

Table des matières

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 6 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 management, son contexte social, son insertion économique, ses aspects techniques et ses produits. Il a également pour but de se familiariser au travail quotidien de l'ingénieur en participant 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.

1 Environnement de stage

Cette section a pour objectif de décrire l'entreprise à différents points de vue. Tout d'abord, une description de la structure de l'entreprise (dont l'équipe dans laquelle se trouve le stagiaire) et de ses objectifs est reprise. Ensuite, sont détaillées succinctement la gestion de production mais aussi la gestion marketing, la gestion financière et la gestion des ressources humaines. Enfin, un descriptif de la cellule qualité clôturera ce premier point.

1.1 Structure de l'entreprise

C'est donc chez **Thales Telecommunication Belgium** que je me suis rendu pour réaliser mon stage d'immersion en entreprise.

1.2 Objectifs de l'entreprise

1.3 Équipe du stagiaire

1.4 Gestion de production

1.5 Gestion marketing

1.6 Gestion financière

1.7 Gestion des ressources humaines

1.8 Politique QHSE

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, une définition des notions théoriques est abordée. 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. *Side-Channel Attacks* (attaques par canal auxiliaire)
4. Simulation d'un tracé et d'une attaque par canal auxiliaire sur MATLAB
5. Statistiques et t-test
6. Simulation d'un t-test sur MATLAB

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

3 Projet du stage

Cette section décrit l'ensemble des objectifs, cités à la section ??, 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 ?? ci-dessous présente le principe de fonctionnement d'un algorithme de chiffrement.

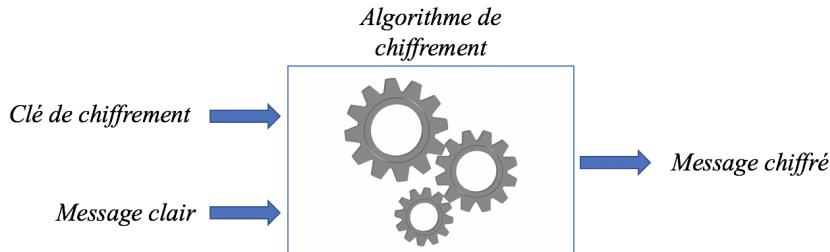


Figure 1 : L'algorithme de chiffrement, caractérisé par diverses opérations mathématiques, utilise une clé de chiffrement en entrée pour chiffrer un message clair. Cela produit un message chiffré, non compréhensible pour une personne ne connaissant pas la clé de chiffrement.

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. Une explication plus détaillée de cet algorithme est reprise à la section ??
- **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.

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

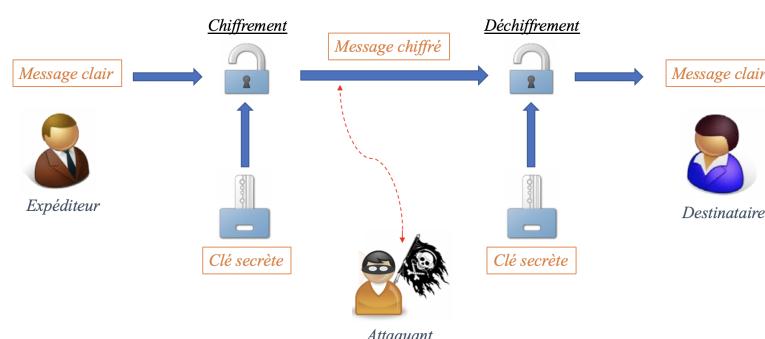


Figure 2 : Chiffrement symétrique : Une seule clé est utilisée pour chiffrer et déchiffrer les messages.

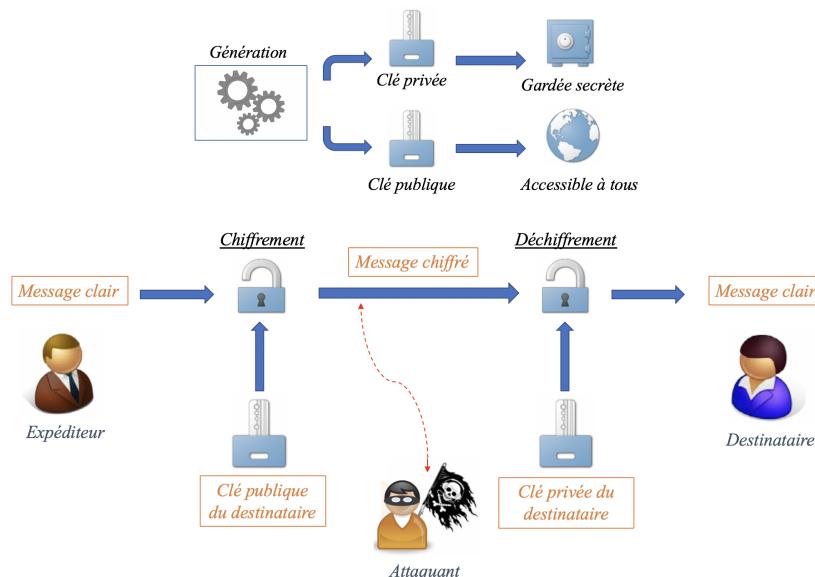


Figure 3 : Chiffrement asymétrique : Un clé publique est utilisée pour chiffrer le message et une clé privée est utilisée pour le déchiffrer.

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.

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 ??), 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 ?? 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 4 : 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 ?? (page suivante) permet de visualiser l'ordre d'exécution chronologique de ces 4 opérations.

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

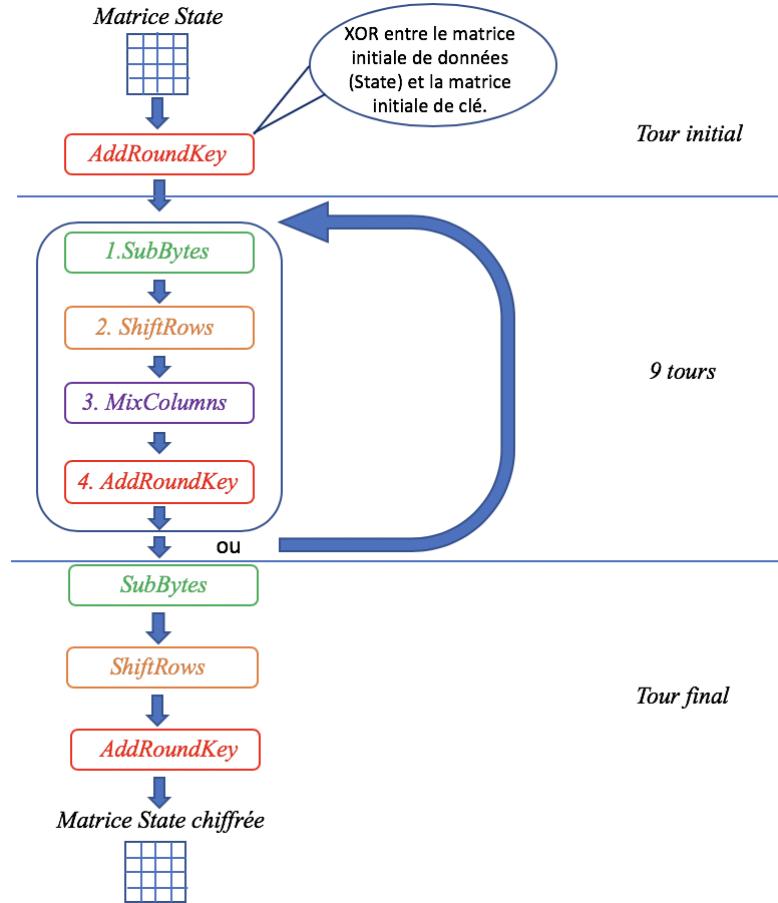


Figure 5 : Principe de fonctionnement de l'algorithme AES-128.

Fonctionnement :

Initialement, deux matrices vont être utilisées : la matrice STATE, contenant les données claires, et la matrice clé, contenant la clé secrète initiale. En effet, une première opération (*AddRoundKey*) est appliquée sur ces deux matrices. Plus précisément, cette opération réalise un XOR (symbole \oplus) entre chaque élément de la matrice STATE et chaque élément de la matrice clé. Le résultat est ré-écrit dans la matrice STATE. La figure ?? ci-dessous présente le principe de fonctionnement de cette première opération :

04	e0	48	28	⊕	a0	88	23	2a	=	a0	88	23	2a
66	cb	f8	06		fa	54	a3	6c		fa	54	a3	6c
81	19	d3	26		fe	2c	39	76		fe	2c	39	76
e5	9a	7a	4c		17	b1	39	05		17	b1	39	05

Matrice STATE *Matrice clé* *Nouvelle matrice STATE*

Figure 6 : Opération XOR entre la matrice STATE et la matrice clé.

Ensuite, une série de 4 opérations se répétant 9 fois (9 tours cycliques) est exécutée. Ces 4 opérations sont appliquées dans l'ordre suivant : *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*.

Enfin, une fois les 9 tours exécutés, le dernier *round* se lance exécutant 3 opérations : *SubBytes*, *ShiftRows* et *AddRoundKey*. À la fin de ces 3 dernières opérations, une matrice de taille 4x4 présente le message chiffré de 128 bits.

Description des 4 opérations :

- 1. SubBytes** : La figure ?? ci-dessous présente le principe de fonctionnement de l'opération *SubBytes* :

ShiftRows

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	<i>Pas de décalage</i>	$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	<i>Décalage de chaque élément de <u>1</u> position vers la gauche</i>	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	<i>Décalage de chaque élément de <u>2</u> positions vers la gauche</i>	$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	<i>Décalage de chaque élément de <u>3</u> positions vers la gauche</i>	$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Matrice STATE avant l'opération

Matrice STATE après l'opération

Figure 7 : 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 a pour indice 0. La figure ?? ci-dessous présente le principe de fonctionnement de l'opération *ShiftRows* :

ShiftRows

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	<i>Pas de décalage</i>	$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	<i>Décalage de chaque élément de <u>1</u> position vers la gauche</i>	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	<i>Décalage de chaque élément de <u>2</u> positions vers la gauche</i>	$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	<i>Décalage de chaque élément de <u>3</u> positions vers la gauche</i>	$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Matrice STATE avant l'opération

Matrice STATE après l'opération

Figure 8 : 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 ??) 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 ?? ci-dessous présente le principe de fonctionnement de l'opération *MixColumns* :

- 4. AddRoundKey** : La gestion des clés se fait au travers des fonctions addRoundKey et keyExpansion. AddRounkey réalise un XOR entre STATE et la clé courante (roundkey 8). Le résultat de ce XOR est placé dans STATE. Cette roundkey est dérivée de la clé secrète. On change de clé courante à chaque tour pour éviter d'utiliser toujours la même valeur ; cela rend la tâche d'attaquants plus difficile. KeyExpansion est là pour étendre la clé afin, comme nous l'avons dit plus haut, de ne pas utiliser la même clé à chaque tour. La taille de la clé, après avoir été étendue, est égale à la longueur d'un bloc multipliée par le nombre de tours plus un. En effet, il nous faut une clé différente par tour ; de plus, au début de l'algorithme, on exécute un addRounKey initial avant le premier tour. Pour AES-128 (10 tours et clé de 16 octets), la clé, après avoir été étendue, aura 176 octets. $16(10+1) = 176$ octets. Le fonctionnement de KeyExpansion étant le même pour la construction de chaque sous-clé, nous allons voir comment on crée la première sous-clé à

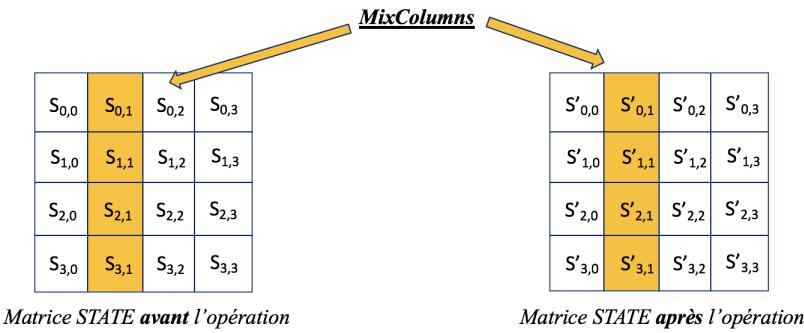


Figure 9 : Opération *MixColumns* exécutée sur la matrice STATE.

$$\begin{bmatrix} S'_{0,1} \\ S'_{1,1} \\ S'_{2,1} \\ S'_{3,1} \end{bmatrix} =
 \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,1} \\ S_{1,1} \\ S_{2,1} \\ S_{3,1} \end{bmatrix}$$

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 10 : Exemple de l'opération *MixColumns* exécutée sur la deuxième colonne de la matrice STATE.

partir de la clé de départ ; ceci pourra se généraliser pour les autres clés en remplaçant « première sous-clé » par « i-ième sous-clé » et « clé de départ » par « sous-clé précédente ».

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. 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 ?? ci-dessous résume les différents types d'attaques physiques possibles.

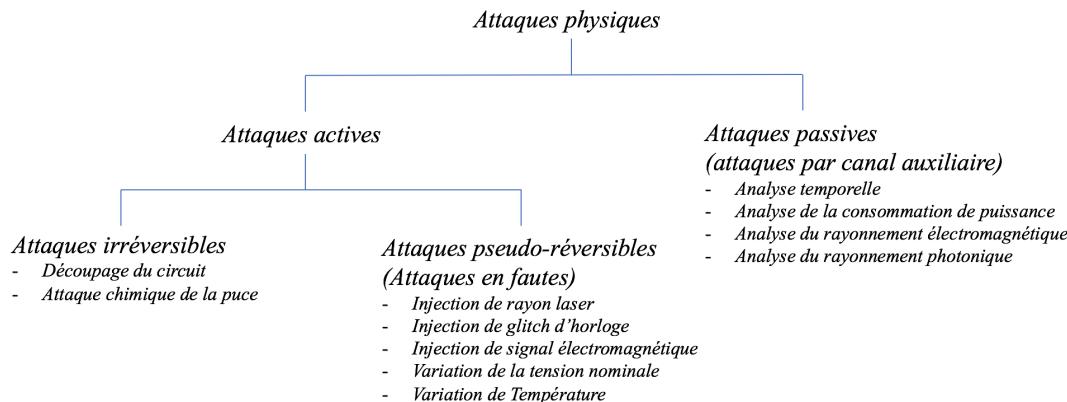


Figure 11 : Les 2 grandes familles d'attaques physiques possibles. La suite de ce rapport se concentre essentiellement sur les attaques physiques dites passives.

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 ?? ci-dessous présente les différentes façons possibles de réaliser des attaques passives.

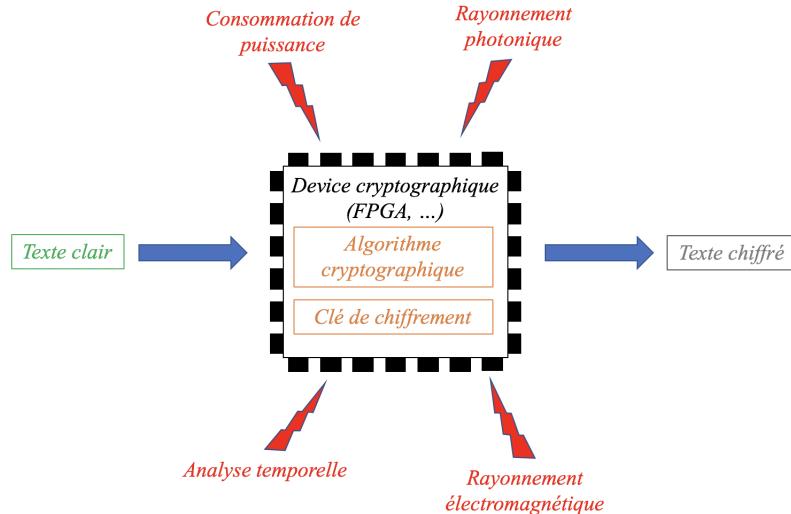


Figure 12 : Les différentes façons d'attaquer passivement un device cryptographique en vue de casser l'algorithme de chiffrement qui se trouve dessus.

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 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 comme les opérations exécutées (afin d'identifier un algorithme par exemple), les informations secrètes complètes (clé de chiffrement) ou partielles (poids de Hamming d'une 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 sera placée en série avec un canal du device cryptographique afin que l'oscilloscope soit capable de capturer une différence de tension minime. En effet, étant donné que les courants qui circulent sur le device cryptographique sont de valeurs très faibles (μ A), un amplificateur est utilisé afin d'amplifier la différence de tension.

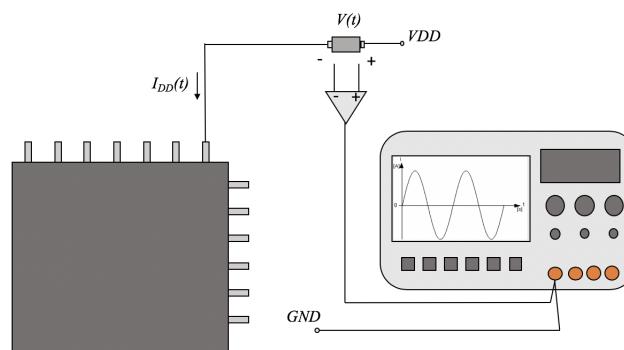


Figure 13 : Principe de mesure à l'oscilloscope.

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*)

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 ?? présente vulgairement le principe général d'une attaque CPA.

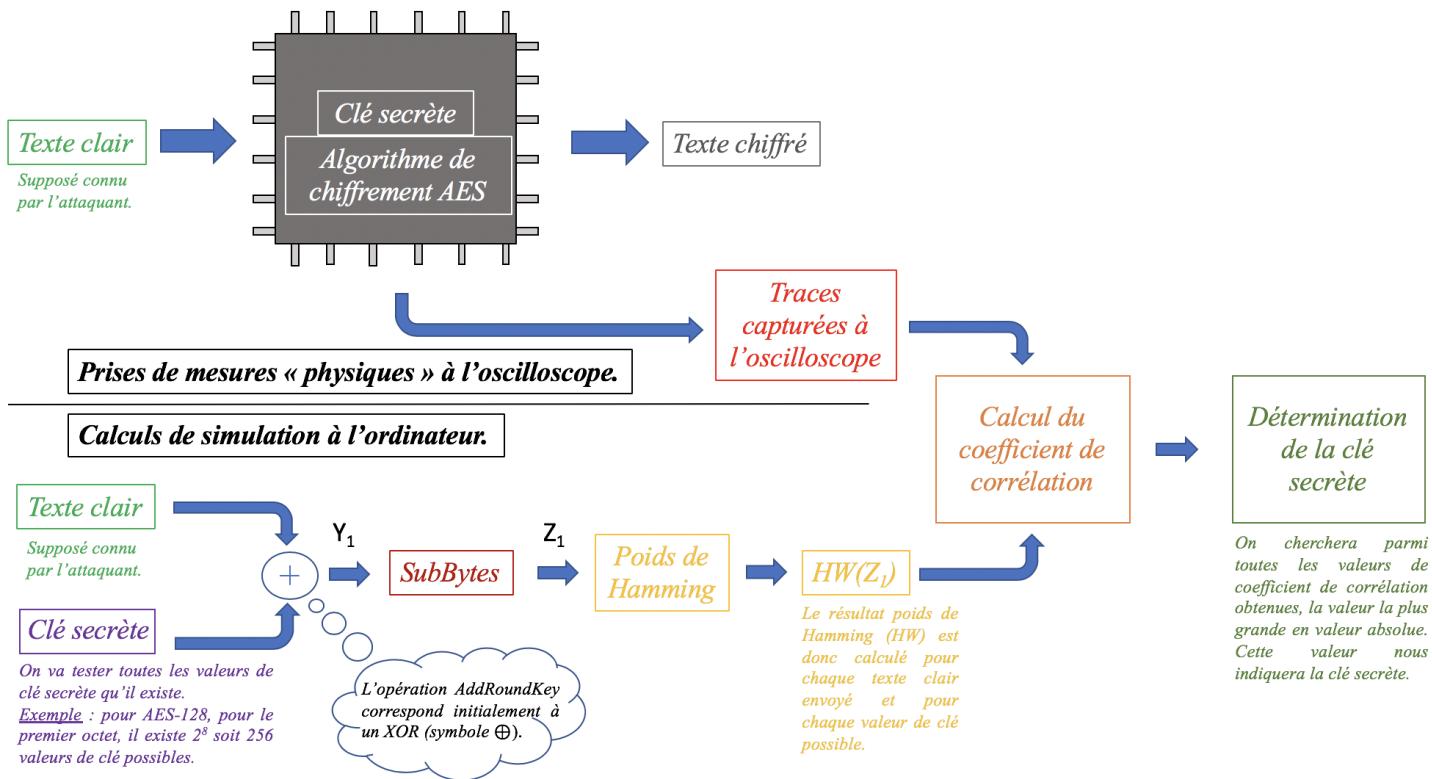


Figure 14 : Principe générale d'une attaque CPA.

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).

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

La figure ?? ci-dessous présente le schéma de l'inverseur CMOS :

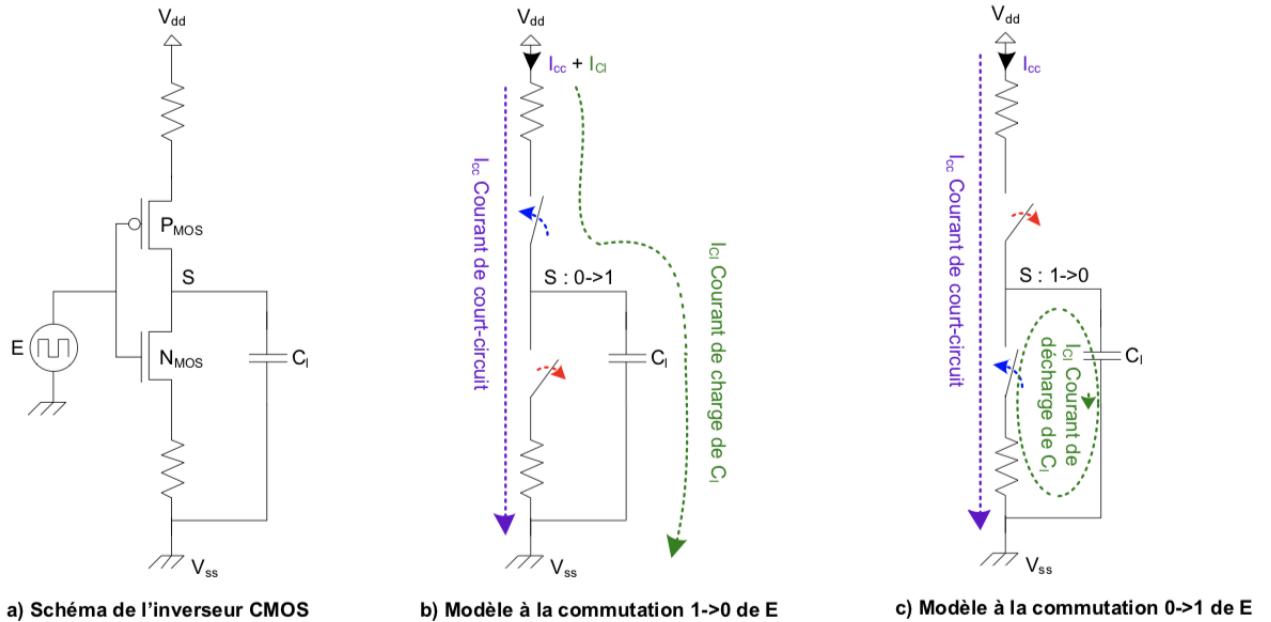


Figure 16 : Schéma de l'inverseur CMOS.

Si on applique à l'entrée un niveau haut, le transistor N est passant et le P est bloqué. On place ainsi la sortie au potentiel Vss (la masse), c'est-à-dire à l'état bas. Inversement, quand on met l'entrée à l'état bas, le transistor P est passant et le N est bloqué. La sortie est donc à l'état haut. On a donc bien réalisé une fonction inversion.

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 ?? 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 ??), 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 :

$$P_{total} = P_{stat} + P_{dyn} \quad (1)$$

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 (??), 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 :

$$P_{stat} = I_{fuite} \cdot V_{DD} \quad (2)$$

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 ?? 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 17 : 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 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 ?? présente le schéma de l'inverseur CMOS lorsqu'il y a une commutation de sa sortie de l'état 0 à l'état 1 (partie gauche du schéma) et lorsqu'il y a une commutation de sa sortie de l'état 1 à l'état 0 (partie droite du schéma). 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 (??) :

$$P_{chrg} = \frac{1}{T} \int_0^T p_{chrg}(t) dt = \alpha \cdot f \cdot C_L \cdot V_{DD}^2 \quad (3)$$

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 (??) :

$$P_{cc} = \frac{1}{T} \int_0^T p_{cc}(t) dt = \alpha \cdot f \cdot C_L \cdot V_{DD} \cdot I_{fuite} \cdot t_{cc} \quad (4)$$

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 existe.

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 : $P_{total} = P_{op} + P_{data} + P_{noise}$.

3.4.4 Modèles de puissance

Distance de Hamming...

Poids de Hamming ...

3.4.5 L'attaque CPA en concret

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

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. 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 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 ?? 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.

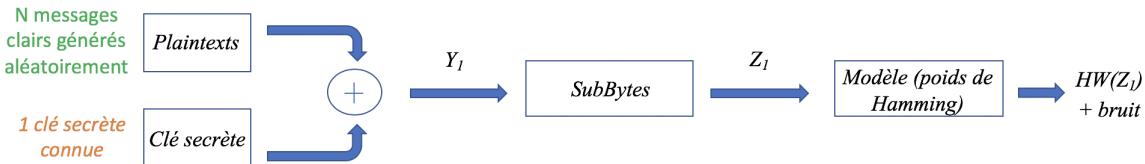


Figure 18 : Schéma-bloc permettant de comprendre la simulation d'un point d'une trace.

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 ?? 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.

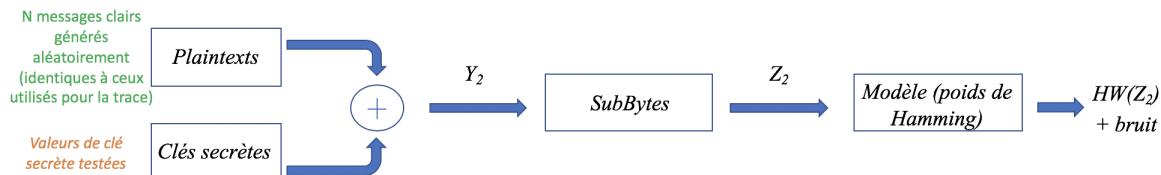


Figure 19 : Schéma-bloc permettant de comprendre la seconde phase de l'exercice 1.

La figure ?? est quasi identique à la figure ?? . 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 ?? 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 ?? présente les différentes tailles de matrices utilisées pour réaliser la simulation.

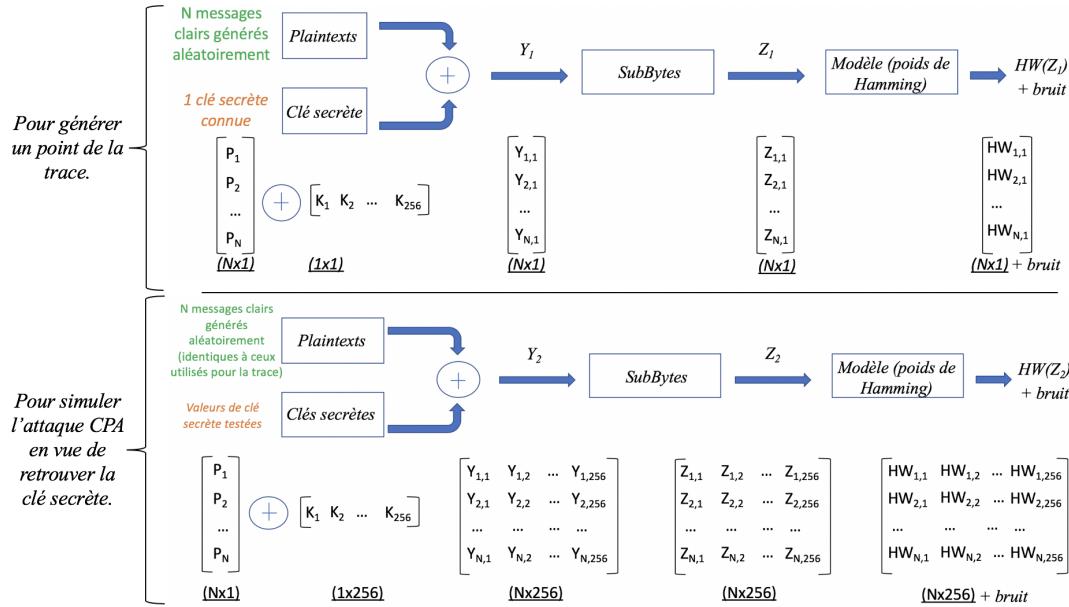


Figure 20 : Schéma-bloc permettant de visualiser la taille des différentes matrices employées pour la simulation.

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 tailler 4×4 . 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.
- 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 ?? ci-dessous présente le calcul final qui permettra de retrouver le premier octet de la clé secrète.

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

Figure 21 : Calcul du coefficient de corrélation entre un point d'une trace simulée et le poids de Hamming.

Les graphes ??, ?? et ?? ci-dessous présentent les résultats obtenus. Il s'agit de graphes indiquant les valeurs de coefficient de corrélation en fonction des 256 valeurs de clé testées. 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 nombres 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 grand, au plus l'intervalle des valeurs de corrélation sera petit 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 -0,83 et 0,8 ; pour 100 traces, le coefficient de corrélation varie entre -0,26 et 0,34 ; pour 1000 traces, le coefficient de corrélation varie entre -0,24 et 0,21. Pour chacun des trois graphes, on peut néanmoins distinguer la valeur de la clé secrète dont la valeur du coefficient de corrélation culmine à 0,98. Dans l'exemple, la clé secrète qui était utilisée valait 200 (en décimal).

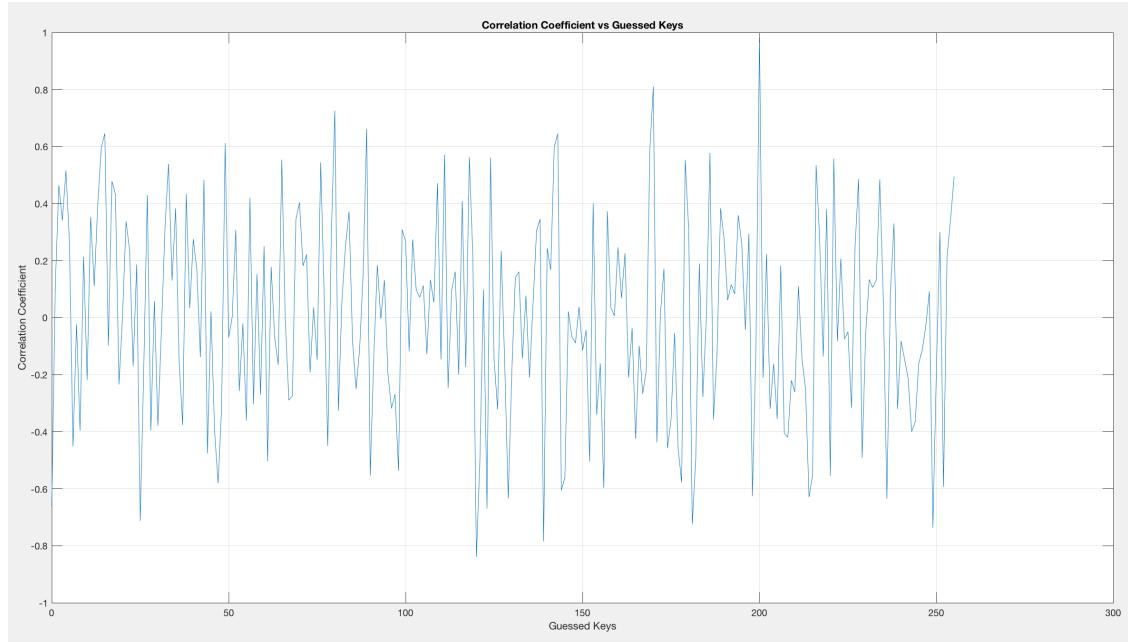


Figure 22 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 10 traces.

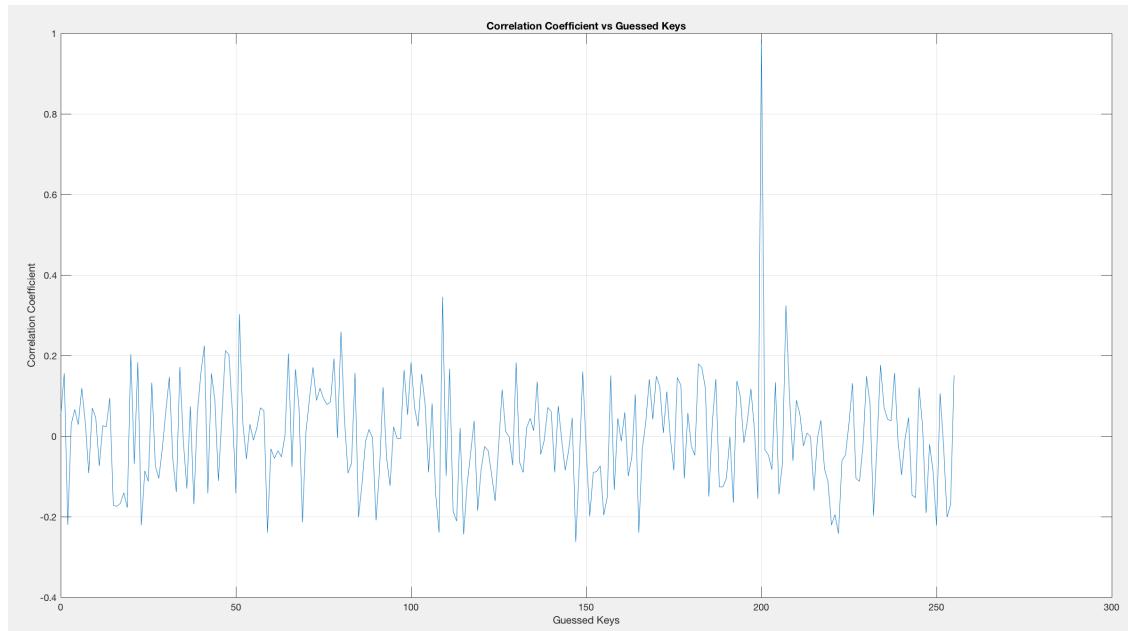


Figure 23 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 100 traces.

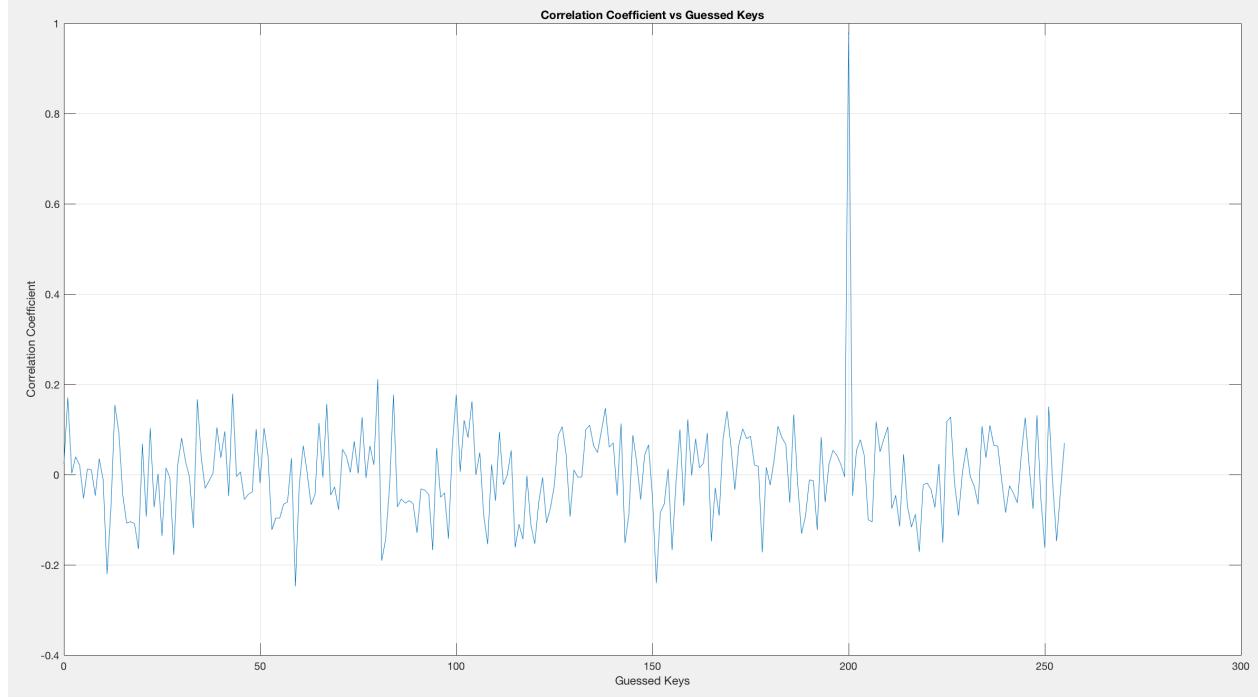


Figure 24 : Coefficient de corrélation en fonction de la valeur de la clé lorsqu'on analyse 1000 traces.

Exercice 2 : La figure ?? 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.

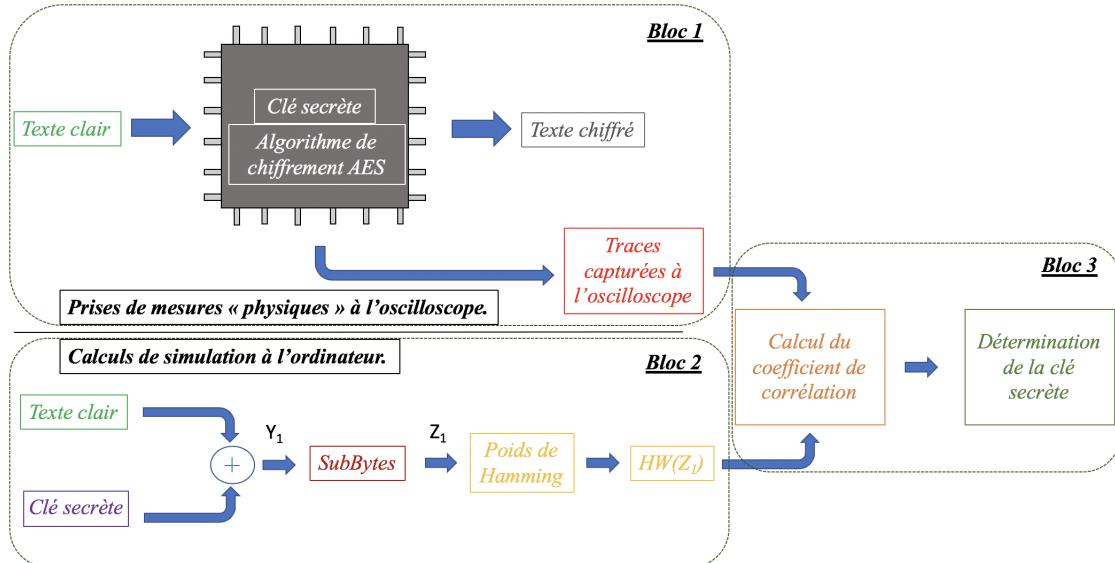


Figure 25 : Schéma-bloc permettant de comprendre le principe de fonctionnement du deuxième exercice.

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 ??.
- Dernièrement, 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 enregistrées à l'oscilloscope. Il y en a 2380.
- On connaît le nombre d'échantillons pris dans une seule 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. La figure ?? présente les quatre premières traces, parmi les 2380, mesurées à l'oscilloscope.
2. La figure ?? 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. La figure ?? 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. La figure ?? 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.

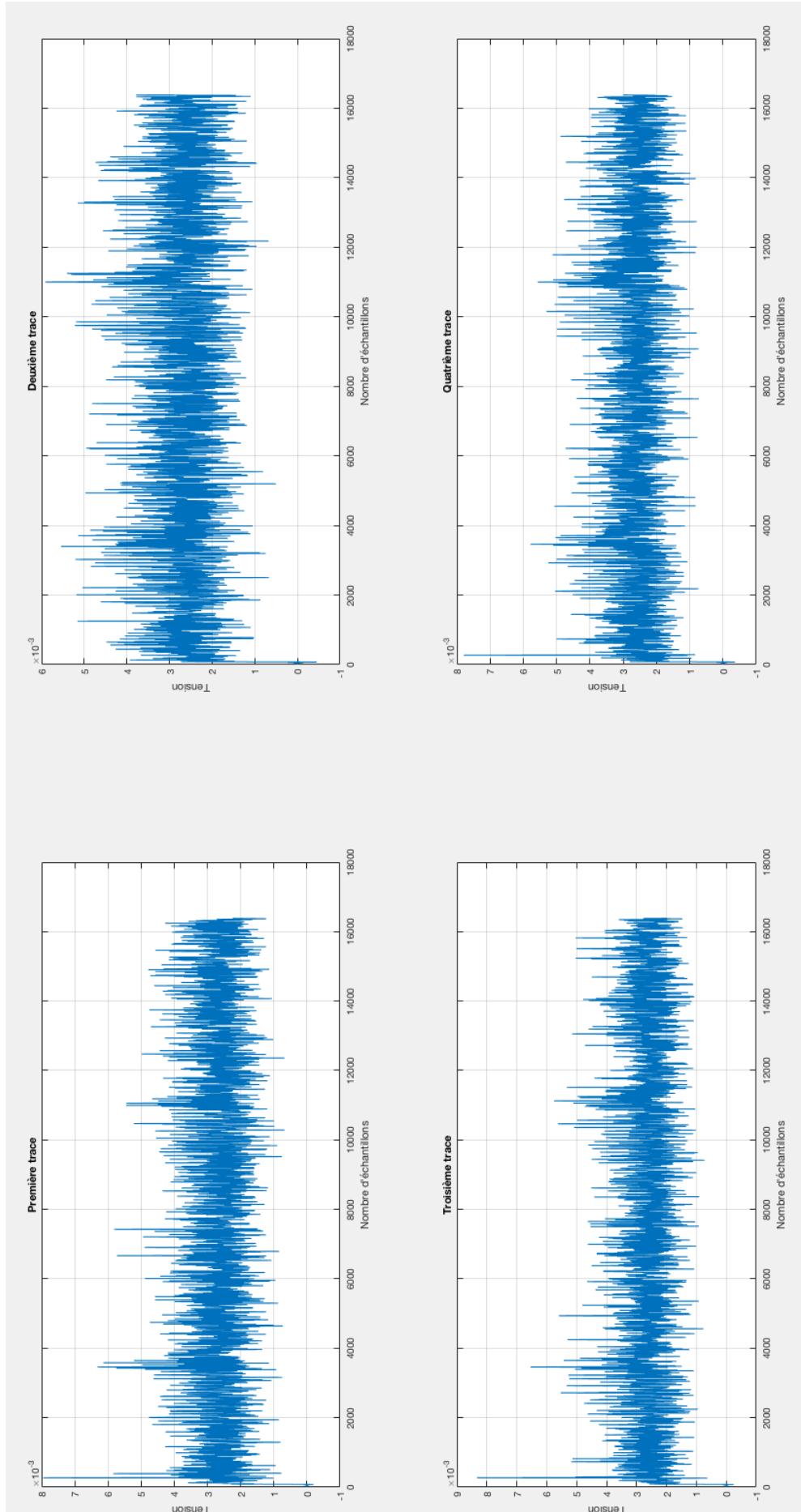


Figure 26 : Les 4 premières traces mesurées à l'oscilloscope.

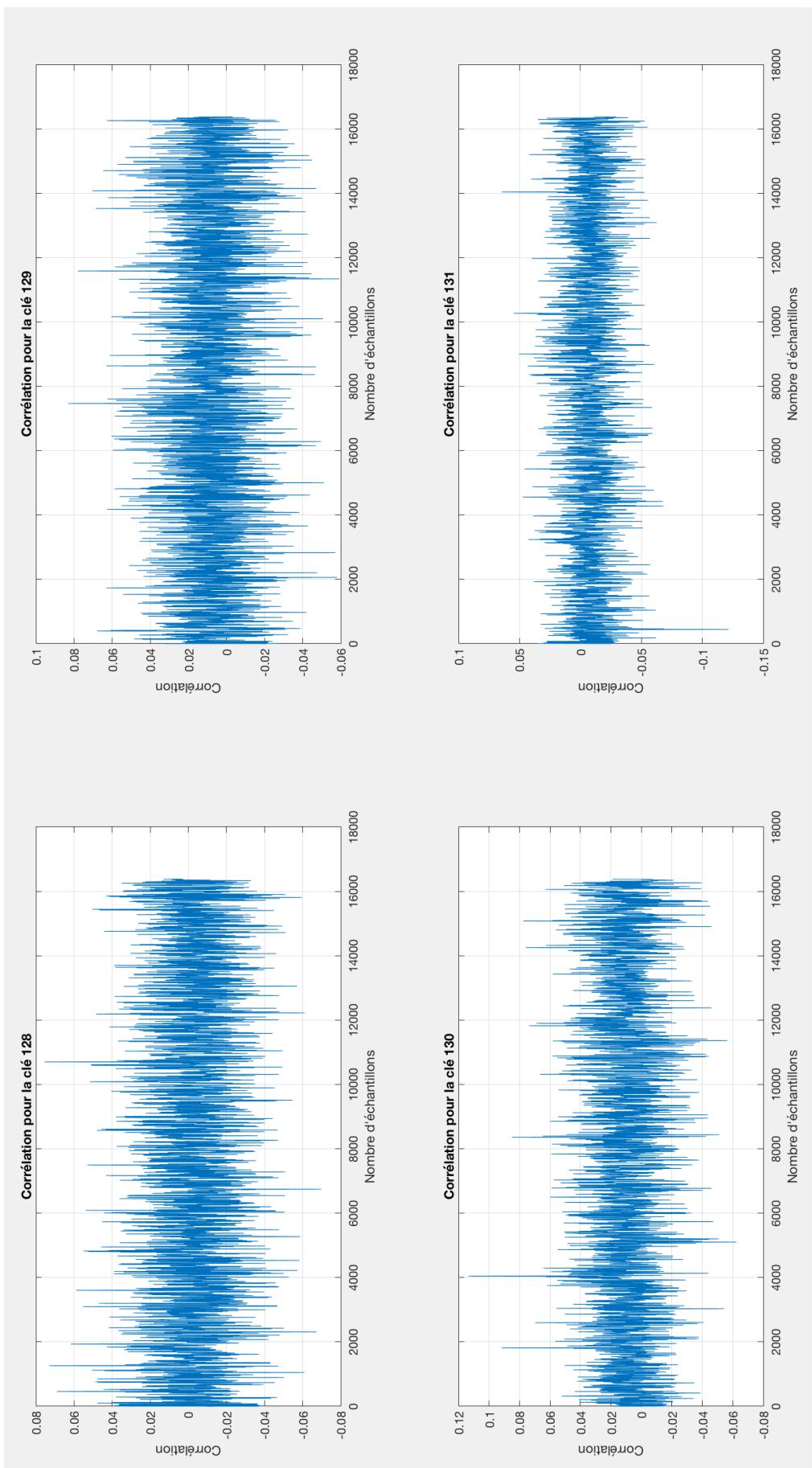


Figure 27 : Les valeurs de coefficient de corrélation obtenues pour les valeurs de clé : 128, 129, 130 et 131 respectivement.

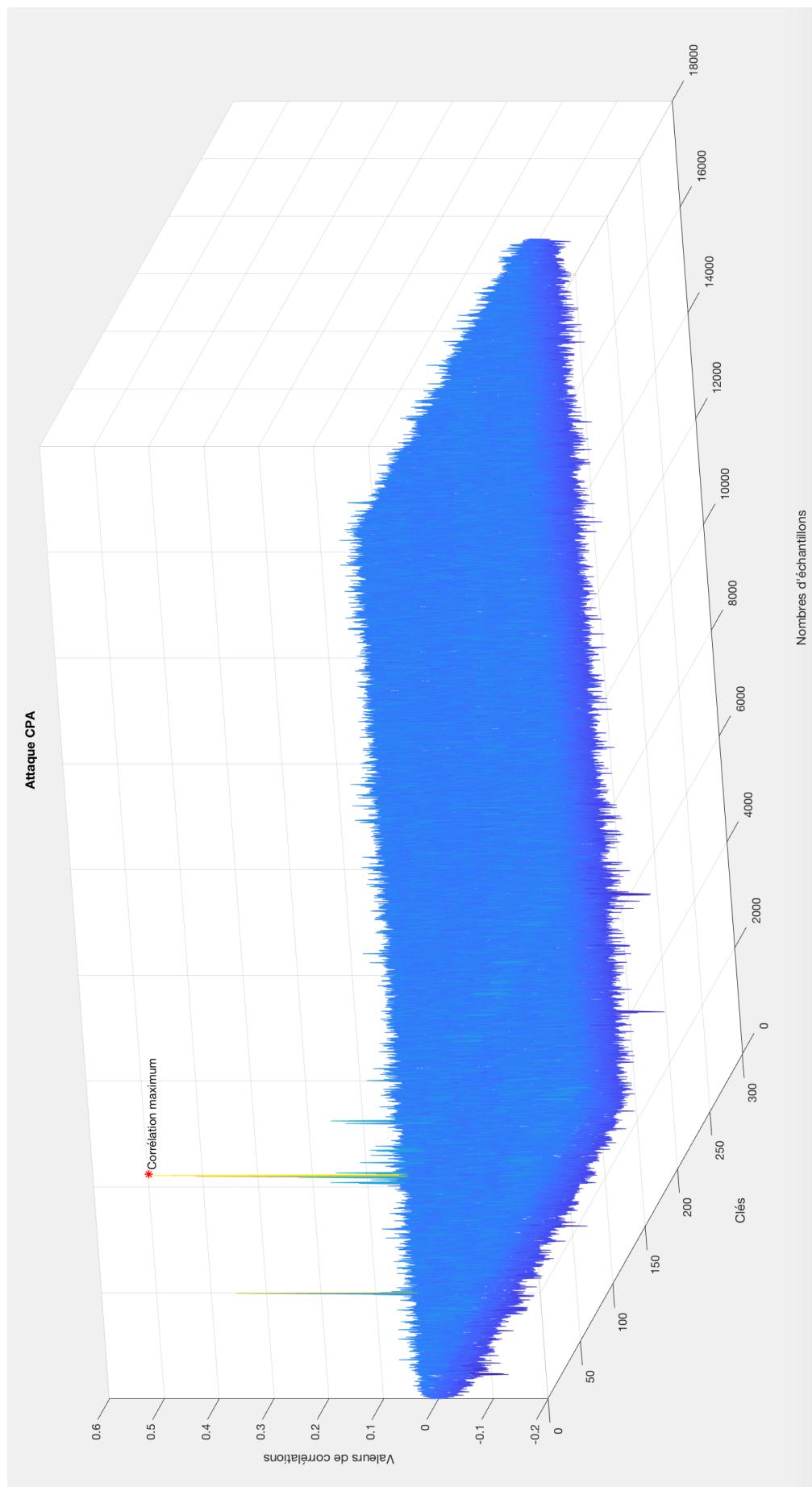


Figure 28 : Affichage en 3D de toutes les valeurs de corrélation obtenues pour chaque valeur de clé possible.

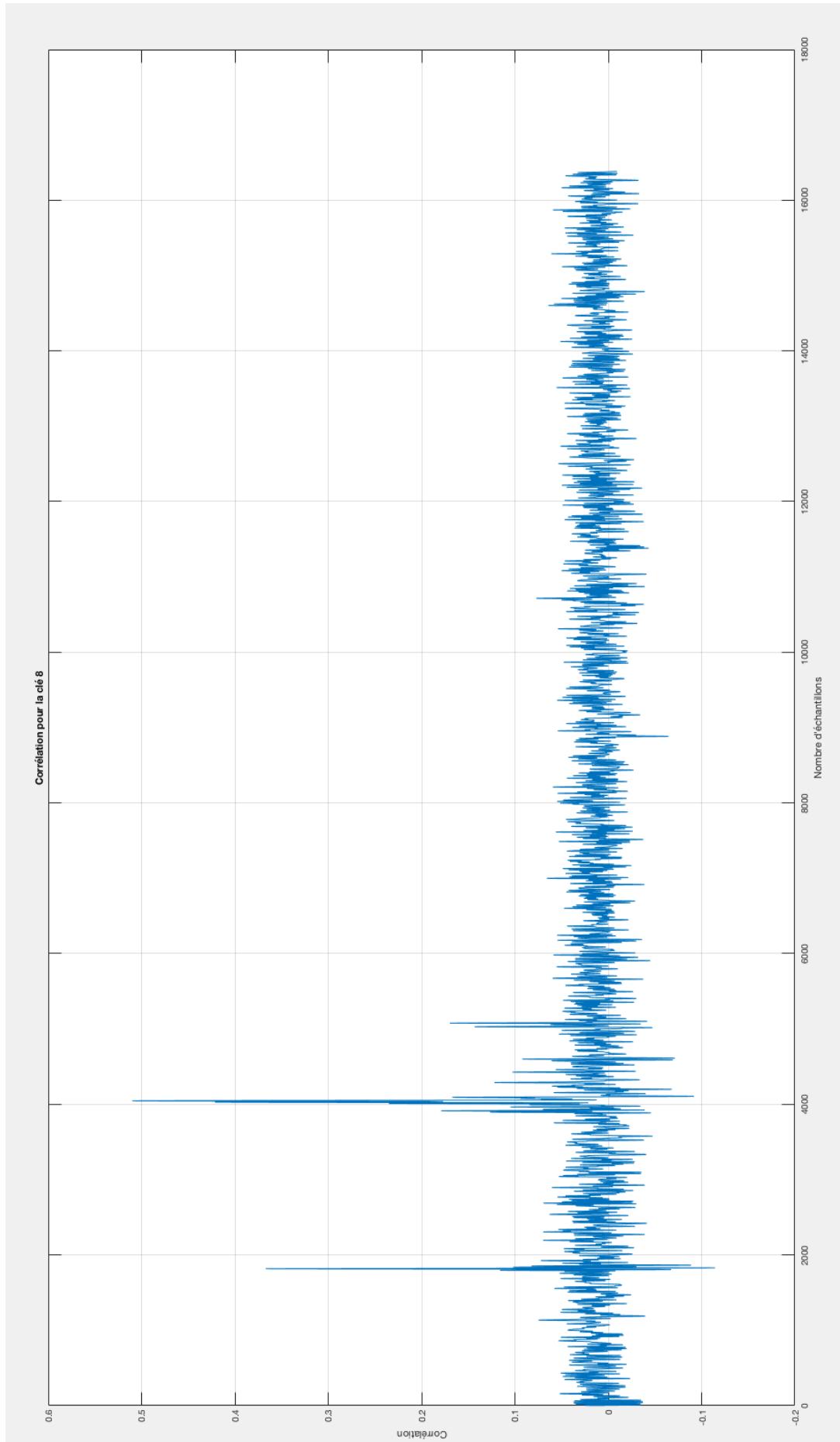


Figure 29 : Valeurs de corrélation obtenue pour la clé secrète 8 (bonne clé).

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 ??) 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 vulgairement caractérisées par des traces en tension. 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 (??). Elle peut également être exprimée en dB (??) :

$$SNR = \frac{P_{signal}}{P_{noise}} \quad (5)$$

$$SNR_{dB} = 10 \cdot \log_{10} \frac{P_{signal}}{P_{noise}} \quad (6)$$

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 (??), 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.

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.
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.* Comme dit précédemment, l'idéal serait 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.

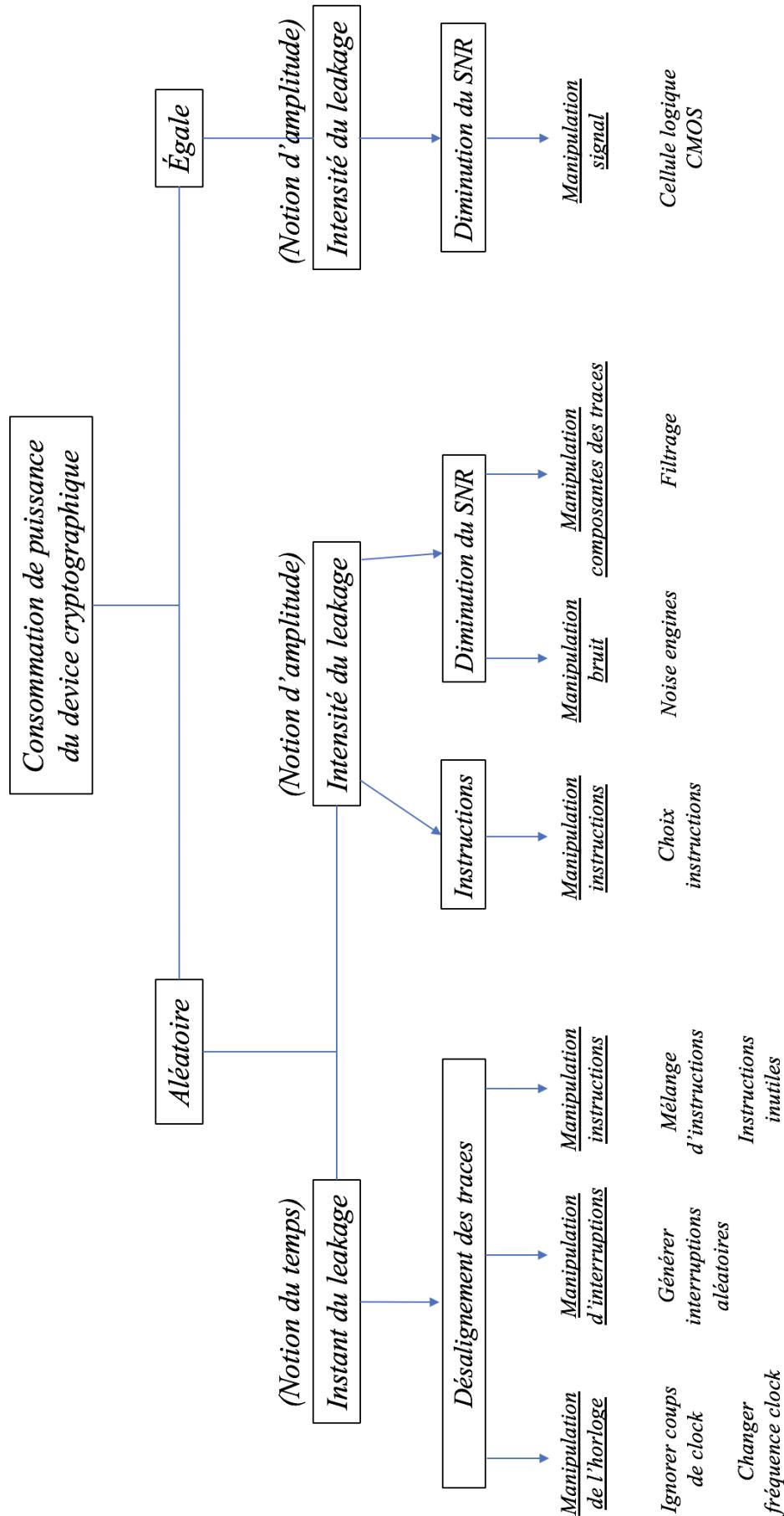
La figure ?? ci-dessous présente les mesures possibles selon l'approche suivie tandis que la figure ?? page suivante reprend l'ensemble des contre-mesures de type hiding.

<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 30 : Les contre-mesures de type hiding sont utilisées pour rendre aléatoire ou égale la consommation de puissance du device cryptographique.

Sur base de la figure ??, 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.

**Figure 31 :** Ce schéma présente une partie des contre-mesures de types hiding possibles.

- 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 ??) 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.

3.6.3 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 message chiffré.

L'opération $*$ représente les opérations exécutées par l'algorithme de chiffrement. 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.

3.6.4 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.

Par exemple, dans l'algorithme AES, on retrouve l'opération XOR. Dans ce cas, l'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.

Un contre-exemple, dans l'algorithme AES, concerne l'opération *SubBytes*. Il s'agit d'une opération non-linéaire. Autrement dit, $f(x \oplus m) \neq f(x) \oplus f(m)$. On remarque donc qu'il n'est pas facile de calculer un masque booléen pour une fonction non-linéaire. Dans ce cas, une alternative est de calculer un masque arithmétique. En effet, la relation $f(x \times m) = (x \times m)^{-1} = f(x) \times f(m)$ est respectée.

4 Conclusion

Crédits

- Tableau ?? provenant de :
Le blog officiel de lemakistatheux sur le test de Welch : <https://lemakistatheux.files.wordpress.com/2013/05/add5.png>
- Figure ?? provenant de :
Par Yves-Laurent (Travail personnel) [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>)], de Wikimedia Commons
- Figure ?? provenant de :
Par Inductiveload (Travail personnel) [Public domain], de Wikimedia Commons
- Figure ?? provenant de :
Par Toriicelli (Travail personnel) [Public domain], de Wikimedia Commons
- Figure ?? provenant de :
Par Daniel Braun [GFDL (<http://www.gnu.org/copyleft/fdl.html>), CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>) ou CC BY 2.5 (<https://creativecommons.org/licenses/by/2.5/>)], de Wikimedia Commons

Références

- https://www.emse.fr/~dutertre/documents/synth_AES128.pdf
- <https://www.j3ea.org/articles/j3ea/pdf/2012/01/j3ea12004.pdf>
- Attaques par canaux auxiliaires : nouvelles attaques, contre-mesures et mises en oeuvre (thèse)
- From Improved Leakage Detection to the Detection of Points of Interests in Leakage Traces
- <https://support.minitab.com/fr-fr/minitab/18/help-and-how-to/statistics/basic-statistics/supporting-topics/basics/type-i-and-type-ii-error/>
- <https://www.j3ea.org/articles/j3ea/pdf/2012/01/j3ea12004.pdf>
- https://www.emse.fr/~dutertre/documents/synth_AES128.pdf
- <http://www.sthda.com/french/wiki/test-t-de-welch>
- <https://hal.archives-ouvertes.fr/hal-00753215/document>
 - A voir :
 - <https://tel.archives-ouvertes.fr/tel-00937136/document>
 - http://pagesped.cahuntsic.ca/sc_sociales/psy/methosite/consignes/testtt.htm
 - http://www.pifo.uvsq.fr/epideao/esp/chap_2/hypothse_nulle.html
 - <http://www.ta-formation.com/acrobat-cours/bruit.pdf>
 - <http://www.sthda.com/french/wiki/table-de-student-ou-table-t>
 - <http://www.sthda.com/french/wiki/test-t-de-welch>
 - https://icwww.epfl.ch/~chappeli/it/courseFR/I2subsec_Hdist.php
 - <http://www.montefiore.ulg.ac.be/~dumont/pdf/crypto09-10.pdf>
 - <https://perso.uclouvain.be/fstandae/PUBLIS/196.pdf>
 - <https://help.xlstat.com/customer/fr/portal/articles/2062454quelle-est-la-différence-entre-un-test-bilatéral-et-un-test-unilatéral>
 - b_id=9283
 - <http://homepages.ulb.ac.be/~cverhoev/BMOL-G4400/chapitre1.pdf>

A Table de Welch (ou table t)

α (bilatéral)	50 %	60 %	70 %	80 %	90 %	95 %	98 %	99 %	99,5 %	99,8 %	99,9 %
$1 - \gamma$ (unilatéral)	75 %	80 %	85 %	90 %	95 %	97,5 %	99 %	99,5 %	99,75 %	99,9 %	99,95 %
<i>k</i>											
1	1,000	1,376	1,963	3,078	6,314	12,71	31,82	63,66	127,3	318,3	636,6
2	0,816	1,061	1,386	1,886	2,920	4,303	6,965	9,925	14,09	22,33	31,60
3	0,765	0,978	1,250	1,638	2,353	3,182	4,541	5,841	7,453	10,21	12,92
4	0,741	0,941	1,190	1,533	2,132	2,776	3,747	4,604	5,598	7,173	8,610
5	0,727	0,920	1,156	1,476	2,015	2,571	3,365	4,032	4,773	5,893	6,869
6	0,718	0,906	1,134	1,440	1,943	2,447	3,143	3,707	4,317	5,208	5,959
7	0,711	0,896	1,119	1,415	1,895	2,365	2,998	3,499	4,029	4,785	5,408
8	0,706	0,889	1,108	1,397	1,860	2,306	2,896	3,355	3,833	4,501	5,041
9	0,703	0,883	1,100	1,383	1,833	2,262	2,821	3,250	3,690	4,297	4,781
10	0,700	0,879	1,093	1,372	1,812	2,228	2,764	3,169	3,581	4,144	4,587
11	0,697	0,876	1,088	1,363	1,796	2,201	2,718	3,106	3,497	4,025	4,437
12	0,695	0,873	1,083	1,356	1,782	2,179	2,681	3,055	3,428	3,930	4,318
13	0,694	0,870	1,079	1,350	1,771	2,160	2,650	3,012	3,372	3,852	4,221
14	0,692	0,868	1,076	1,345	1,761	2,145	2,624	2,977	3,326	3,787	4,140
15	0,691	0,866	1,074	1,341	1,753	2,131	2,602	2,947	3,286	3,733	4,073
16	0,690	0,865	1,071	1,337	1,746	2,120	2,583	2,921	3,252	3,686	4,015
17	0,689	0,863	1,069	1,333	1,740	2,110	2,567	2,898	3,222	3,646	3,965
18	0,688	0,862	1,067	1,330	1,734	2,101	2,552	2,878	3,197	3,610	3,922
19	0,688	0,861	1,066	1,328	1,729	2,093	2,539	2,861	3,174	3,579	3,883
20	0,687	0,860	1,064	1,325	1,725	2,086	2,528	2,845	3,153	3,552	3,850
21	0,686	0,859	1,063	1,323	1,721	2,080	2,518	2,831	3,135	3,527	3,819
22	0,686	0,858	1,061	1,321	1,717	2,074	2,508	2,819	3,119	3,505	3,792
23	0,685	0,858	1,060	1,319	1,714	2,069	2,500	2,807	3,104	3,485	3,767
24	0,685	0,857	1,059	1,318	1,711	2,064	2,492	2,797	3,091	3,467	3,745
25	0,684	0,856	1,058	1,316	1,708	2,060	2,485	2,787	3,078	3,450	3,725
26	0,684	0,856	1,058	1,315	1,706	2,056	2,479	2,779	3,067	3,435	3,707
27	0,684	0,855	1,057	1,314	1,703	2,052	2,473	2,771	3,057	3,421	3,690
28	0,683	0,855	1,056	1,313	1,701	2,048	2,467	2,763	3,047	3,408	3,674
29	0,683	0,854	1,055	1,311	1,699	2,045	2,462	2,756	3,038	3,396	3,659
30	0,683	0,854	1,055	1,310	1,697	2,042	2,457	2,750	3,030	3,385	3,646
40	0,681	0,851	1,050	1,303	1,684	2,021	2,423	2,704	2,971	3,307	3,551
50	0,679	0,849	1,047	1,299	1,676	2,009	2,403	2,678	2,937	3,261	3,496
60	0,679	0,848	1,045	1,296	1,671	2,000	2,390	2,660	2,915	3,232	3,460
80	0,678	0,846	1,043	1,292	1,664	1,990	2,374	2,639	2,887	3,195	3,416
100	0,677	0,845	1,042	1,290	1,660	1,984	2,364	2,626	2,871	3,174	3,390
120	0,677	0,845	1,041	1,289	1,658	1,980	2,358	2,617	2,860	3,160	3,373
∞	0,674	0,842	1,036	1,282	1,645	1,960	2,326	2,576	2,807	3,090	3,291

Figure 32 : La table de Welch donne la probabilité α pour que t égale ou dépasse, en valeur absolue, une valeur donnée, en fonction du nombre de degrés de liberté (d.d.l)..