KeySchedule*.

A.1 Chiffre de César

Texte clair	Texte chiffré
A	D
B	E
C	F
D	G
E	H
F	I
G	J
H	K
I	L
J	M
K	N
L	O
M	P
N	Q
O	R
P	S
Q	T
R	U
S	V
T	W
U	X
V	Y
W	Z
X	A
Y	B
Z	C

Table A.1 : Chiffre de César.

Exemple : le message clair "*chiffrement*" devient le message chiffré "*fkluiuhphqw*". En effet, le détail du procédé de chiffrement est donné à la table A.2.

Message clair	C	H	I	F	F	R	E	M	E	N	T
Message chiffré	F	K	L	I	I	U	H	P	H	Q	W

Table A.2 : Exemple de chiffre de César.

A.2 Code algorithme AES-128 - Ordre d'exécution des opérations.

```
Require : STATE, Key
Ensure : STATE

KeySchedule (Key)
AddRoundKey (STATE, ExpandedKey [ 0 ] )
for i=1 < 10 do
    SubBytes (STATE)
    ShiftRows (STATE)
    MixColumns (STATE)
    AddRoundKey (STATE, ExpandedKey [ i ] )
    i = i + 1
end for
SubBytes (STATE)
ShiftRows (STATE)
AddRoundKey (STATE, ExpandedKey [ 10 ] )
```

A.3 Code algorithme AES-256 - Ordre d'exécution des opérations.

```
Require : STATE, Key
Ensure : STATE

KeySchedule (Key)
AddRoundKey(STATE, ExpandedKey [ 0 ] )
for i=1 < 14 do
    SubBytes (STATE)
    ShiftRows (STATE)
    MixColumns (STATE)
    AddRoundKey (STATE, ExpandedKey [ i ] )
    i = i + 1
end for
SubBytes (STATE)
ShiftRows (STATE)
AddRoundKey (STATE, ExpandedKey [ 10 ] )
```

A.4 Sbox - Table de substitution

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

Table A.3 : La Sbox représente une table de substitution utilisée pour l'opération non-linéaire SubBytes de l'algorithme AES. À chaque donnée hexadécimale d'entrée correspond une donnée hexadécimale de sortie.

A.5 Opération *KeySchedule*

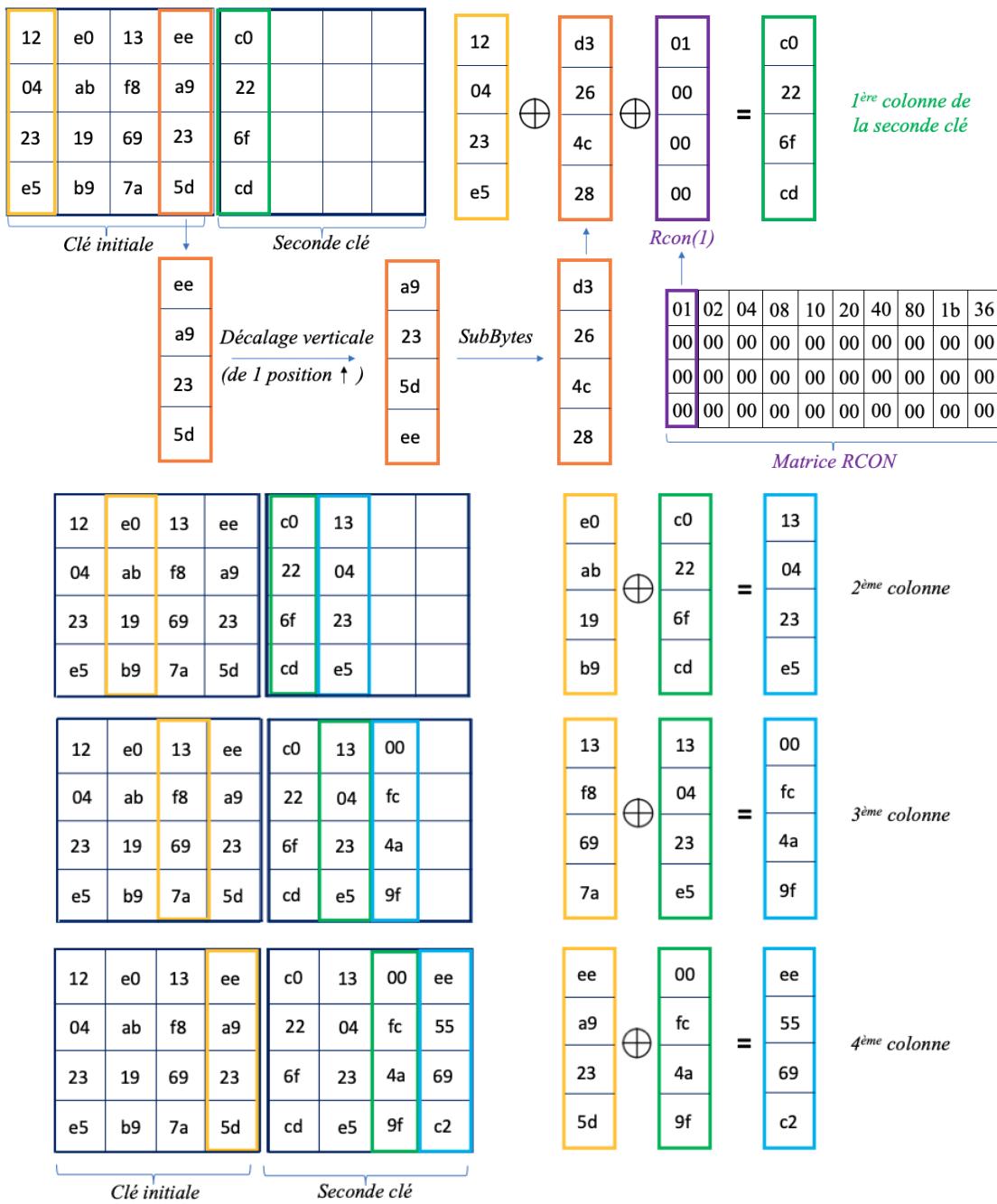


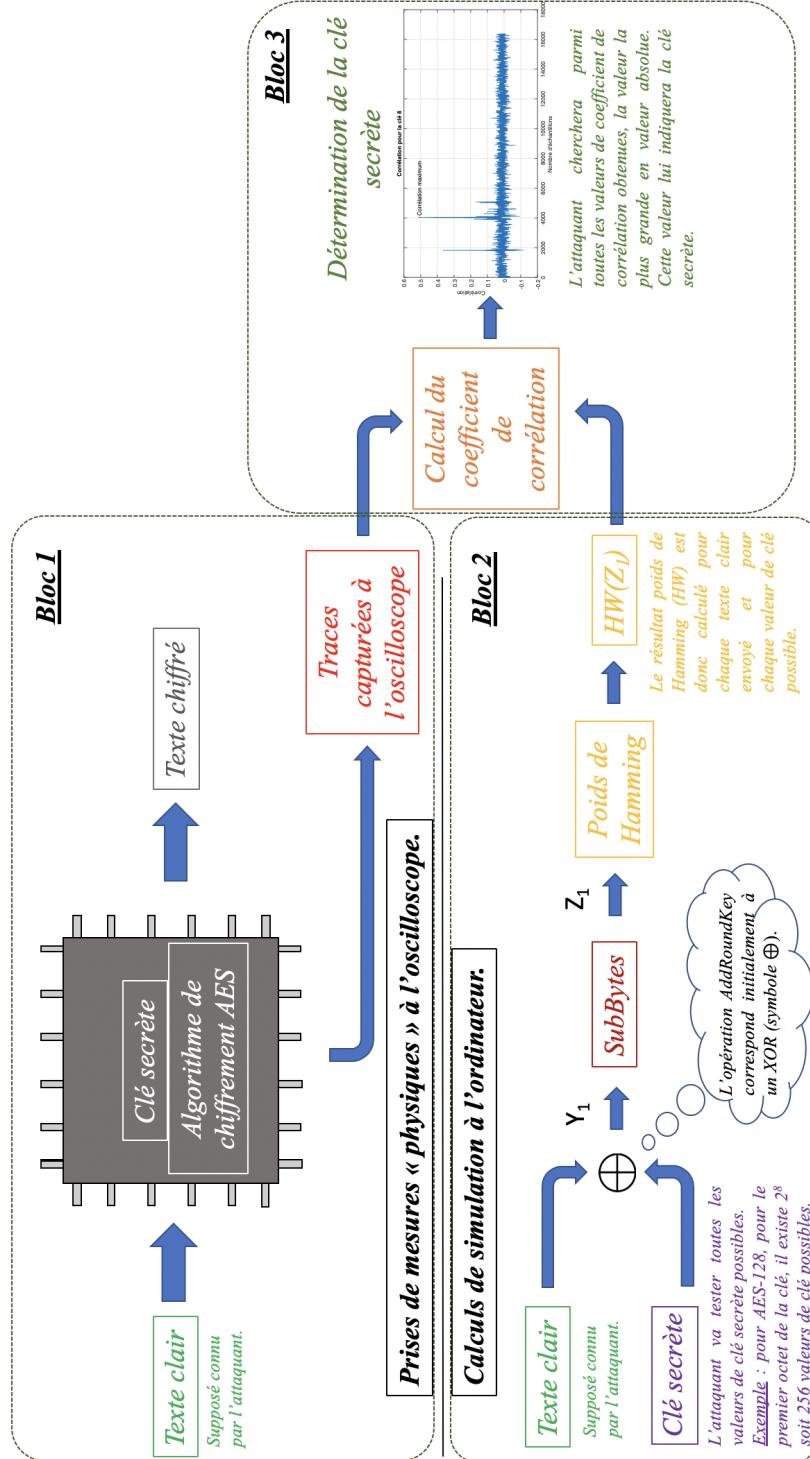
Figure A.1 : Principe de fonctionnement de l'opération *KeySchedule*.

Annexe **B**

Liste des annexes du chapitre 4

- Annexe **B.1** : Principe de fonctionnement d'une attaque CPA.
- Annexe **B.2** : Représentation des traces de puissance prises à l'oscilloscope.
- Annexe **B.3** : Traces.
- Annexe **B.4** : Corrélation pour les clés 1 à 4.
- Annexe **B.5** : Corrélations 3D.
- Annexe **B.6** : Corrélation pour la clé 8.

B.1 Principe de fonctionnement d'une attaque CPA.



B.2 Représentation des traces de puissance prises à l'oscilloscope.

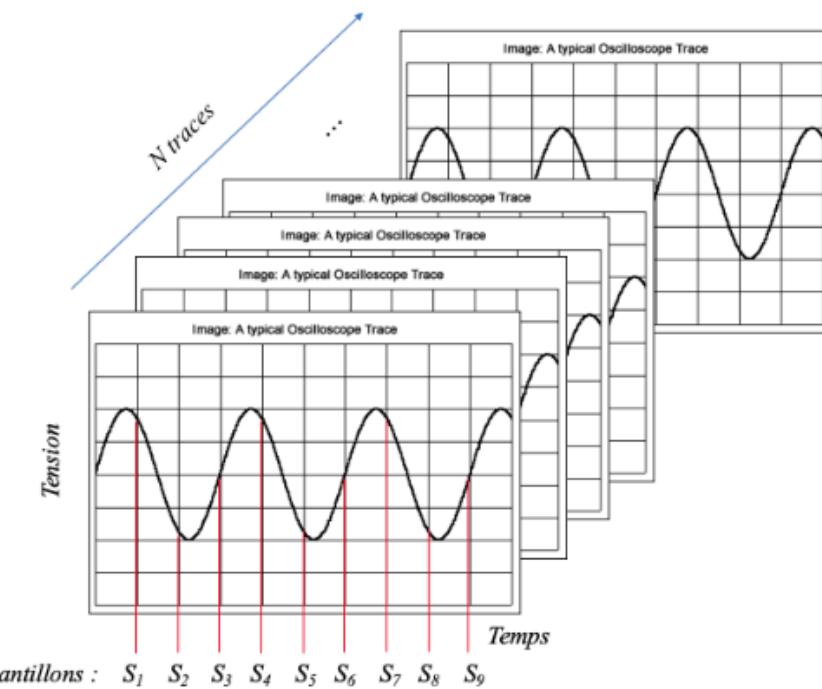
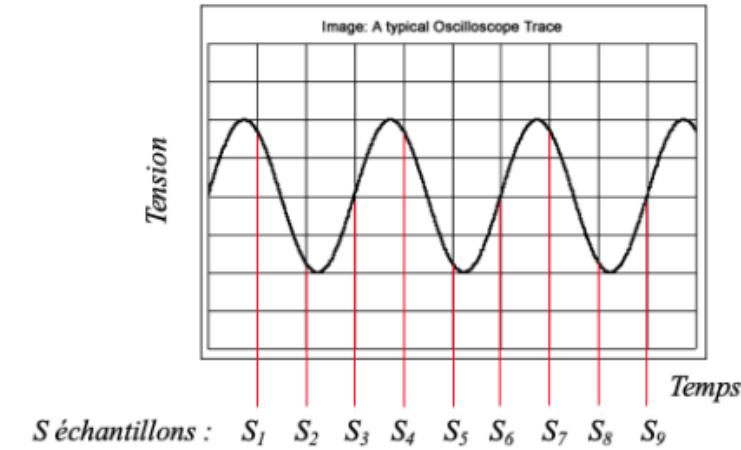
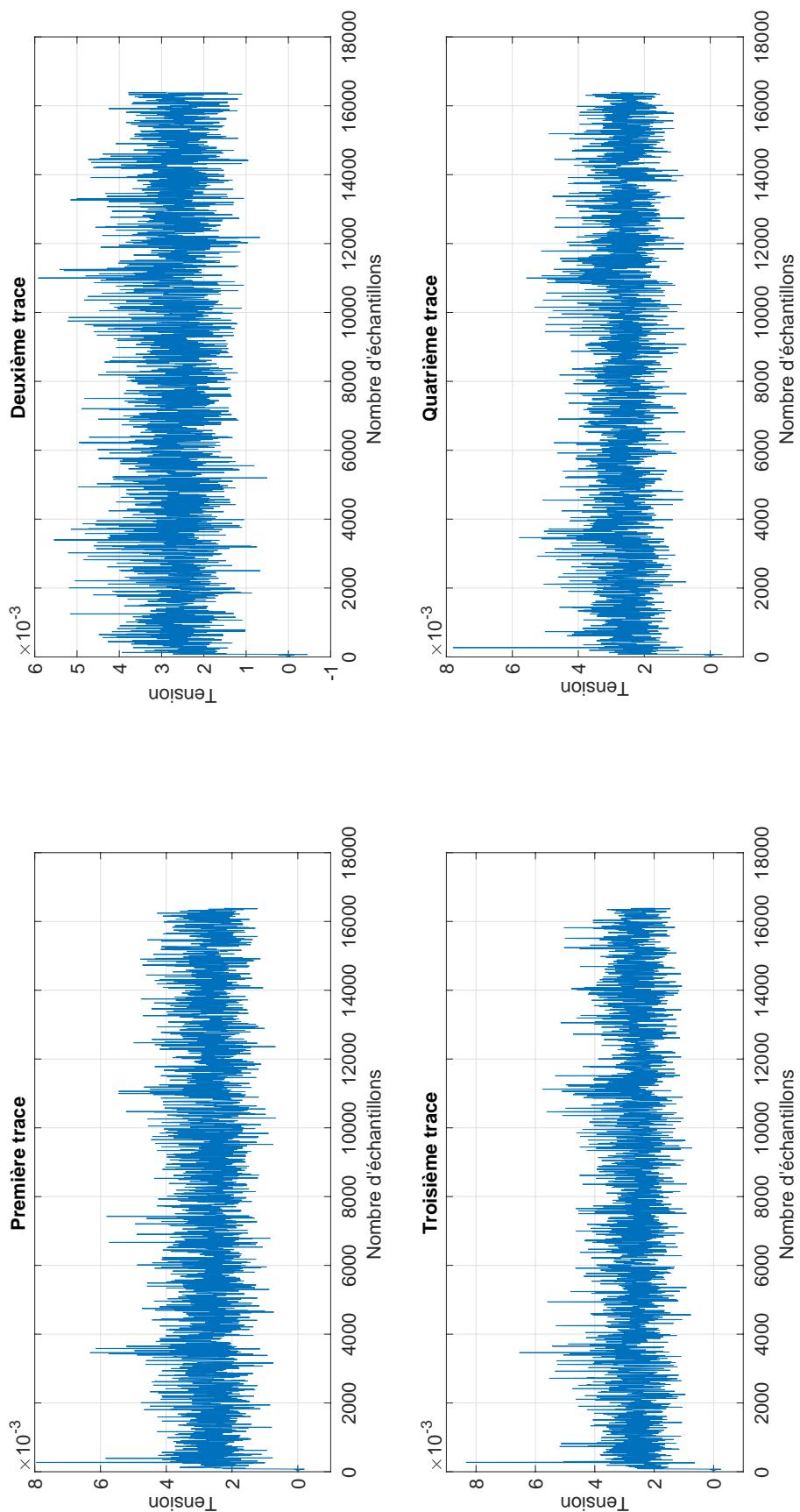
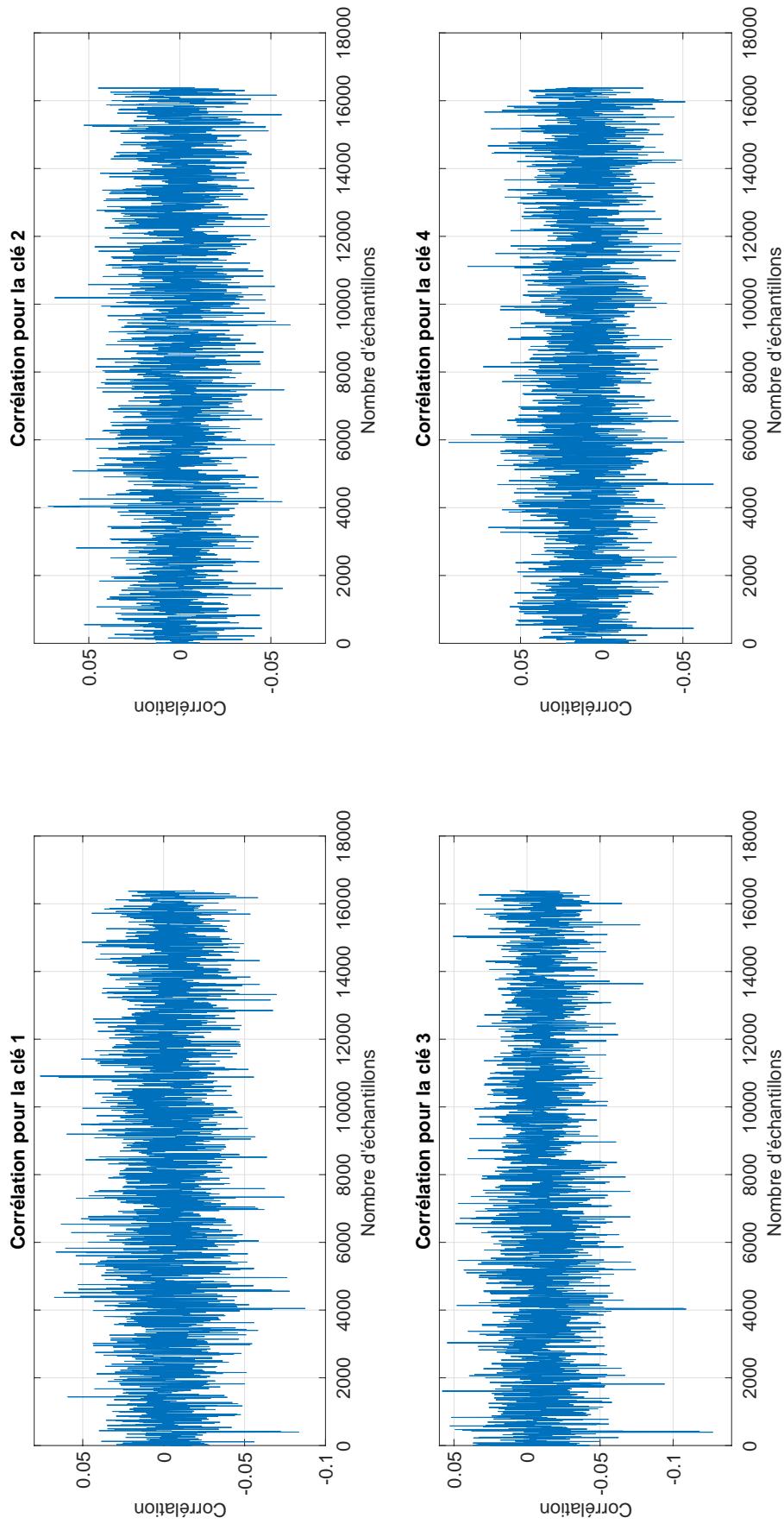


Figure B.1 : La figure du haut représente une trace de puissance (i.e 1 plaintext utilisé) dans laquelle on retrouve 9 échantillons. La figure du bas représente N traces de puissance dans lesquelles on retrouve également 9 échantillons. En pratique, le nombre d'échantillons dans une trace de puissance est beaucoup plus élevé (de l'ordre de 10000) et sera noté par la lettre S (pour *Sampling*).

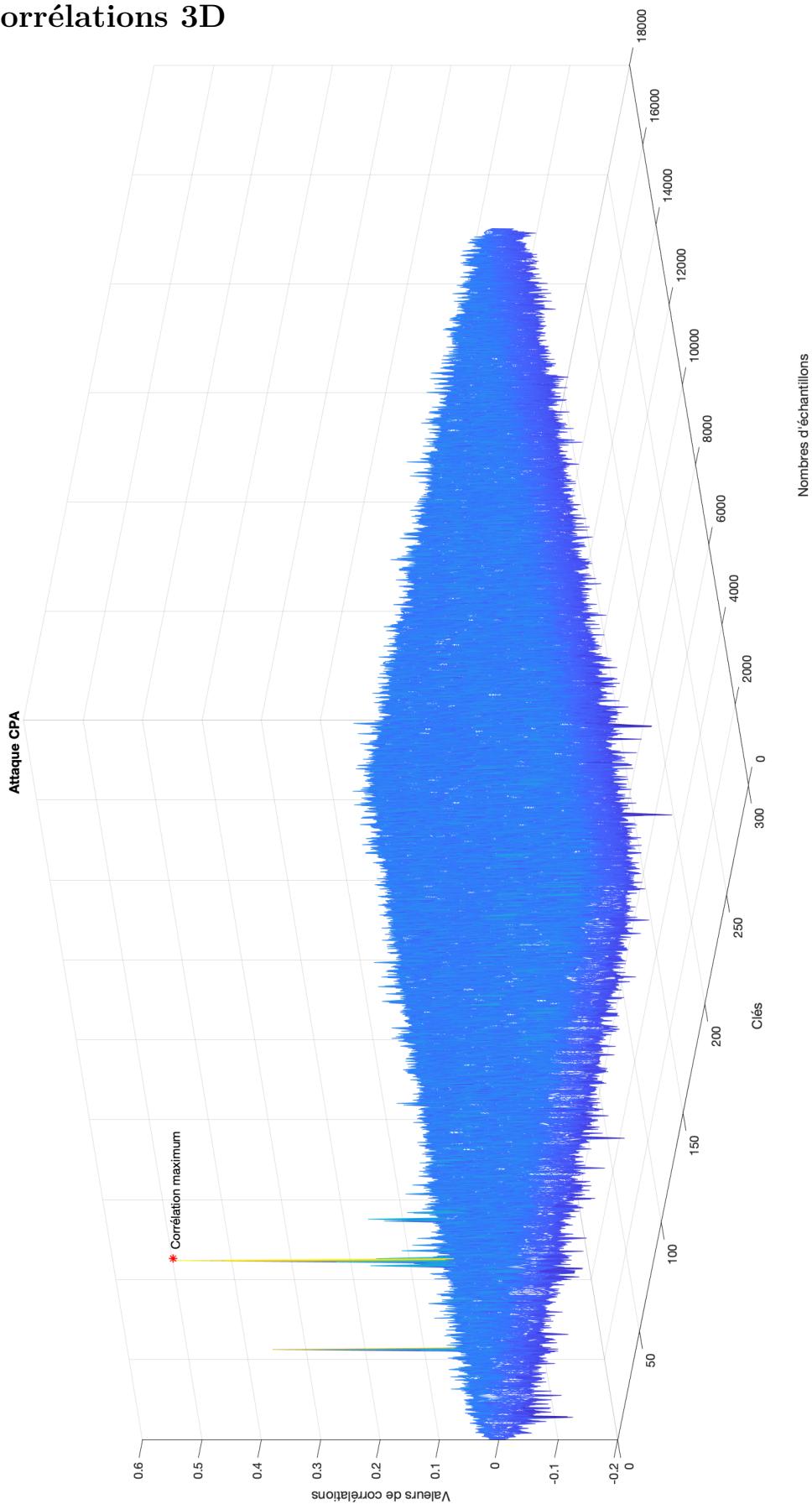
B.3 Traces



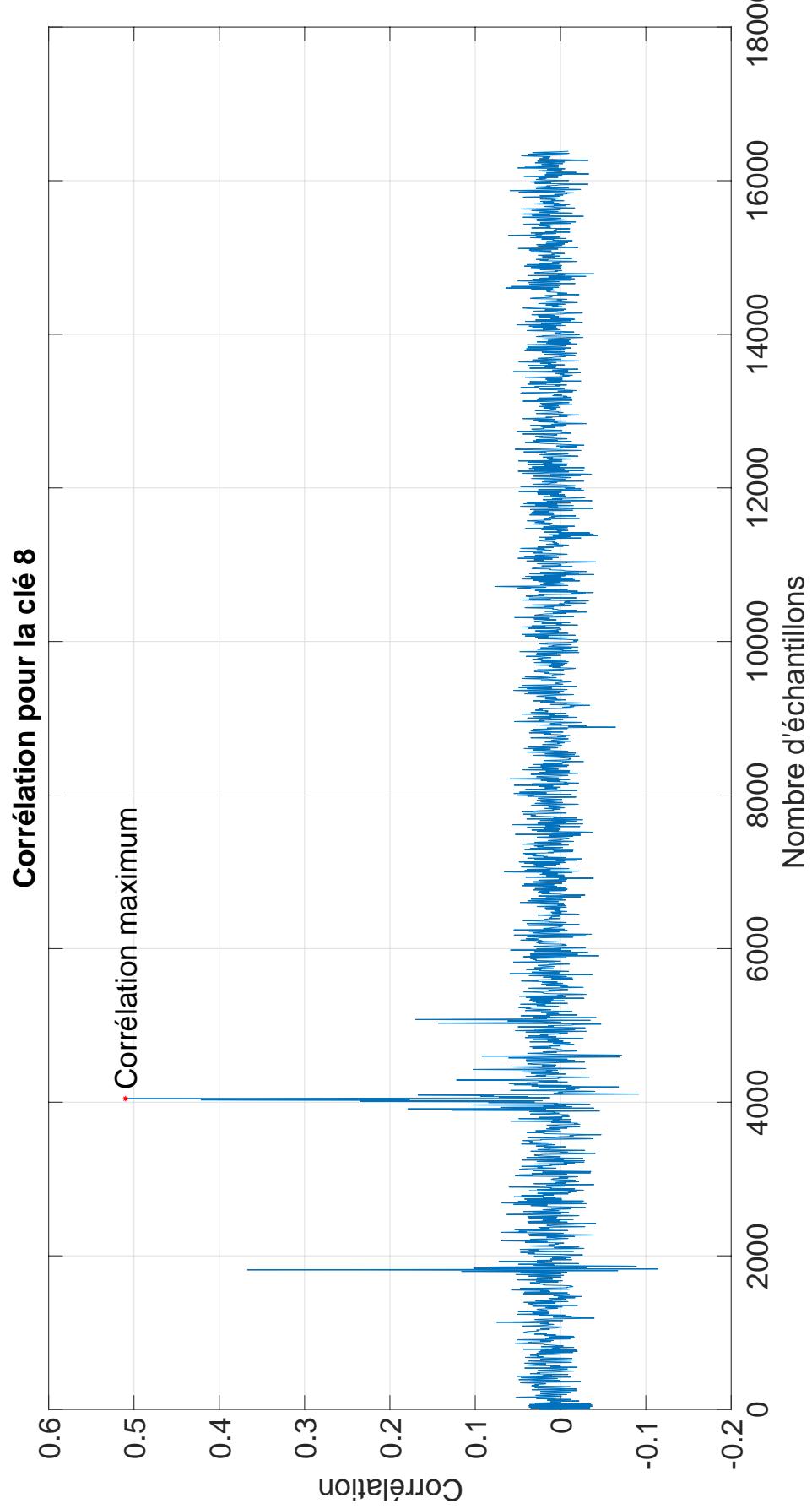
B.4 Corrélation pour les clés 1 à 4



B.5 Corrélations 3D



B.6 Corrélation pour la clé 8



Annexe **C**

Liste des annexes du chapitre 5

- Annexe C.1 : Contre-mesures *Hiding*.
- Annexe C.2 : Contre-mesure *Faking*.

C.1 Contre-mesures *Hiding*

C.2 Contre-mesure *Faking*

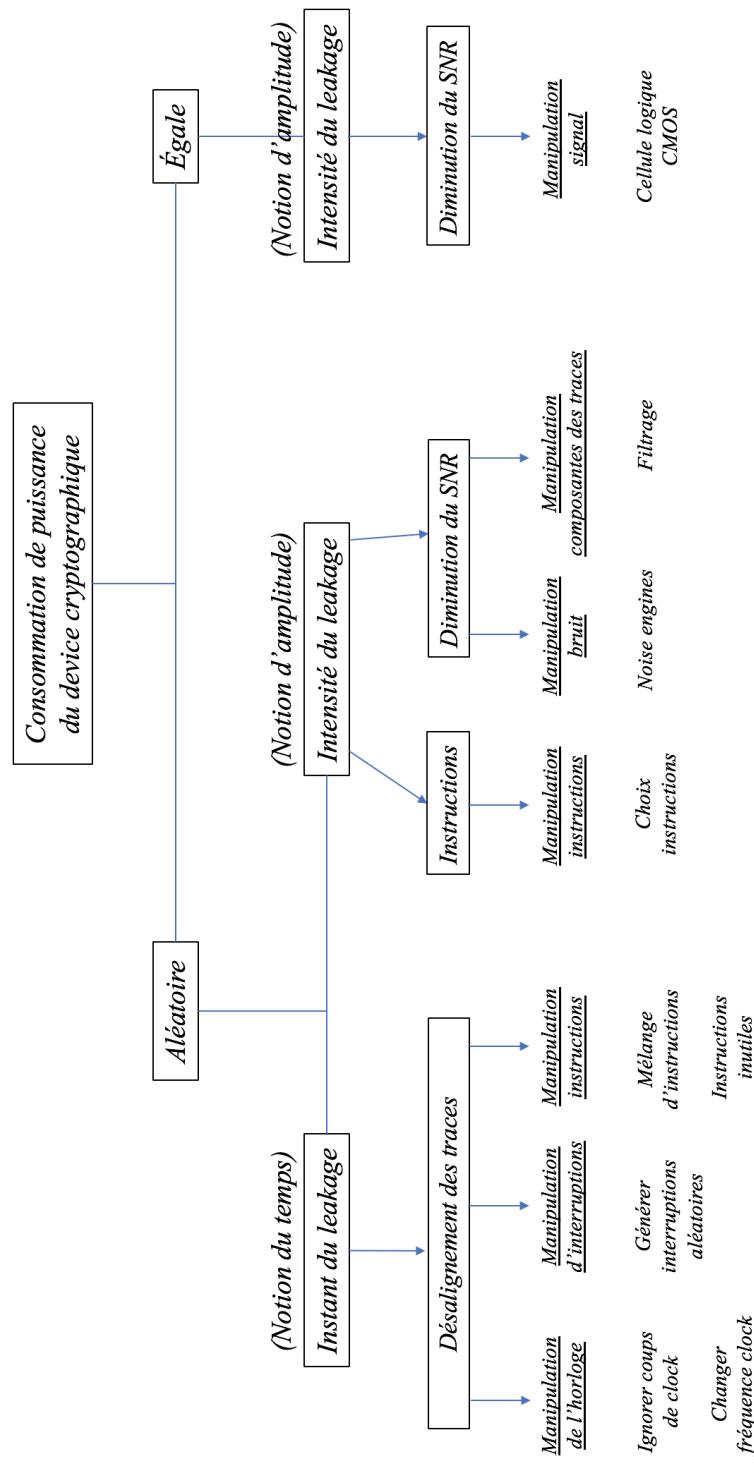


Figure C.1 : Ce schéma présente les différentes sortes de contre-mesures de type hiding.

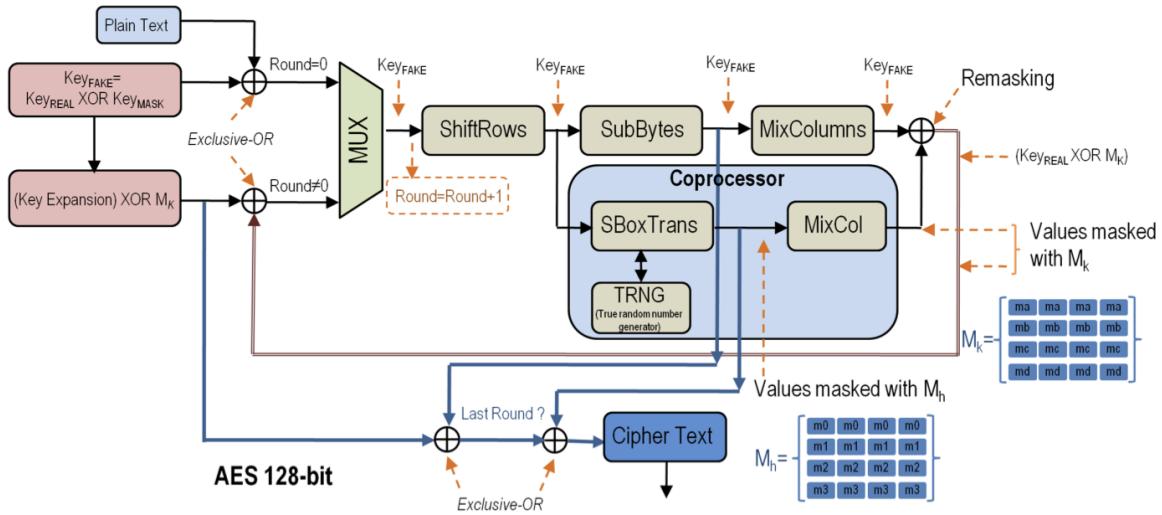


Figure C.2 : Ce schéma-bloc présente le principe de fonctionnement de la contre-mesure de type Faking. On remarque que la clé manipulée par l'algorithme AES (pour chaque round exécuté) est une fausse clé (Key_{Fake}), obtenue en dissimulant la vraie clé (K_{Real}) sous un masque (Key_{Mask}).

Annexe **D**

Autres annexes

- Annexe **D.1** : Lettre de motivation.
- Annexe **D.2** : Certificat de stage.

D.1 Lettre de motivation

Anizet Thomas
 Rue Henri Loriaux, n°38
 6210 Frasnes-Lez-Gosselies
 Belgique
 +32 476 62 69 90
thomas.anizet@hotmail.com

Thales Communication
 Rue des Frères Taymans, 28
 1480 Tubize

Les Bons Villers, le 27 Avril 2018

OBJET : LETTRE DE MOTIVATION
 DOSSIER DE STAGE D'INSERTION PROFESSIONNELLE
 DOSSIER DE TRAVAIL DE FIN D'ÉTUDE (TFE)

Monsieur Dardenne,

Candidat "Officier de carrière" à l'École Royale Militaire, je suis actuellement ma formation académique à l'École Centrale des Arts et Métiers (ECAM) en option électronique.

Durant notre 2^{ème} année de Master, les étudiants doivent réaliser dans un premier temps un stage d'insertion professionnelle en entreprise d'une durée de 30 jours (Septembre – Novembre 2018). Ce stage consiste en l'étude d'une organisation entrepreneuriale avec son management, son contexte social, son insertion économique, ses aspects techniques et ses produits. Il se veut également participatif en nous encourageant à se consacrer de manière autonome et active à un projet industriel.

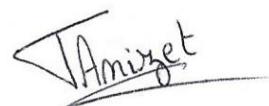
Par la suite, durant le second semestre (Février - Juin 2019), il est prévu que les élèves consacrent 2 jours de leur semaine à la réalisation de leur « Travail de Fin d'Étude » (TFE). Il est généralement convenu qu'une entreprise propose à un étudiant un sujet de TFE basé sur le stage préalablement suivi. Le stage peut donc servir d'objectif intermédiaire à atteindre avant de poursuivre le projet dans le cadre du TFE.

Ayant réalisé mon stage de 3^{ème} Bachelor chez AIRBUS DS SLC, j'aimerais découvrir une nouvelle entreprise travaillant dans les télécommunications. C'est pourquoi, je suis fortement intéressé de réaliser mon stage d'insertion professionnelle ainsi que mon TFE chez Thales afin d'en apprendre davantage sur la société mais aussi afin d'approfondir mes connaissances dans les télécommunications.

Vous trouverez ci-joint mon CV.

Je me tiens à votre entière disposition pour tout renseignement supplémentaire.

Dans l'attente d'une réponse favorable à ma demande, je vous prie d'agréer, Monsieur Dardenne, mes respectueuses salutations.



Thomas Anizet

THALES

D.2 Certificat de stage

THALES BELGIUM
Rue en Bois 63
4040 HERSTAL
Belgium
Tel. : +32 (0) 4 248 20 77
Fax : +32 (0) 4 248 25 10
www.thalesgroup.com

CERTIFICAT DE STAGE

Je soussigné, Alain QUEVRIN, en qualité de Chief Executive Officer, atteste par la présente que Thomas Anizet , né à Nivelles (Belgique), le 30 Avril 1996, et domicilié à Rue Henri Loriaux n°38, 6210 Les Bons Villers, a effectué un stage dans notre entreprise du 17 Septembre 2018 au 26 Octobre 2018.

Fait à Tubize, le 26 Octobre 2018.

Alain QUEVRIN

CEO



(po. Alain Quevrin)

Cachet de l'entreprise

THALES Belgium
Rue en Bois 63
4040 Herstal