

Table des matières

Introduction	3
1 Environnement de stage	4
1.1 Historique	4
1.2 Structure/Organisation - Domaines d'activité	5
2 Objectifs	6
3 Projet du stage	6
3.1 Notions de chiffrement	6
3.2 Algorithme <i>AES (Advanced Encryption Standard)</i>	8
3.3 Introduction aux <i>attaques par canal auxiliaire</i>	12
3.3.1 Types d'attaques	12
3.3.2 Introduction aux attaques par analyse de la consommation de puissance	13
3.4 Attaque CPA (<i>Correlation Power Analysis</i>)	15
3.4.1 Technologie CMOS	15
3.4.2 Puissance statique et dynamique	16
3.4.3 Composition des traces de puissance	18
3.4.4 Modèles de puissance	18
3.4.5 L'attaque CPA en concret	18
3.5 Simulations sur MATLAB	19
3.6 Contre-mesures	24
3.6.1 Contre-mesures <i>Hiding</i>	25
3.6.2 Contre-mesures <i>Masking</i>	26
3.6.3 Contre-mesures <i>Faking</i>	27
4 Conclusion	28
Crédits	29
Références	29
A Code algorithme AES-128	30
B Sbox - Table de substitution	30
C Opération <i>KeySchedule</i>	31
D Traces	32
E Corrélation pour les clés 1 à 4	33
F Corrélations 3D	34

G Corrélation pour la clé 8	35
H Contre-mesures <i>Hiding</i>	36
I Contre-mesure <i>Faking</i>	37

Introduction

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

Durant notre 2ème année de Master, les étudiants doivent réaliser un stage d'immersion en entreprise d'une durée de six semaines. Ce stage consiste, entre autres, à s'insérer dans une entreprise afin d'y découvrir différents aspects tels que l'organisation générale d'une entreprise, son vécu, sa structure interne et ses domaines d'activité. Il a également pour but de se familiariser au travail quotidien de l'ingénieur en participant de façon autonome à diverses activités.

Ayant réalisé mon stage de 3ème Bachelier chez **AIRBUS DS SLC** sur le site de Diegem et de Elancourt, il était important pour moi de saisir la chance et l'opportunité de découvrir une nouvelle entreprise renommée à travers le monde. C'est ainsi que je décidai de réaliser mon stage chez **THALES Telecommunications Belgium** sur le site de Tubize. L'objectif principal de ce stage de six semaines fut essentiellement d'introduire et de parcourir l'ensemble des notions théoriques et pratiques sur un sujet peu connu à l'heure actuelle : les "*Side Channel Attacks*" (SCA). En effet, devant réaliser par la suite mon *Travail de Fin d'Étude* (TFE) chez Thales, ce stage devait me permettre d'apprendre et de développer le bagage nécessaire pour la mise en oeuvre de mon sujet de TFE, à savoir : "*Développer une contre-mesure pour les attaques par analyse de la consommation de puissance*".

1 Environnement de stage

Cette section a pour objectif de décrire l'entreprise à différents points de vue. Tout d'abord, une description de l'entreprise d'un point de vue historique est abordée. Ensuite, sont détaillés succinctement la structure de l'entreprise, son organisation ainsi que ses domaines d'activités.

1.1 Historique

C'est donc chez **Thales Telecommunication Belgium** que je me suis rendu pour réaliser mon stage d'immersion en entreprise. Thales est une société anonyme (S.A.) d'origine Française. Elle est spécialisée dans cinq domaines clés, à savoir : l'aéronautique, l'espace, la défense, la sécurité et le transport terrestre. Née d'une fusion entre plusieurs entreprises, Thales hérite ainsi d'un passé prestigieux qui remonte à plus d'un siècle. La figure 1 ci-dessous retrace les dates-clés qui ont permis à Thales de se développer progressivement mais sûrement.

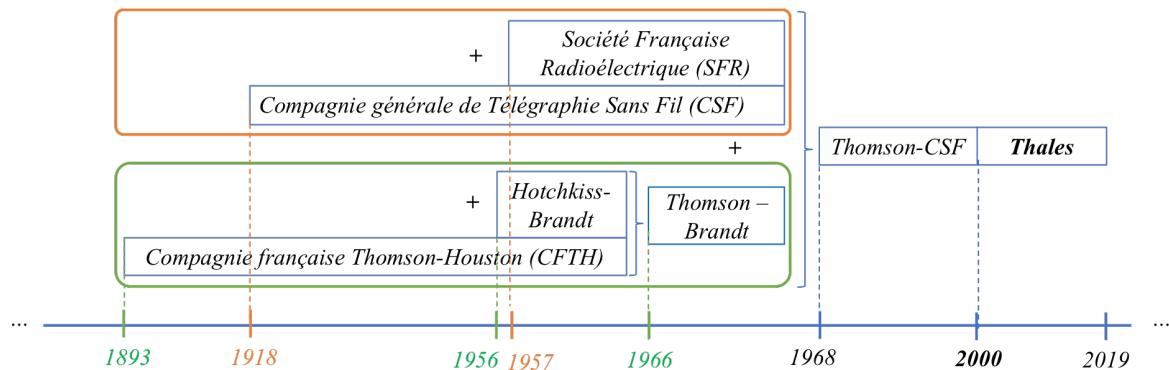


Figure 1 : Ligne du temps reprenant les dates majeures dans la création de la société **Thales**.

Il faut remonter en **1893** pour connaître l'origine de **Thales**. À cette date figure la naissance de la **Compagnie Française Thomson-Houston (CFTH)**. Cette compagnie avait essentiellement pour but d'exploiter en France les brevets de la société américaine *Thomson-Houston Electric Company*, dans le domaine de la production et du transport de l'électricité.

Parallèlement, en **1918** est créé la **Compagnie Générale de Télégraphie sans Fil (CSF)** qui deviendra plus tard l'un des pionniers des transmissions hertziennes. En **1957**, **CSF** absorbe la **Société Française Radioélectrique (SFR)**. Toutes deux tiennent un rôle primordial dans le développement de la radiodiffusion, des radiocommunications sur ondes courtes, de l'électro-acoustique ainsi que du radar et de la télévision.

En **1966**, la **Compagnie Thomson-Houston** est renommée **Thomson-Brandt** en raison de sa fusion avec **Hotchkiss-Brandt** (créé en **1956**), fabriquant d'appareils électroménagers, de voitures, d'armes et de munitions.

Deux années plus tard, c'est-à-dire en **1968**, né **Thomson-CSF**, fusion de **CSF** et des activités d'électronique professionnelle de **Thomson-Brandt**. **Thomson-CSF** se développe dans divers domaines. Citons comme exemples les composants électroniques tels que les semi-conducteurs en silicium, l'imagerie médicale ou encore la commutation téléphonique numérique. Cependant, en **1982**, la société est nationalisée en raison d'une situation financière difficile. Cela s'explique par le fait que le portefeuille d'activités est très diversifié, ce qui rend ses parts de marchés trop faibles pour être rentables. Dans les années qui suivent, l'entreprise se recentre sur les activités d'électronique professionnelle et de la défense.

En **1998**, **Aérospatiale**, **Alcatel**, **Dassault Industries** et **Thomson-CSF** signent un accord de coopération grâce au gouvernant français. À l'issu de celui-ci, **Thomson-CSF** obtient les activités d'électronique professionnelle et de défense d'**Alcatel** et **Dassault Electronique**. De cette façon, **Thomson-CSF** consolide son périmètre d'activité, ses positions concurrentielles dans la défense et l'électronique industrielle, ainsi que son implantation dans plusieurs pays européens. S'en suivent diverses acquisi-

tions à travers le monde, permettant à la société de se développer d'avantage. Ainsi, en **juillet 2000**, une nouvelle organisation en trois pôles est mise en place. Cette organisation s'articule autour de la **défense**, l'aéronautique, et des technologies de l'information et des services. Cette même année, en **décembre 2000**, pour marquer le coup, *Thomson-CSF* devient la compagnie privée ***Thales***.

L'organisation actuelle de *Thales* a été mise en place en **2007**, année où *Thales* acquiert les activités de transport, sécurité et aéronautique ***d'Alcatel-Lucent***. Cela a permis au groupe de devenir un des leaders mondiaux dans ses domaines d'études. Aujourd'hui, *Thales* est présent dans 56 pays à travers le monde et compte près de 64 000 employés. En 2016, *Thales* affiche un chiffre d'affaire de 14.9 milliards d'euros et a réinvesti près de 731 millions d'euros en *R&D*. Cela place l'entreprise comme le 10ème acteur mondial dans le domaine de la défense, qui représente 50% de ses ventes, l'autre part étant générée par le marché civil.

1.2 Structure/Organisation - Domaines d'activité

Depuis plus de 50 ans, *Thales* travaille pour les marchés belges de la défense, du spatial, de la sécurité et des transports. Actuellement, le groupe emploie environ 1000 salariés, répartis sur 7 sites à travers la Belgique : Bruxelles, Charleroi, Tubize, Herstal, Genk, Hasselt et enfin Leuven. Le site de Tubize, sur lequel j'ai effectué mon stage, compte environ 150 employés dont une part importante d'ingénieurs et de techniciens.

Le groupe laisse à chacune de ses divisions une certaine autonomie, ce qui permet à ***Thales Belgium*** de fonctionner avec la flexibilité d'une PME tout en bénéficiant de la solidité conférée par un grand groupe international. Elle possède un large panel de compétences : du hardware (conception et production), du software (développement et maintenance), de la sécurité des systèmes développés, de l'ingénierie système via le cycle IVVQ (Intégration, Validation, Vérification, Qualification). En parallèle, des équipes sont en charge des ventes et du marketing, du management des projets, des achats et d'autres fonctions supports. Cette variété de compétences permet de constituer pour chaque projet une équipe pluridisciplinaire qui prendra en charge l'ensemble du projet, du concept au prototypage en passant par l'industrialisation.

Comme dit précédemment, *Thales Group* est spécialisée dans cinq domaines clés : l'aéronautique, l'espace, le transport terrestre, la défense et la sécurité (voir figure 2).



Figure 2 : Les domaines d'activité du groupe ***Thales***.

Les domaines d'activités spécifiques au site de *Tubize* sont la conception et la maintenance de systèmes de communication et d'information :

- Systèmes de communication : équipements radio HF et VHF robustes et sécurisés (boîtes d'antennes, amplificateurs de puissance, coupleurs d'antennes) pour les communications terrestres, navales ou aéronautique, qu'elles soient civiles ou militaires.
- Systèmes d'information : dispositifs de gestion d'urgence, d'information aux usagers de transports, BMS (*Battlefield Management System*), soldat modernisé, vétroïque (systèmes électroniques de navigation et de communication des véhicules militaires).
- Protection de l'information : sécurisation des échanges de données et protection contre le vol et la publication de données confidentielles.

2 Objectifs

L'objectif de ce stage était d'introduire l'ensemble des notions élémentaires, nécessaires pour la réalisation du *Travail de Fin d'Étude* (TFE). Ce travail de fin d'étude qui allait se poursuivre durant 6 mois à compter du mois de Novembre 2018. Dans un premier temps, il s'agit donc surtout de définir les concepts théoriques. Ensuite, un exemple ou une simulation est mise en oeuvre afin d'appuyer le concept théorique.

La liste ci-dessous reprend l'ensemble des objectifs fixés et réalisés durant les 6 semaines de stage :

1. Chiffrement
2. L'algorithme AES (*Advanced Encryption Standard*)
3. Introduction aux *Side-Channel Attacks* (attaques par canal auxiliaire)
4. Cas spécifique des *Side-Channel Attacks* : Attaques CPA
5. Simulation d'une attaque CPA sur MATLAB
6. Contre-mesures
7. Simulation contre-mesures sur MATLAB

Ces différents objectifs sont décrits dans la section 3 "Projet du stage".

3 Projet du stage

Cette section décrit l'ensemble des objectifs, cités à la section 2, fixés pour le stage.

3.1 Notions de chiffrement

Les systèmes de sécurité modernes utilisent des algorithmes de chiffrement pour assurer la disponibilité, la confidentialité et l'intégrité de données. Ces algorithmes de chiffrement sont en réalité des fonctions mathématiques qui prennent typiquement :

- 2 paramètres en entrée : un *message clair* (nommé *plaintext* en anglais) et une *clé de chiffrement* (nommée *key* en anglais).
- 1 paramètre en sortie : le *message chiffré* (nommé *ciphertext* en anglais).

Le procédé transformant les données claires en entrée en données chiffrées en sortie est appelé le **chiffrement**. Ce procédé est réalisé grâce à un *algorithme de chiffrement* utilisant une clé de chiffrement et diverses opérations mathématiques. Il est important de préciser que tous les détails décrivant le fonctionnement d'un algorithme sont disponibles publiquement, seule la clé de chiffrement doit rester secrète. En effet, la sécurité offerte par un algorithme de chiffrement ne doit pas dépendre du secret de son implémentation. Un bon algorithme est un algorithme dont on ne parviendra pas à déchiffrer les données chiffrées. Lorsque la clé d'un algorithme est trouvée, le déchiffrement des données confidentielles peut être réalisé. On dit que l'algorithme de chiffrement est *cassé*.

La figure 3 ci-dessous présente le principe de fonctionnement d'un algorithme de chiffrement.

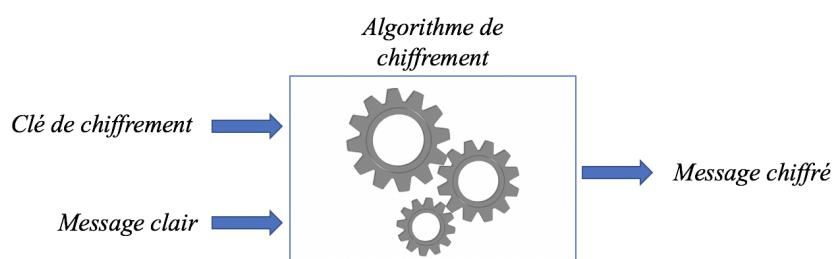


Figure 3 : L'algorithme de chiffrement, caractérisé par diverses opérations mathématiques, utilise une clé de chiffrement en entrée pour chiffrer un message clair. Cela produit un message chiffré, non compréhensible pour une personne ne connaissant pas la clé de chiffrement.

Nous distinguons 2 types d’algorithmes de chiffrements :

- **Chiffrement symétrique** : Le chiffrement est dit symétrique lorsque le procédé de chiffrement (algorithme) utilise une seule clé, 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. 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 (*Advanced Encryption Standard*). Une explication plus détaillée de cet algorithme est reprise à la section 3.2
- **Chiffrement asymétrique** : Le chiffrement est dit asymétrique lorsque le procédé de chiffrement (algorithme) utilise 2 clés : une *clé publique* et une *clé privée*. 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 code, dans un algorithme de chiffrement donné, un message. 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*).

Les figures 4 et 5 ci-dessous présentent les principes de fonctionnement des chiffrements symétriques et asymétriques respectivement.

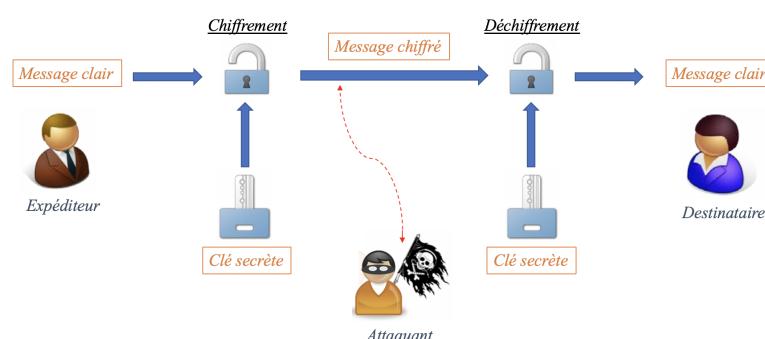


Figure 4 : Chiffrement symétrique : Une seule clé est utilisée pour chiffrer et déchiffrer les messages.

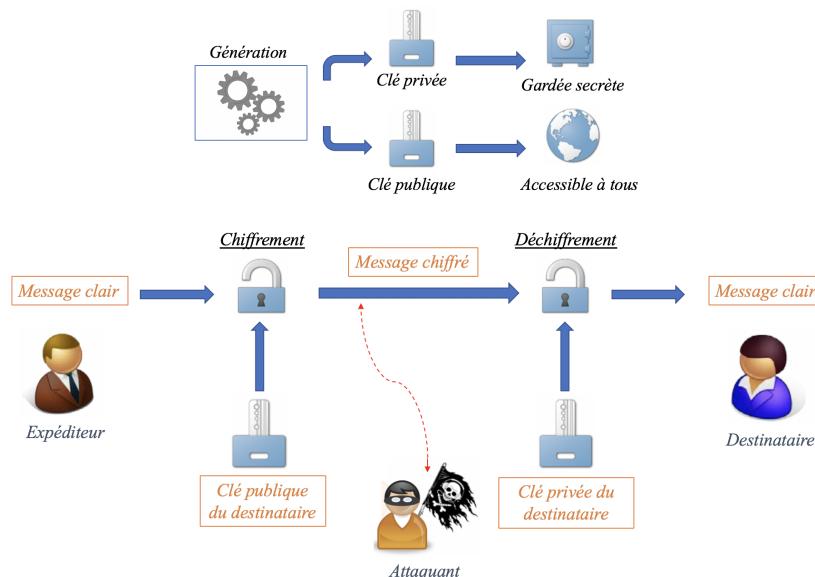


Figure 5 : Chiffrement asymétrique : Un clé publique est utilisée pour chiffrer le message et une clé privée est utilisée pour le déchiffrer.

Que ce soit pour un algorithme de chiffrement symétrique ou asymétrique, la clé de chiffrement doit être stockée sur un support physique. Ce support, appelé *device cryptographique*, doit être suffisamment sécurisé pour contenir de manière protégée la clé. Ainsi, un **device cryptographique** est un device qui implémente des algorithmes de chiffrement et qui stocke des clés de chiffrement (exemple : *FPGA*).

3.2 Algorithme 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**. C'est-à-dire 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 cependant, comme nous le verrons par la suite, avec les attaques par canal auxiliaire (section 3.3), le problème peut vite être contourné. La description qui suit est basée sur l'algorithme AES-128 bits, c'est-à-dire que la clé de chiffrement a une taille de 128 bits.

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 représentés 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) appelé STATE contient l'ensemble des résultats intermédiaires résultant des diverses opérations que subissent les données (depuis leur état initial).

La figure 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 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.

Par ailleurs, 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 (12 tours pour une clé de 192 bits et 14 tours pour une clé de 256 bits). Un tour est défini par 4 opérations appliquées succinctement sur la matrice STATE. Ces 4 opérations sont : *AddRoundKey*, *SubBytes*, *ShiftRows* et *MixColumns*. Elles sont appliquées à divers instants dans l'exécution de l'algorithme AES. La figure 7 (page suivante) permet de visualiser l'ordre d'exécution chronologique de ces 4 opérations. L'annexe A reprend quant à elle le code réalisé pour l'exécution de l'algorithme AES-128.

La figure 7 ci-dessous présente le principe de fonctionnement général de l'algorithme AES-128.

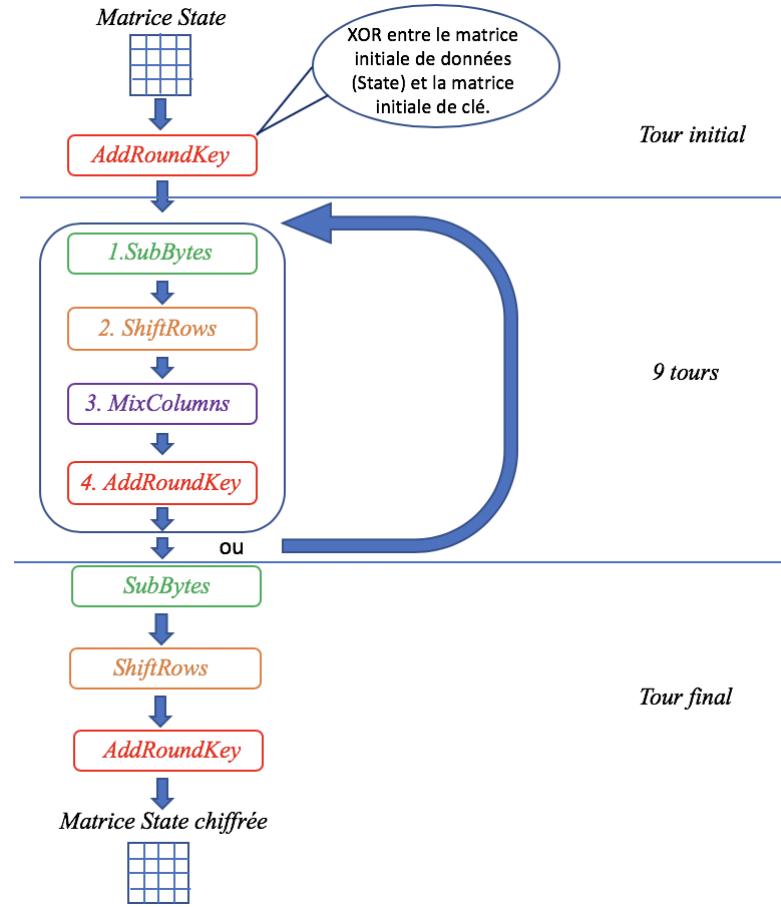


Figure 7 : Principe de fonctionnement de l'algorithme AES-128.

Fonctionnement :

Initialement, deux matrices vont être utilisées : la matrice STATE, contenant les données claires, et la matrice clé, contenant la clé secrète initiale. 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 8 ci-dessous présente le principe de fonctionnement de cette première opération :

<table border="1"> <tr><td>12</td><td>e0</td><td>13</td><td>28</td></tr> <tr><td>04</td><td>ab</td><td>f8</td><td>d3</td></tr> <tr><td>23</td><td>19</td><td>69</td><td>26</td></tr> <tr><td>e5</td><td>b9</td><td>7a</td><td>4c</td></tr> </table>	12	e0	13	28	04	ab	f8	d3	23	19	69	26	e5	b9	7a	4c	\oplus	<table border="1"> <tr><td>ab</td><td>88</td><td>23</td><td>2a</td></tr> <tr><td>10</td><td>fe</td><td>a3</td><td>6c</td></tr> <tr><td>fe</td><td>2c</td><td>39</td><td>75</td></tr> <tr><td>17</td><td>b1</td><td>39</td><td>05</td></tr> </table>	ab	88	23	2a	10	fe	a3	6c	fe	2c	39	75	17	b1	39	05	=	<table border="1"> <tr><td>b9</td><td>68</td><td>30</td><td>02</td></tr> <tr><td>14</td><td>55</td><td>5b</td><td>bf</td></tr> <tr><td>dd</td><td>35</td><td>50</td><td>53</td></tr> <tr><td>f2</td><td>08</td><td>43</td><td>49</td></tr> </table>	b9	68	30	02	14	55	5b	bf	dd	35	50	53	f2	08	43	49
12	e0	13	28																																																	
04	ab	f8	d3																																																	
23	19	69	26																																																	
e5	b9	7a	4c																																																	
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																																																	
<i>Matrice STATE</i>		<i>Matrice clé</i>		<i>Nouvelle matrice STATE</i>																																																

Figure 8 : Opération *AddRoundKey* entre la matrice STATE et la matrice clé.

Ensuite, une série de quatre opérations se répétant neuf fois (9 tours cycliques) est exécutée. Ces quatres opérations sont appliquées dans l'ordre suivant : *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*. Enfin, une fois les neufs tours exécutés, le neuvième et dernier *round* se lance exécutant trois opérations : *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 9 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 B. 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 Sbox. Un exemple est donné afin de comprendre le principe : Si le byte $S_{1,2}$ vaut 53 (hexa) alors, selon la Sbox, la valeur du byte résultant $S'_{1,2}$ vaut ed (hexa).

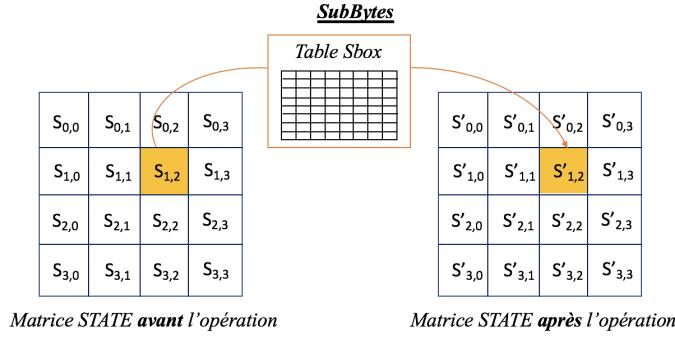


Figure 9 : Opération *SubBytes* exécutée sur la matrice STATE.

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 10 ci-dessous présente le principe de fonctionnement de l'opération *ShiftRows* :

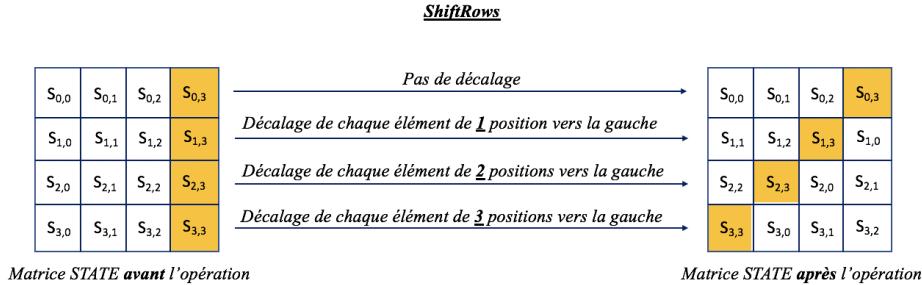


Figure 10 : Opération *ShiftRows* exécutée sur la matrice STATE.

3. MixColumns :

Comme son nom 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 12) et un vecteur colonne (taille 4x1) de la matrice STATE.** Cela produit un nouveau vecteur colonne (taille 4x1) permettant de définir la nouvelle matrice STATE. La figure 11 ci-dessous présente le principe de fonctionnement de l'opération *MixColumns*. Un exemple est ensuite donné à la figure 12.

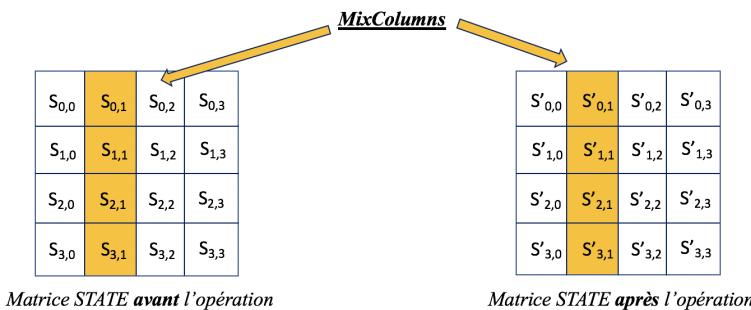


Figure 11 : Opération *MixColumns* exécutée sur la matrice STATE.

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

Colonne 1 de la matrice STATE après opération Matrice fixée Colonne 1 de la matrice STATE avant 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 12 : Exemple de l'opération *MixColumns* exécutée sur la deuxième colonne (i.e colonne 1) de la matrice STATE.

4. AddRoundKey : La gestion des clés se fait au travers des fonctions *AddRoundKey* et *KeySchedule*. 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.

On sait que la matrice STATE est 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. L'opération qui modifie la matrice de clé initiale en une nouvelle matrice de 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, 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*.

La figure 8 reste inchangée pour décrire le principe de fonctionnement de l'opération *AddRoundKey*. Nous rappelons toutefois que la matrice clé est bien modifiée lors de chaque exécution de cette opération.

L'annexe C permet quant à elle 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.

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 1^{ère} colonne de la nouvelle clé (soit les quatre 1ers bytes), on va prendre 3 vecteurs colonnes (taille 4x1), à savoir :

1. Prendre la 4^{ème} 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 1^{ère} colonne de la clé initiale.
3. Prendre la 1^{ère} 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 1^{ère} 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 C permet de comprendre plus facilement ce qui vient d'être énoncé.

3.3 Introduction aux attaques par canal auxiliaire

3.3.1 Types d'attaques

A la fin des années 1990, une nouvelle contrainte pour la conception de système informatique 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, justement, 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 de TFE.

La figure 13 ci-dessous résume les différents types d'attaques physiques possibles.

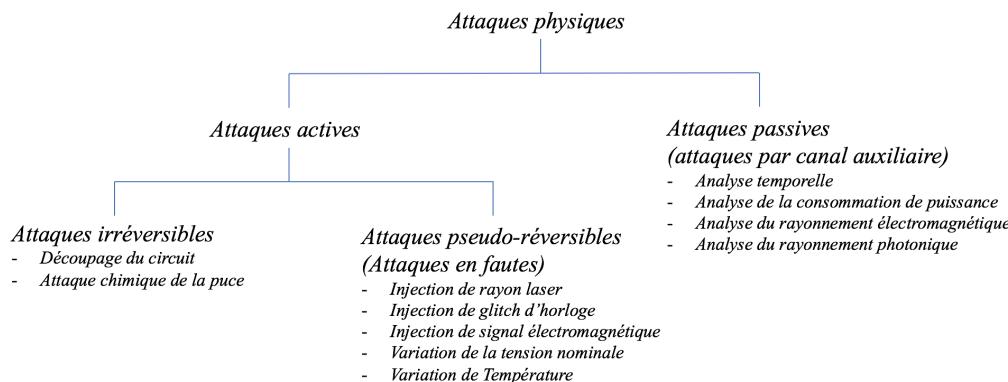


Figure 13 : Les 2 grandes familles d'attaques physiques possibles. La suite de ce rapport se concentre essentiellement sur les attaques physiques dites passives.

Les attaques passives sont globalement beaucoup plus simples à mettre en œuvre 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 14 ci-dessous présente les différentes façons possibles d'attaquer **passivement** un device.

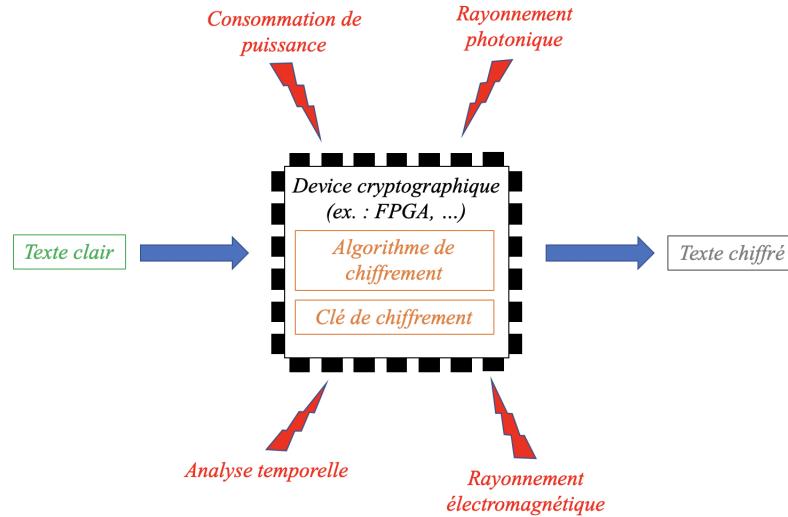


Figure 14 : Les différentes façons d'attaquer passivement un device cryptographique en vue de casser l'algorithme de chiffrement qui se trouve dessus.

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.

3.3.2 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 électrique 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 électrique (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 15 ci-dessous présente le principe de mesure à l'oscilloscope.

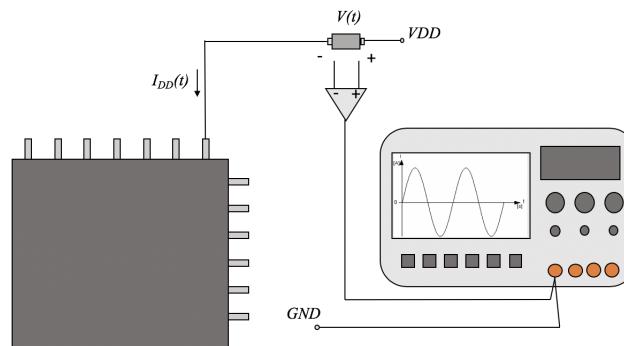


Figure 15 : Principe de mesure à l'oscilloscope.

On parle d'attaque par analyse de la consommation de puissance ou avec l'oscilloscope, on mesure une tension et non une puissance. Cela est juste car comme le démontre l'équation 2, la puissance consommée ($p(t)$) est bien proportionnelle à la tension consommée ($V(t)$). En effet, en supposant que la tension d'alimentation V_{DD} est constante et par simple application de la loi d'Ohm, on a :

$$\begin{cases} u(t) = V_{DD} \\ i(t) = \frac{V(t)}{R} \end{cases} \quad (1)$$

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

$$p(t) = u(t).i(t) = V_{DD} \cdot \frac{V(t)}{R} \quad (2)$$

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, c'est 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 continuant l'introduction sur les attaques par analyse de la consommation de puissance, et en prenant le cas particulier d'une attaque dite CPA, nous pouvons dire que :

En supposant connu 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 de coefficient 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 16 présente vulgairement le principe général d'une attaque CPA.

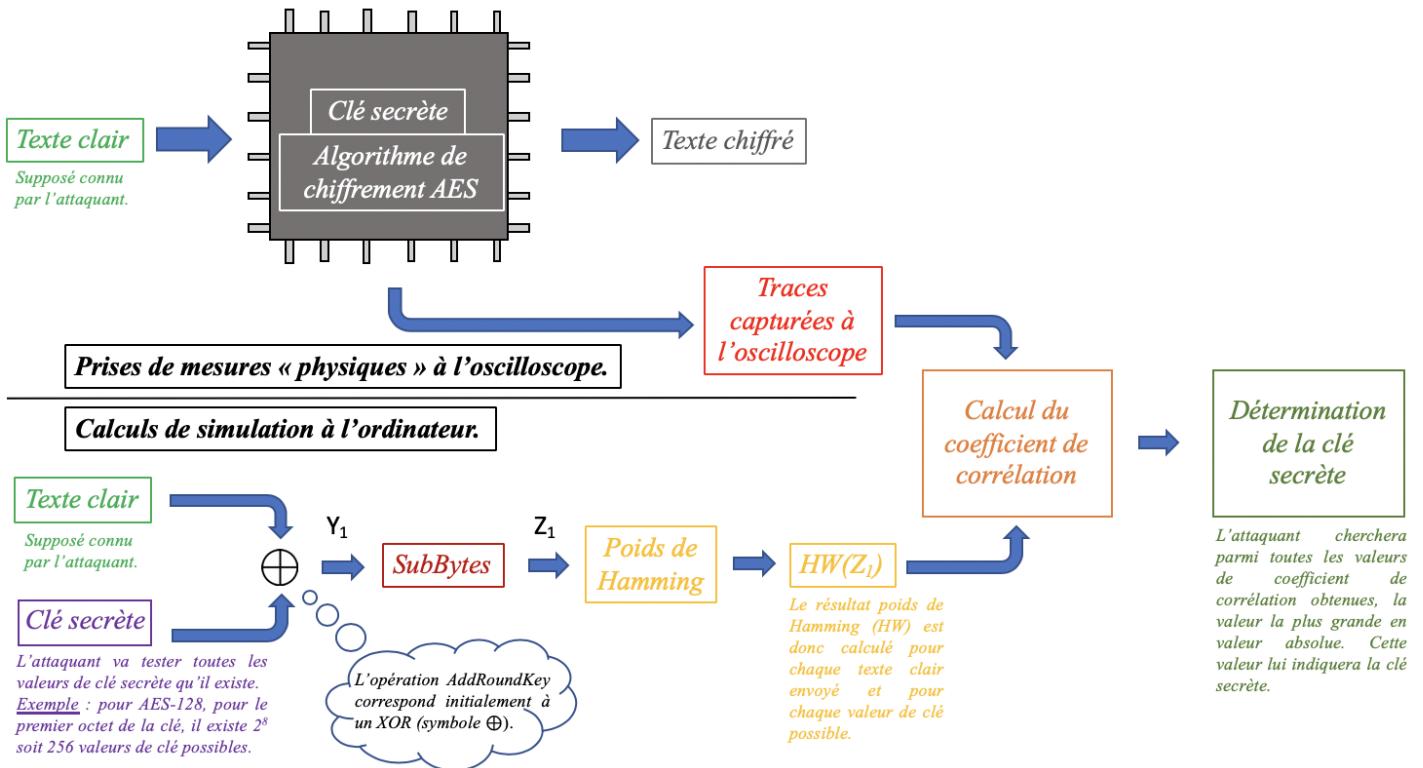


Figure 16 : Principe générale d'une attaque CPA.

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 3.4 définit toutes les notions élémentaires pour réaliser une attaque CPA.

3.4 Attaque CPA (Correlation Power Analysis)

3.4.1 Technologie CMOS

La *technologie CMOS* (pour *Complementary MOS*) 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 englobait 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, NAND, 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. Ainsi lorsqu'un des deux transistor 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 simplement une fonction *NON*. Voici donc sa table de vérité :

Entrée	Sortie
0	1
1	0

Figure 17 : Table de vérité pour la fonction NON (inverseur CMOS).

La figure 18 ci-dessous (schéma a) présente le schéma de l'inverseur CMOS :

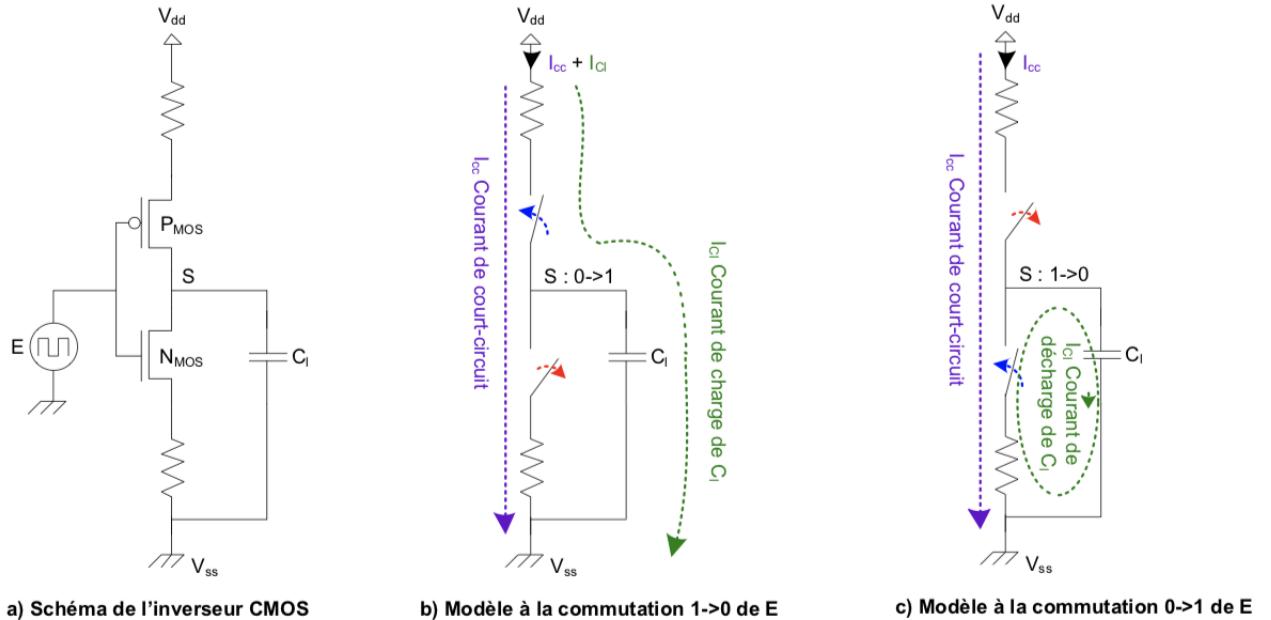


Figure 18 : Schéma de l'inverseur CMOS.

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 cet ouvrage, on considérera toujours que les circuits attaqués sont réalisés en technologie CMOS.

3.4.2 Puissance statique et dynamique

Il est évident que les circuits digitaux modernes consomment de la puissance lorsqu'ils exécutent des instructions (opérations) sur des données. Dans le domaine de la cryptanalyse, cette puissance va être mesurée et analysée afin de déterminer si le device cryptographique est attaquable ou non. Dans cette section, nous allons étudier plus particulièrement la consommation de puissance d'un type précis de circuit : les circuits utilisant la technologie CMOS (voir section 3.4.1 pour plus de précision). Cette technologie est très répandue et couvre la plupart des circuit 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 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 3.4.1), nous allons tenter de comprendre quand et pourquoi 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 cette cellule commute.

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

$$P_{total} = P_{stat} + P_{dyn} \quad (3)$$

Mais de façon plus précise, que vaut la puissance statique ? De même, comment pourrait-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 (3.4.1), 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 pas totalement 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 :

$$P_{stat} = I_{fuite} \cdot V_{DD} \quad (4)$$

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 19 ci-dessous.

Transitions	Type de puissance consommée
$0 \rightarrow 0$	Statique
$0 \rightarrow 1$	Statique + Dynamique
$1 \rightarrow 0$	Statique + Dynamique
$1 \rightarrow 1$	Statique

Figure 19 : 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 dans les deux transitions suivantes : 0-1 et 1-0. 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 0-0 et 1-1), la puissance totale reste plus ou moins constante. En effet, la puissance dynamique étant nulle, on ne retrouve dans le calcul de puissance total que la puissance statique. Autrement dit : $P_{total} = P_{stat}$.
- **Lorsqu'il y a une commutation** (transitions 0-1 et 1-0), la puissance totale augmente. En effet, en plus de la puissance statique, 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.

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 18 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), la 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, 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 calculé de la façon suivante (5) :

$$P_{chrg} = \frac{1}{T} \int_0^T p_{chrg}(t) dt = \alpha \cdot f \cdot C_L \cdot V_{DD}^2 \quad (5)$$

Où :

- $p_{chrg}(t)$ représente la consommation de puissance de chargement instantané 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 calculé de la façon suivante (5) :

$$P_{cc} = \frac{1}{T} \int_0^T p_{cc}(t) dt = \alpha \cdot f \cdot C_L \cdot V_{DD} \cdot I_{fuite} \cdot t_{cc} \quad (6)$$

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 existe.

En conclusion, les systèmes informatiques 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 du type de transition. Elle n'est **pas négligée**.

Ceci nous conduit donc à l'équation suivante (7) :

$$P_{total} = P_{stat} + P_{dyn} \cong P_{dyn} = P_{chrg} + P_{cc} \quad (7)$$

3.4.3 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 sera élevée et au plus l'analyse de la consommation de puissance sera difficile.

Ainsi, chaque point d'une trace de puissance peut être modélisé comme la somme des 3 composantes définies ci-dessus, soit : $P_{total} = P_{op} + P_{data} + P_{noise}$. De plus, en reprenant l'équation 7 définie dans la section 3.4.2, nous pouvons conclure que : $P_{total} = P_{op} + P_{data} + P_{noise} = P_{stat} + P_{dyn} \cong P_{dyn} = P_{chrg} + P_{cc}$

3.4.4 Modèles de puissance

Dans une attaque CPA, l'attaquant doit utiliser ce qu'on appelle un **modèle de puissance** afin de prédire la consommation de puissance du device cryptographique attaqué. Une fois ces prédictions réalisées, celles-ci sont comparées aux mesures réelles de consommation de puissance du device (prises à l'oscilloscope) dans le but de retrouver la clé secrète. 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).

1. **Poids de Hamming (HW)** : Le poids de Hamming est le modèle de consommation de puissance le plus basique. 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 pas à un excès 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 est proportionnelle au nombre de bits à 1 d'une exécution de traitement de données.
Exemple : HW(100110) = 3
2. **Distance de Hamming (HD)** :
Exemple : HD(110110 ; 100100) = 2

3.4.5 L'attaque CPA en concret

La figure ?? ci-dessous présente le principe de fonctionnement d'une attaque CPA comme décrite ci-dessus.

3.5 Simulations sur MATLAB

Afin de bien assimiler une attaque par canal auxiliaire, il m'a été demandé de tester, par des simulation 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 20 ci-dessous représente le schéma-bloc de la première phase de l'exercice 1, c'est-à-dire qu'il présente les différentes étapes à réaliser pour simuler un point d'une trace.

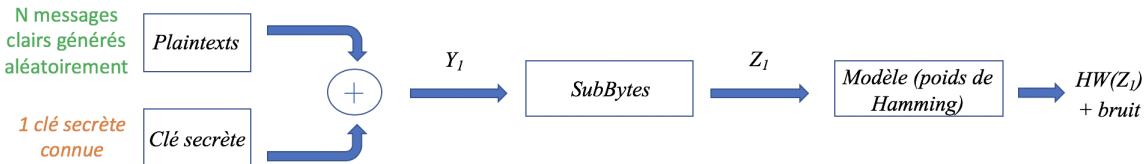


Figure 20 : Schéma-bloc permettant de comprendre la simulation d'un point d'une trace.

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, en réalité, lors d'une prise de mesure, un élément parasite vient toujours s'additionner au signal que l'on étudie, il s'agit de bruit électronique.

La figure 21 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.

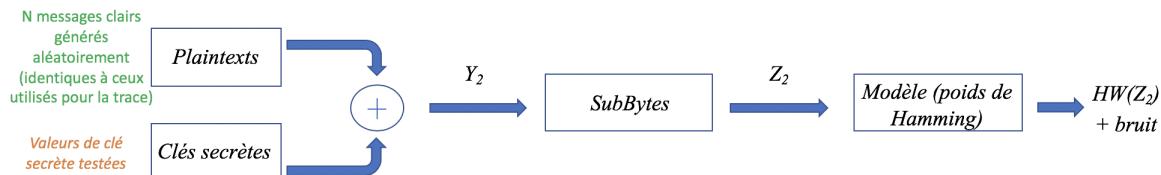


Figure 21 : Schéma-bloc permettant de comprendre la seconde phase de l'exercice 1.

La figure 21 est quasi identique à la figure 20. 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 20 pour chiffrer nos N messages. Ainsi, si on ne s'intéresse qu'à un seul octet de données et donc, par la même occasion, 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 22 présente les différentes tailles de matrices utilisées pour réaliser la simulation.

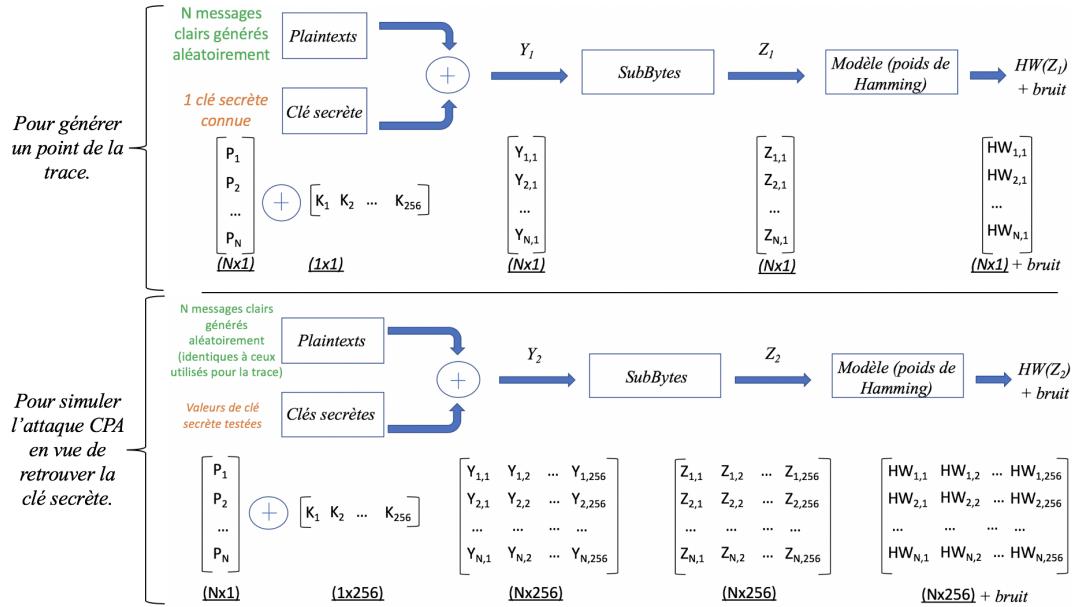


Figure 22 : Schéma-bloc permettant de visualiser la taille des différentes matrices employées pour la simulation.

Comme vu à la section 3.2, 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 $N \times 1$. 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 1×1 . 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 1×256 . 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 $N \times 1$ et $N \times 256$ respectivement. En effet, pour Y_1 , une seul clé est utilisée alors que pour Y_2 , 256 valeurs de clé sont utilisées.
- Z_1 et Z_2 sont des matrices de taille $N \times 1$ et $N \times 256$ 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 $N \times 1$ et $N \times 256$ pour les mêmes raisons 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 sera rendu possible en calculant le coefficient de corrélation pour chaque valeur de clé possible. Le figure 23 ci-dessous présente le calcul final qui permettra de retrouver le premier octet de la clé secrète.

$$\text{corrélation} \begin{bmatrix} HW_{1,1} & HW_{1,1} & HW_{1,2} & \dots & HW_{1,256} \\ HW_{2,1} & ; & HW_{2,1} & HW_{2,2} & \dots & HW_{2,256} \\ \dots & & \dots & \dots & \dots & \dots \\ HW_{N,1} & & HW_{N,1} & HW_{N,2} & \dots & HW_{N,256} \end{bmatrix} = \begin{bmatrix} Corr_{1,1} & Corr_{1,2} & \dots & Corr_{1,256} \\ Corr_{2,1} & Corr_{2,2} & \dots & Corr_{2,256} \\ \dots & \dots & \dots & \dots \\ Corr_{N,1} & Corr_{N,2} & \dots & Corr_{N,256} \end{bmatrix}$$

Figure 23 : Calcul du coefficient de corrélation entre un point d'une trace simulée et le poids de Hamming.

Les graphes 24, 25 et 26 ci-dessous présentent les résultats obtenus. Il s'agit de graphes indiquant les valeurs de coefficient 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 3.4.5, 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).

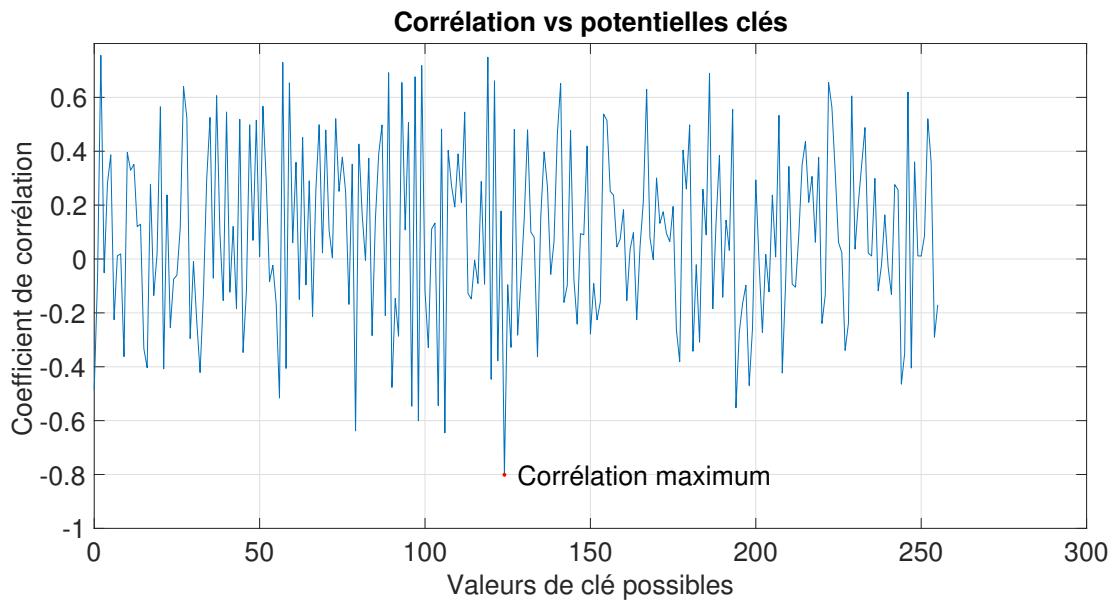


Figure 24 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 10 traces.

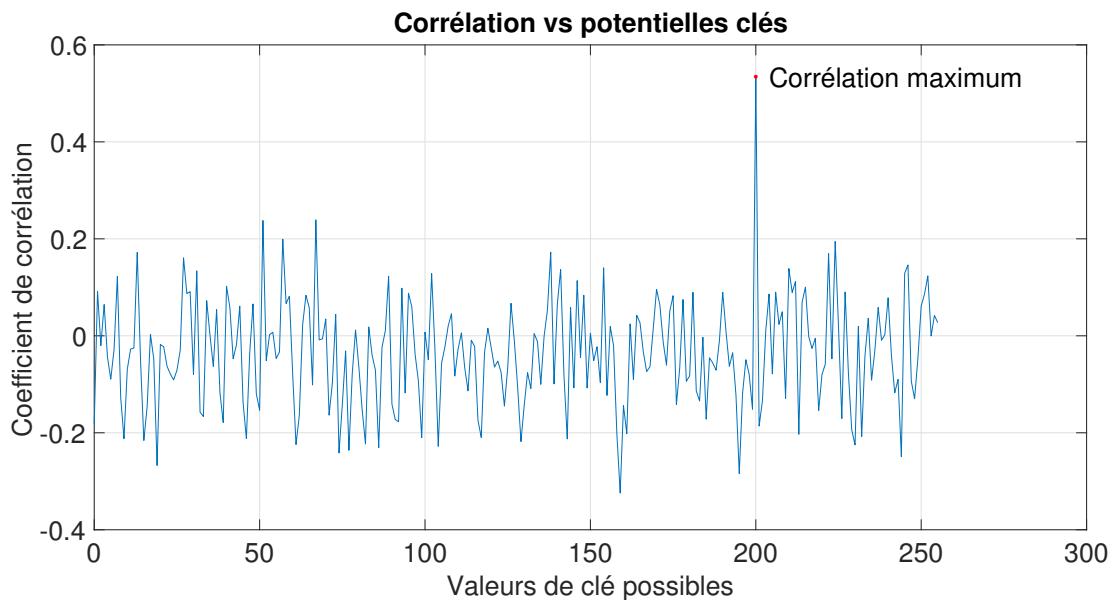


Figure 25 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 100 traces.

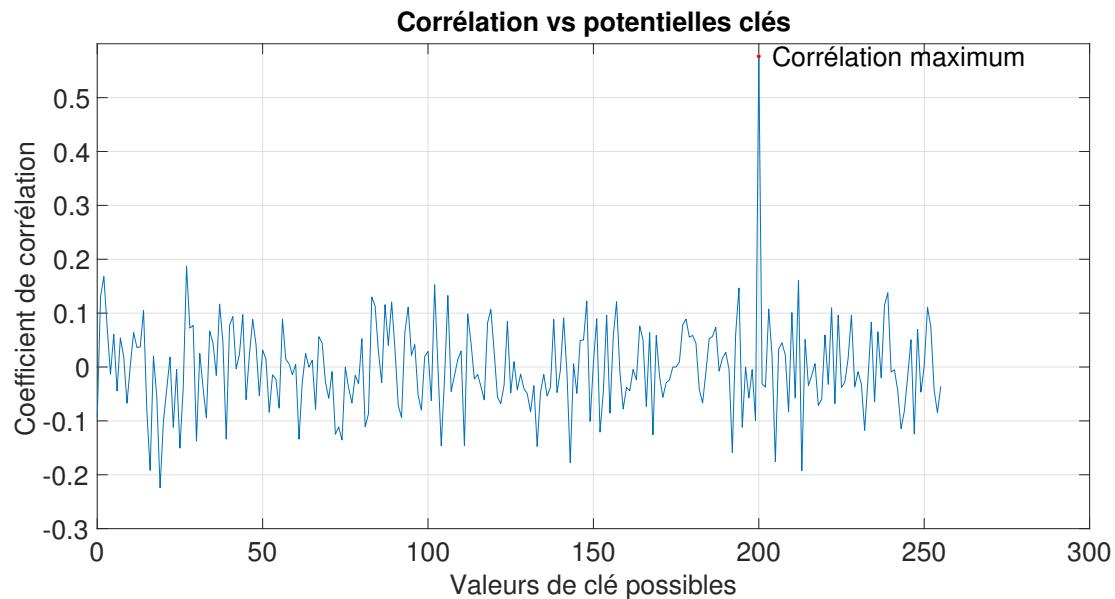


Figure 26 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 1000 traces.

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 27).

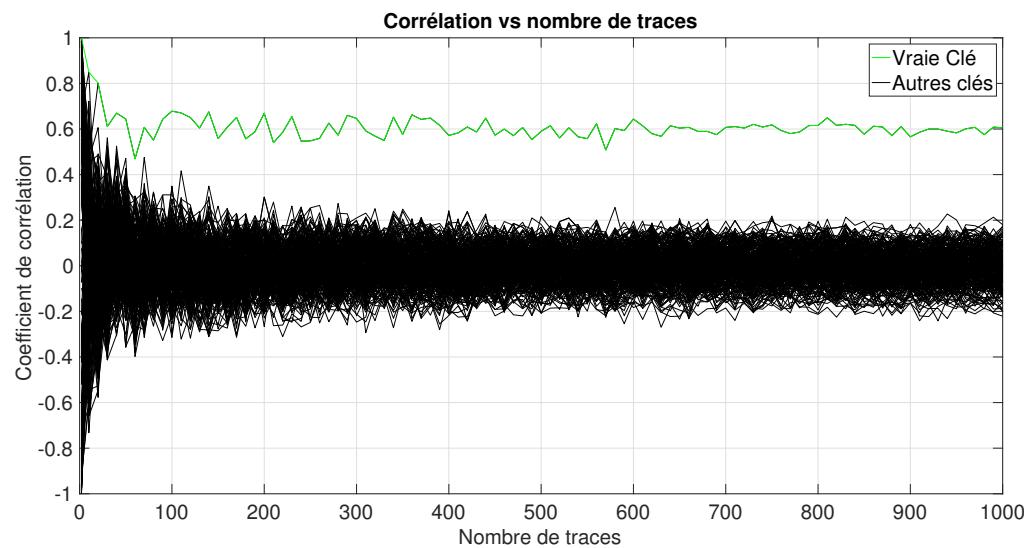


Figure 27 : Graphe présentant la valeur du coefficient de corrélation en fonction du nombre de traces. Plus le nombre de traces est élevé, plus la valeur de la clé secrète employée pour le chiffrement des données se distingue des autres clés. La vraie clé est 200.

Exercice 2 : La figure 28 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.

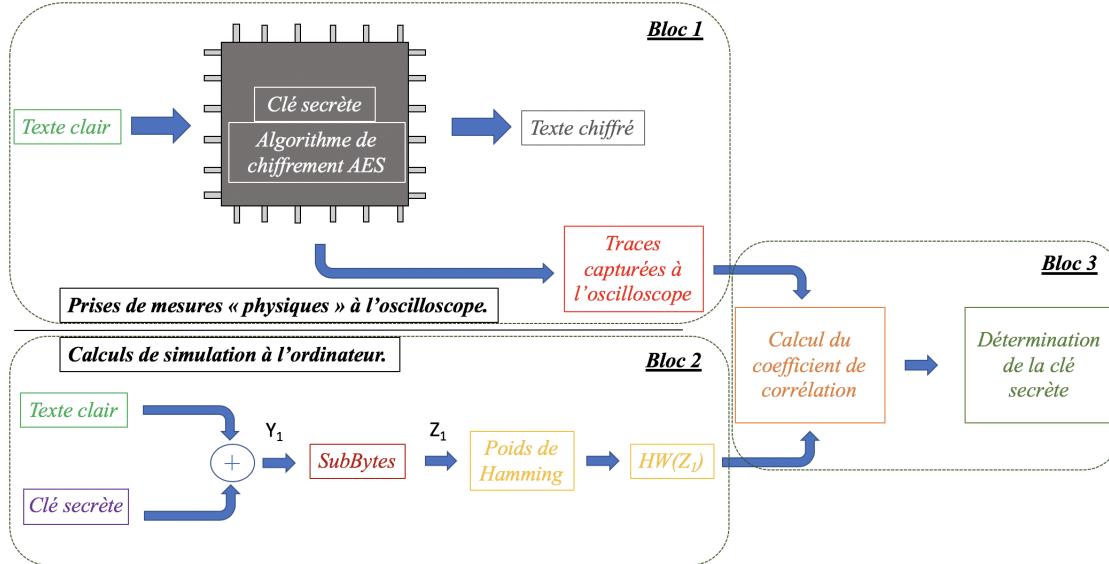


Figure 28 : Schéma-bloc permettant de comprendre le principe de fonctionnement du deuxième exercice.

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

- Premièrement, la prise de mesures à l'oscilloscope. Cette prise de mesure 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 21.
- 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 D présente les quatre premières traces, parmi les 2380, mesurées à l'oscilloscope.
2. L'annexe E présente les valeurs de coefficient 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 F présente en trois dimensions les valeurs de coefficient 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 G présente les valeurs de coefficient 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.

3.6 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) devaient 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 3.6.1) 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 que les traces de puissance obtenues à l'oscilloscope soient faussées. On va donc construire des devices cryptographique 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 3.3.2). 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 (8). Elle peut également être exprimée en dB (9) :

$$SNR = \frac{P_{signal}}{P_{noise}} \quad (8) \qquad SNR_{dB} = 10 \cdot \log_{10} \frac{P_{signal}}{P_{noise}} \quad (9)$$

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 de savoir si le signal de transmission que l'on étudie est fortement bruité ou non. Selon l'équation (8), 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. 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.

3.6.1 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 mal 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 29 ci-dessous présente les possibilités de contre-mesures mises en oeuvre selon le type d'approche suivie tandis que l'annexe H 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 29 : 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 H, 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 (seuls 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 3.4.2) 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 pourrait concerner du 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.

3.6.2 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 pour être plus précis). 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 dépendante 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 le contre-mesures Hiding.

Définition du masque

Une valeur intermédiaire masquée v_m est une valeur intermédiaire v cachée par une valeur aléatoire m : $v_m = v * m$. 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 le *vrai* message chiffré.

L'opération $*$ 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 : $v_m = v + m \pmod{n}$ ou $v_m = v \times m \pmod{n}$ où *modulo* n est défini en fonction de l'algorithme de chiffrement.

Fonctions linéaires et non-linéaires

Les algorithmes de chiffrement utilisent des fonctions linéaires mais aussi des fonctions non-linéaires. **Une fonction est dite *linéaire* si elle respecte la propriété suivante : $f(x * y) = f(x) * f(y)$. À l'inverse, une fonction est dite *non-linéaire* si elle ne respecte pas la propriété.**

Exemple : Dans l'algorithme AES, on retrouve l'opération XOR. Cette opération est dite linéaire car la propriété $f(x \oplus m) = f(x) \oplus f(m)$ est respectée. On remarque donc qu'il est facile de calculer un masque booléen pour une fonction linéaire. *ShiftRows* et *MixColumns* sont également linéaires. Un contre-exemple, dans l'algorithme AES, concerne l'opération *SubBytes*. Il s'agit d'une opération non-linéaire car $f(x \oplus m) \neq f(x) \oplus f(m)$. On remarque donc qu'il n'est pas facile de calculer un masque booléen pour une fonction non-linéaire.

Comme dit précédemment, chaque masque appliqué doit ensuite être retiré afin d'obtenir *in fine* le vrai message chiffré. Nous retiendrons donc ici que pour l'algorithme AES, si l'on souhaite appliquer un masque booléen (et c'est en général le cas le plus fréquent), il sera toujours plus facile de le réaliser sur des opérations linéaires (telles que *AddRoundKey*, *ShiftRows*, *MixColumns*) plutôt que sur des opérations non-linéaires (telles que *SubBytes*).

3.6.3 Contre-mesures *Faking*

Après plusieurs recherches sur *Internet*, j'ai découvert l'existence d'un nouveau type de contre-mesure : 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*.

Le but 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 : $Key_{Fake} = Key_{Real} \oplus Key_{Mask}$. À partir de cette fausse clé, on va exécuter normalement l'algorithme AES (comme décrit à la section 3.2) : 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 bien en mesure de retrouver une valeur de clé. Cependant, cette clé sera non pas la vraie clé initiale mais plutôt la fausse clé générée au départ. En conclusion, l'attaquant ne sera pas capable de 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 I 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*, *MixColumns*). *SubBytesTrans* est définie de façon à annuler l'utilisation de la fausse clé sur les données alors que *MixColumns* réalise la même opération que d'habitude. 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, observé à la figure 30 ci-dessous, 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.

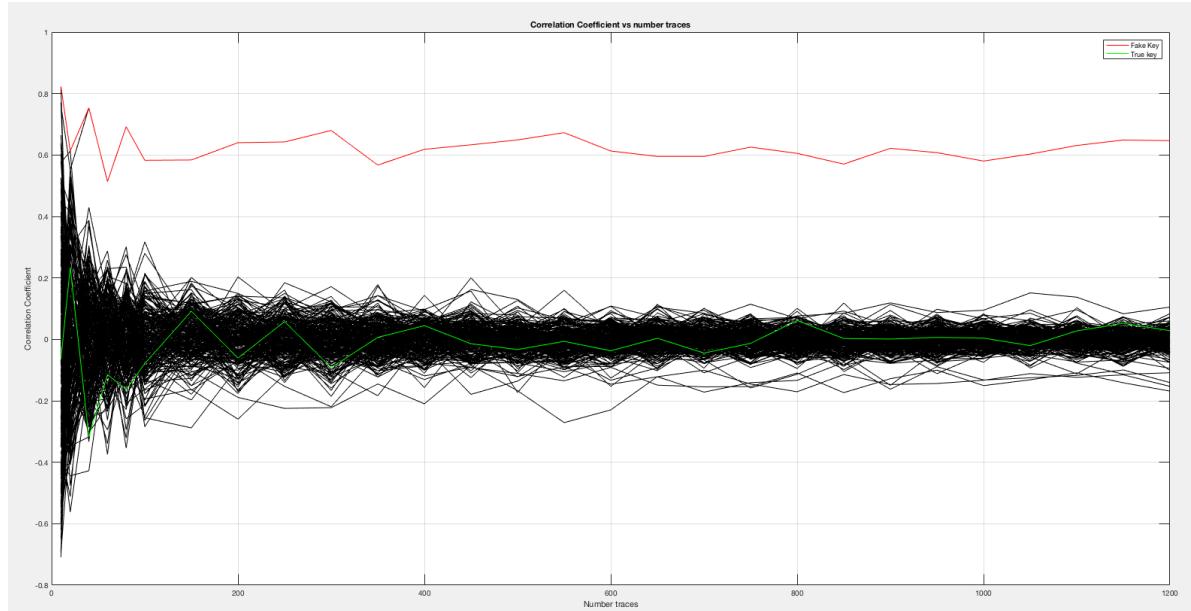


Figure 30 : On voit qu'au plus le nombre de traces augmente au plus une clé se distingue des 255 autres. Il s'agit de la fausse clé. La vraie clé ne laisse fuiter aucune information.

4 Conclusion

Crédits

- Figure 2 provenant du site internet : <http://monipag.com/victoria-petitier/wp-content/uploads/sites/1363/Thales-Group-1.png>
- Annexe B provenant du site internet de J.M. Dutertre "Synthèse AES 128" : https://www.emse.fr/~dutertre/documents/synth_AES128.pdf

Références

- [1] J.M. Dutertre. *Synthèse AES 128*. https://www.emse.fr/~dutertre/documents/synth_AES128.pdf, 2011.
- [2] Lilian BOSSUET. *Approche didactique pour l'enseignement de l'attaque DPA ciblant l'algorithme de chiffrement AES*. EDP Sciences, 25 Octobre 2012.
- [3] 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.
- [4] 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>.
- [5] 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>.
- [6] Abdelaziz Elaabid. *Attaques par canaux cachés : expérimentations avancées sur les attaques template*. <https://tel.archives-ouvertes.fr/tel-00937136/document>, 2014.
- [7] 6.1.2 *Distance de Hamming et poids d'un mot de code*. https://icwww.epfl.ch/~chappeli/it/courseFR/I2subsec_Hdist.php.
- [8] Jean-Philippe Muller. *Le bruit dans les systèmes électroniques*. <http://www.ta-formation.com/acrobat-cours/bruit.pdf>, Juillet 2002.
- [9] Renaud Dumont. *Cryptographie et Sécurité informatique INFO0045-2*. <http://www.montefiore.ulg.ac.be/~dumont/pdf/crypto09-10.pdf>, 2010.
- [10] Thales Group. *Historique*. <https://www.thalesgroup.com/fr/global/groupe/historique>, 2018.
- [11] Thales. *Wikipedia*. <http://fr.wikipedia.org/w/index.php?title=Thales&action=history>, Octobre 2018.

A Code algorithme AES-128

Algorithme 3 Chiffrement AES

Require: state, key

Ensure: state

```

KeyExpansion(key,expandkey[])
AddRoundKey(state,expandkey[0])
for i = 1 < 10 do
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state,expandkey[i])
    i = i + 1
end for
SubBytes(state)
ShiftRow(state)
AddRoundKey(state,expandkey[10])

```

Figure 31 : Code définissant l'ordre des opérations exécutées par l'algorithme AES-128.

B Sbox - Table de substitution

		y																
		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	d1	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

Figure 32 : 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.

C Opération KeySchedule

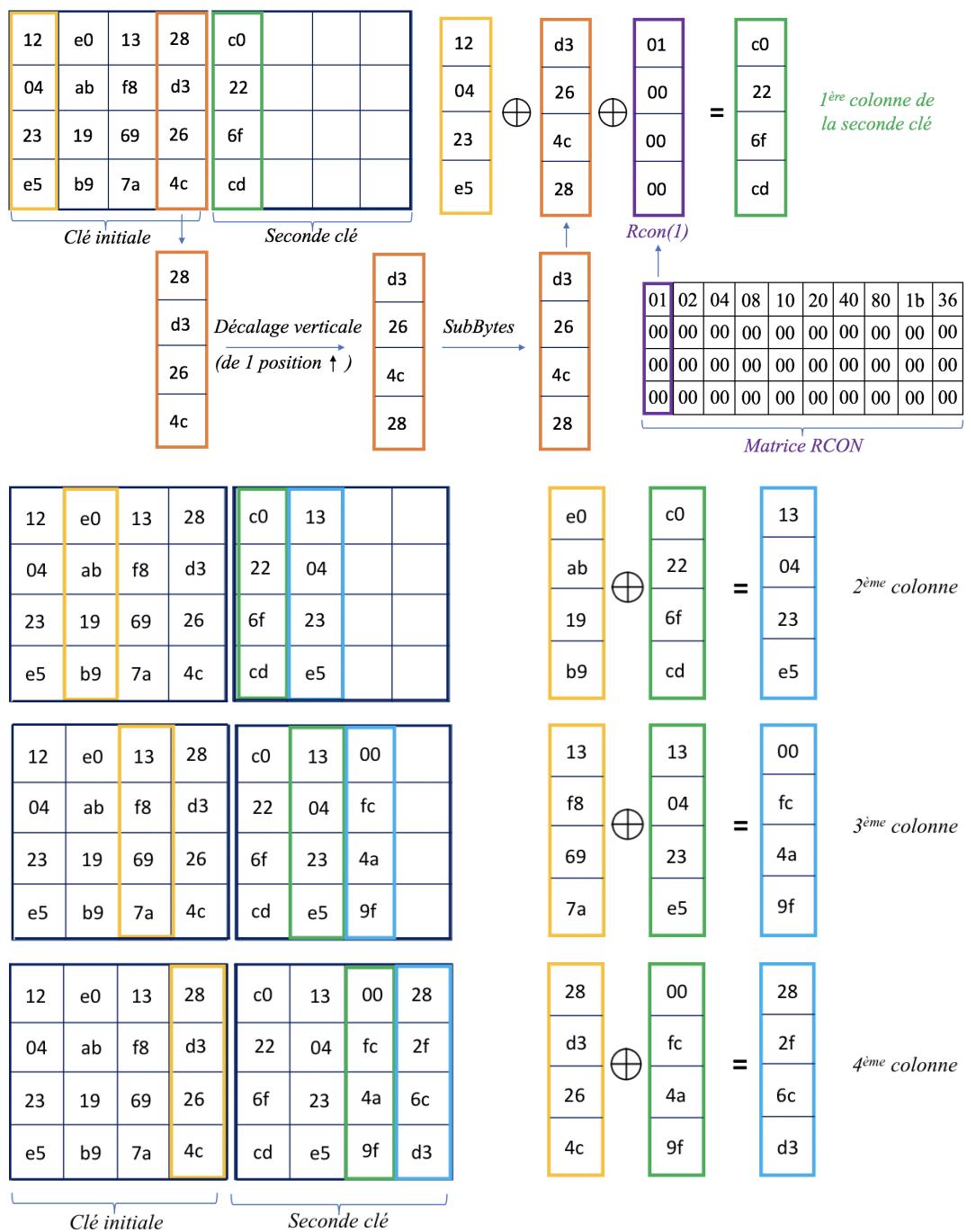
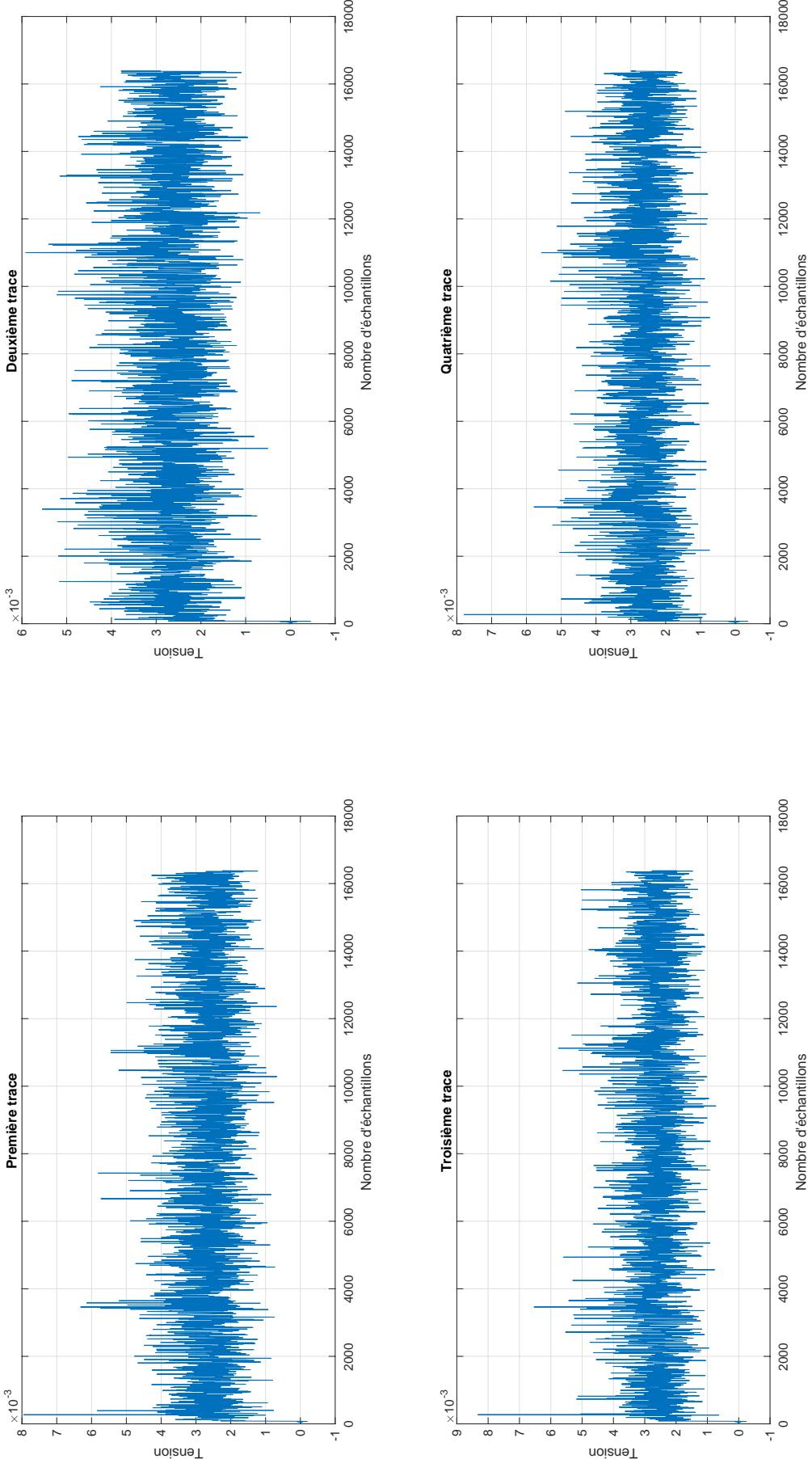
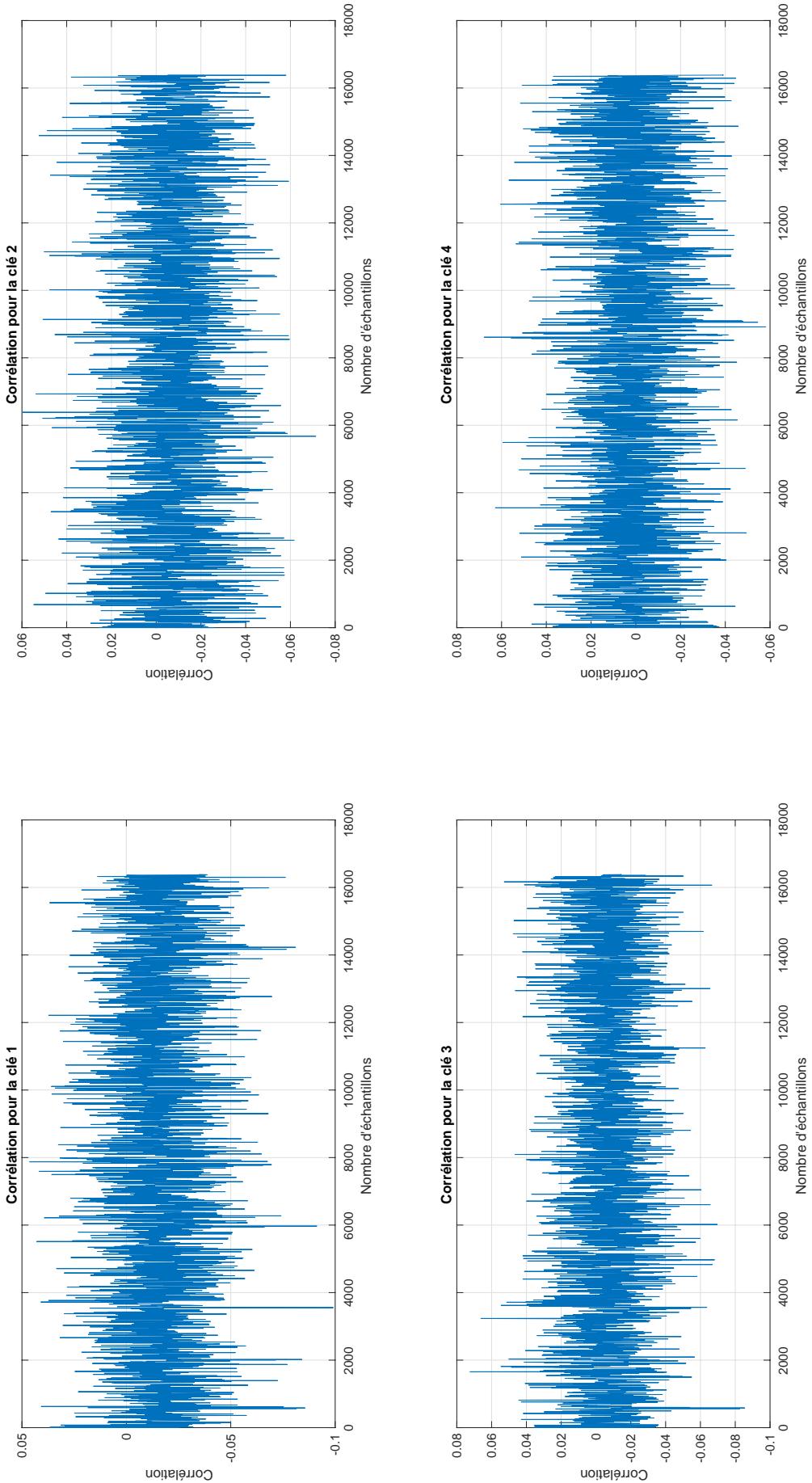
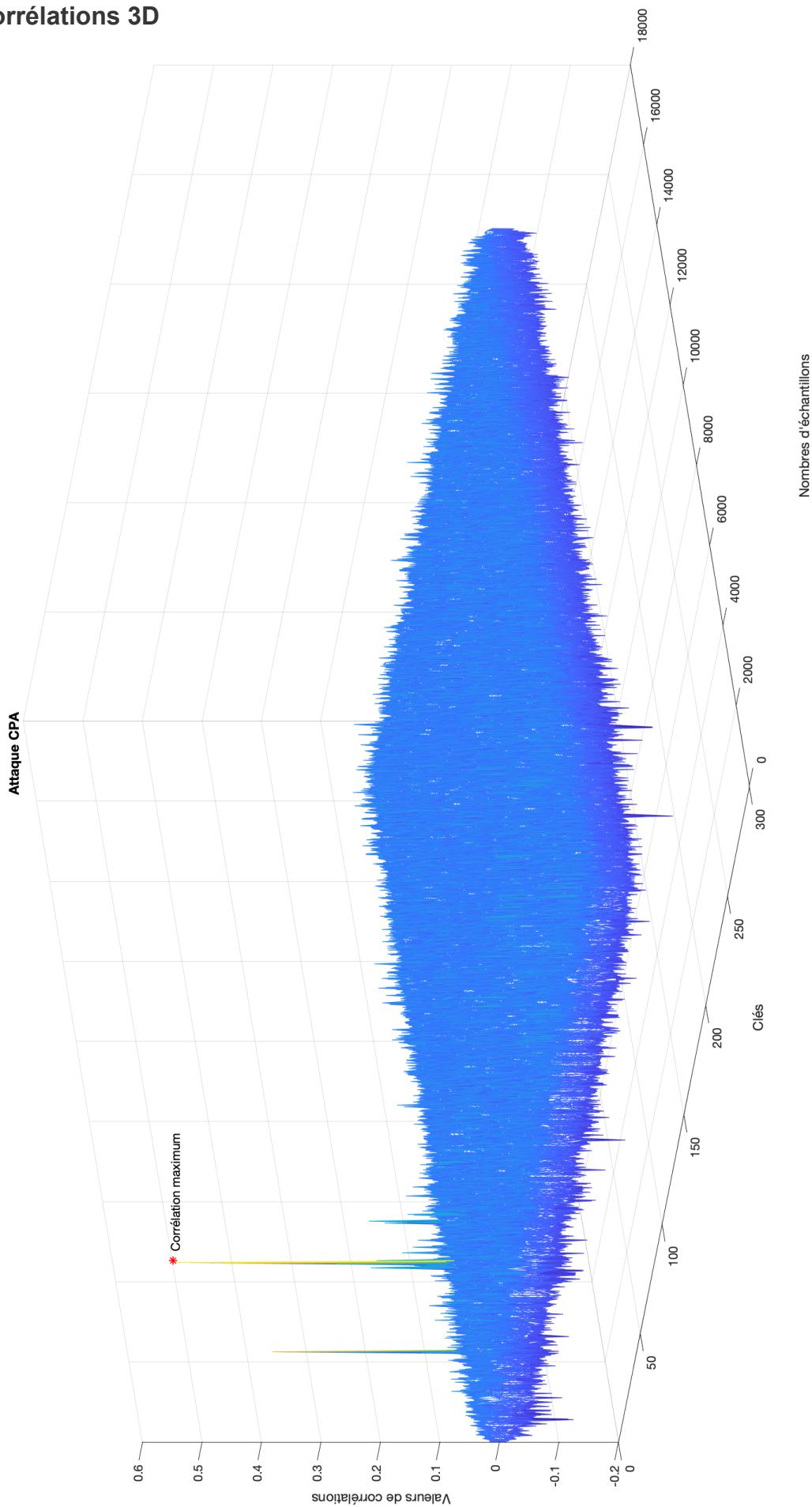


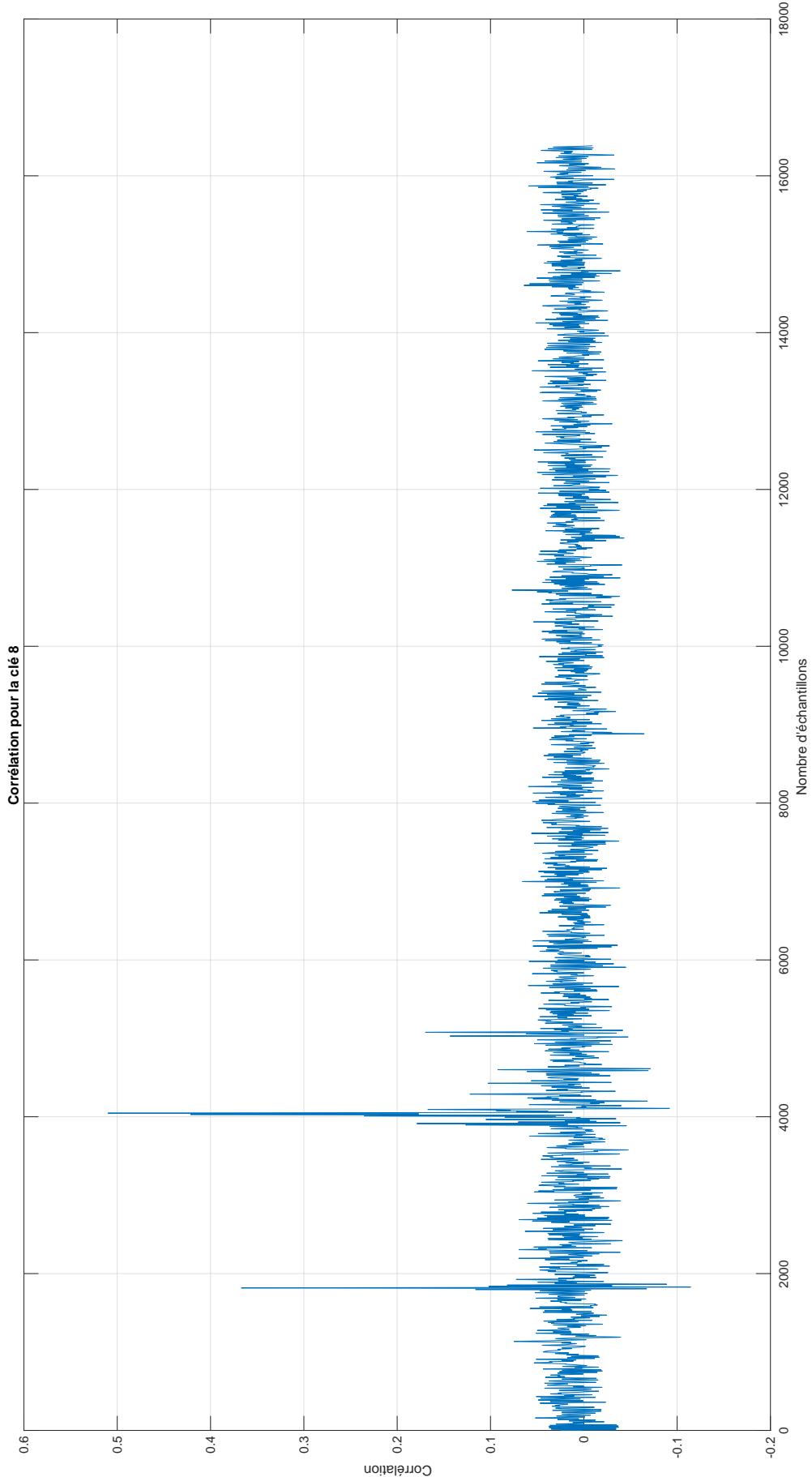
Figure 33 : Principe de fonctionnement de l'opération KeySchedule.

D Traces

E Corrélation pour les clés 1 à 4



F Corrélations 3D

G Corrélation pour la clé 8

H Contre-mesures *Hiding*

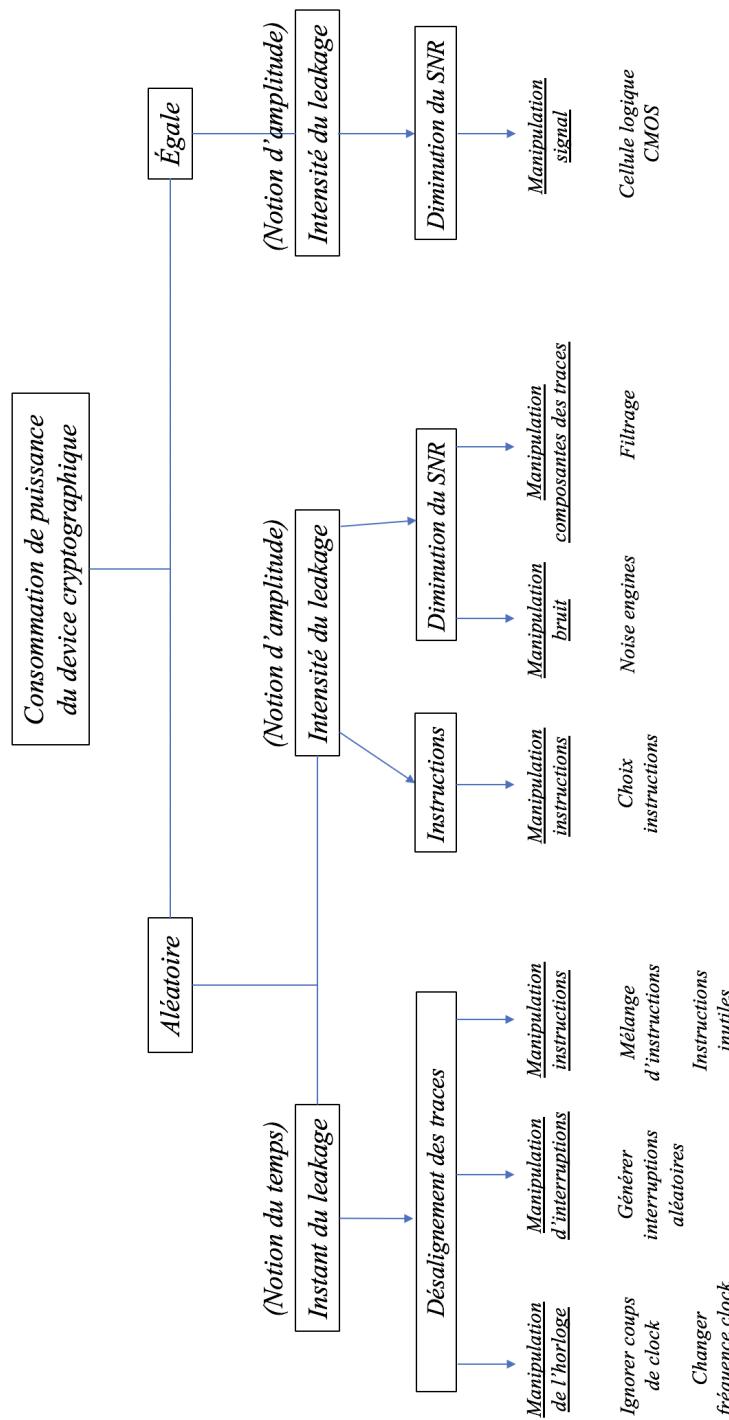


Figure 34 : Ce schéma présente les différentes sortes de contre-mesures de types hiding.

I Contre-mesure *Faking*

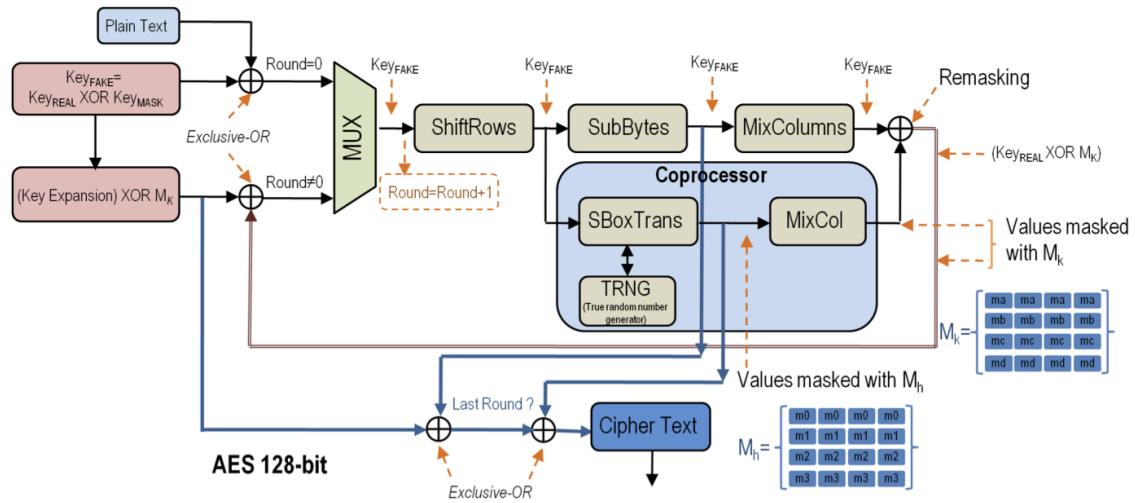


Figure 35 : Ce schéma-bloc présente le principe de fonctionnement de la contre-mesure de type Faking. On remarque que la clé manipulée par l'algorithme AES (pour chaque round exécuté) est une fausse clé (Key_{Fake}), dissimulée sous un masque (Key_{Mask}).