UNIVERSITÉ LIBRE DE BRUXELLES
FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

Cryptanalyse par analyse de consommation : une approche basée sur l'apprentissage automatique

Liran Lerman

Promoteur :
Prof. Gianluca Bontempi &
Prof. Olivier Markowitch

Mémoire présenté en vue de
l'obtention du grade de
Master en Sciences Informatiques

Année académique 2009 - 2010

Table des matières

Remerciements

A travers un stage, des lectures d'articles scientifiques, des mémoires et des thèses, ce mémoire a été plus qu'une simple recherche d'informations. Il fut aussi une période stimulante, motivante et satisfaisante grâce aux différentes découvertes réalisées. Les résultats de ce mémoire n'auraient pas pu arriver à ce succès sans la participation de plusieurs personnes, chacun ayant donné des conseils dans son domaine, telles que :

M. Bontempi G. qui m'a conseillé dans le domaine de l'apprentissage automatique

M. Markowitch O. qui m'a conseillé dans le domaine de la cryptologie

M. Demaertelaere F. qui m'a permis de réaliser un stage chez Atos Worldline

M. Meuter C. qui m'a aidé à la compréhension sur le fonctionnement de la machine de chiffrement et m'a permis la réalisation du stage chez Atos Worldline

M. Fernandes Medeiros S. qui m'a apporté l'ensemble des résultats obtenus durant son stage et qui m'a permis de gagner un temps conséquent pour l'ensemble du mémoire

Et toutes les autres personnes du groupes d'Atos Worldline me donnant divers conseils tels que : M. Beaudoint S., M. Lheureux D., M. Van Begin L. et M. Nyssen B.

Chapitre 1

Introduction

1.1 Présentation du problème

Depuis l'Antiquité, et dans la plupart des civilisations, la cryptographie est utilisée pour protéger les messages de façon à assurer leur intégrité, leur confidentialité, leur disponibilité et leur authenticité, et ce pour leur stockage, leur traitement ou leur transmission. L'intégrité est un contrôle sur la modification possible d'un document; la confidentialité, sur l'accès à l'information; la disponibilité, sur l'obtention de la donnée et l'authenticité, sur la source de l'information.

Il est commode d'appeler l'émetteur par Alice, le destinataire par Bob et l'espion par Eve. La question qui se pose est comment Alice peut envoyer un message à Bob tout en garantissant qu'Eve ne puisse le lire ou le modifier sans qu'Alice et Bob s'en aperçoivent ?

Comme le souligne [53], nous pouvons remonter jusqu'à la période de l'ancienne Egypte pour trouver des traces de la cryptographie. En ces temps, seules certaines personnes avaient la connaissance suffisante pour lire et écrire des messages. Du point de vue cryptographique, cette connaissance peut être vue comme la clé qui amène à la compréhension des messages. Au fil du temps, les techniques se sont complexifiées, passant des algorithmes de chiffrement rudimentaires (chiffre de César, chiffre de Vigenère,...) à des algorithmes de cryptographie moderne. Ces techniques sont le fruit d'une demande de plus en plus poussée pour chiffrer des données, et donc pour les cacher.

La demande de la cryptographie s'est développée en parallèle d'un autre souhait : la cryptanalyse permettant de déchiffrer des messages cachés de manière détournée. Ce principe est similaire à celui des sociologues qui essayent de comprendre, de déchiffrer et d'inférer les gestes des autres. Dans notre cas, ce n'est pas la compréhension de l'Homme qui nous intéresse mais certaines données encodées dans une machine. Outre l'envie de découvrir des informations cachées, le but de la cryptanalyse est d'améliorer la cryptographie en pointant les dangers.

Bien que les algorithmes contemporains soient résistants à certaines attaques, au fil du temps cette robustesse se casse en trouvant des techniques mathématiques permettant d'accélérer le calcul d'une opération. Nous pouvons dire que la cryptographie classique ne s'adapte pas à la technologie contemporaine. En d'autres mots, elle ne résiste pas à la decouverte d'une nouvelle technique d'attaque contre elle-même. Une des solutions à cela est l'utilisation de la cryptographie quantique [1], qui se base sur l'unique algorithme de chiffrement incassable mathématiquement. En d'autres termes, en ne connaissant que le texte chiffré, et en supposant que toutes les clés possibles sont équiprobables, ont la même taille que le texte chiffré, et sont enfin utilisées qu'une seule fois, alors nous ne pouvons pas retrouver théoriquement le message en clair que si nous possédons la clé. Néanmoins, cette technique est encore très complexe et coûteuse à mettre en œuvre, c'est pourquoi nous utilisons la plupart du temps les algorithmes cités précédemment.

De surcroît, le temps n'est pas le seul paramètre de la robustesse d'un algorithme. En effet, bien que ces algorithmes soient résistants en théorie, nous remarquons une perte de la robustesse en pratique, lorsqu'ils sont implémentés dans des cas réels. Par conséquent, depuis quelques années et de manière fascinante, les techniques d'attaques de matériels de chiffrement en n'utilisant que des informations fournies par ce dernier, appelées analyse de canaux auxiliaires, ont montré une très forte efficacité pour contrer toutes les techniques de protection de données. Ainsi, nous n'allons plus essayer de retrouver la clé ou le message en clair avec le message chiffré, mais nous allons nous focaliser sur le comportement de la machine de chiffrement, et plus précisément sur toutes les propriétés physiques du matériel. En reprenant l'analogie du sociologue, nous essayerons de comprendre ce que veut dire une personne par ses gestes, sa posture, le contexte ou son passé et non pas uniquement par les mots qu'elle prononce. Les exemples de fuite d'informations sont : le temps des opérations [2] [29], le son, l'émanation électromagnétique [22], les attaques par faute, la température du matériel et la consommation d'énergie [31]. Cette dernière sera celle sur laquelle ce mémoire se focalisera, bien que la solution générale proposée soit applicable pour les autres exemples ou pour une combinaison de ces derniers. La figure 1.1 illustre ce principe où à l'aide d'un oscilloscope qui mesure le courant, d'un FPGA [3] qui chiffre les données et d'un ordinateur qui collecte et gouverne l'ensemble, nous réalisons l'attaque. Toutefois, le domaine cité précédemment est encore trop vaste pour un mémoire.

1. Nous renvoyons les lecteurs intéressés par la cryptographie quantique au document [32], inspiré par le document [52].

2. Nous pouvons supposer qu'une multiplication dépendante de la clé aura un temps d'exécution différent selon la clé. En effet, pour une question d'optimisation, certains algorithmes chiffrent avec un temps qui varie selon la clé manipulée.

3. Field-Programmable Gate Array.

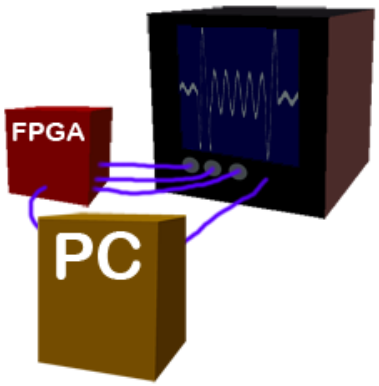


Figure 1.1 : Illustration d'un FPGA, d'un oscilloscope et d'un ordinateur, tous reliés entre eux.

Ainsi, pour cadrer ce mémoire, une seule technique d'attaque - le *template Based DPA* [4] - sera détaillée, analysée, et comparée à une nouvelle solution. Le choix de la technique détaillée vient du fait que selon [21], ce type d'attaques est le plus puissant [5], le plus dévastateur parmi l'ensemble de sa famille. Grossièrement, la technique détaillée essaye de faire le lien entre la consommation d'énergie [6] du matériel de chiffrement et la clé employée, en supposant que cette dépendance existe. Jusqu'à maintenant, ce lien se faisait en supposant que les données suivent une distribution normale multivariée [7]. Néanmoins, des techniques automatiques permettent de relaxer cette contrainte.

Le but final, et ainsi l'originalité de ce travail, est donc d'améliorer le domaine de la cryptanalyse, et plus précisément celui de l'analyse de canaux auxiliaires, en réalisant une intégration prometteuse dans un monde à la fois distinct et complémentaire de la cryptanalyse, celui de l'apprentissage automatique.

Les conséquences de cette recherche peuvent affecter un ensemble de groupes de personnes très différentes, en passant par un public très large qui utilise des techniques de chiffrement souvent sans le savoir [8], jusqu'aux gouvernements, aux militaires et aux entreprises qui comptent sur la cryptographie pour assurer, entre autre, une certaine confi-

4. Differential Power Analysis.

5. La puissance d'une attaque est le pourcentage de bonnes réponses, de bonnes estimations de la clé utilisée.

6. Nous supposons que la consommation d'énergie est influencée par le type d'opération utilisé et le type de donnée manipulé.

7. Les lecteurs voulant connaitre plus de détails sur la loi de distribution normale multivariée sont invités à lire [7].

8. Un ensemble d'appareils de la vie courante utilisent le chiffrement, tel que les cartes à puce qui cachent une clé secrète. Un exemple pratique d'attaques sur un matériel de chiffrement qui permet l'ouverture et la fermeture de la porte d'une voiture est présenté dans [19]. Ceci a comme conséquence qu'un ensemble de voitures, reposant sur la technologie KeeLog, sont non protégées, tel que ceux de Chrysler, Fiat, GM, Jaguar, etc.

dentialité des données échangées entre des ambassades ou en période de guerre, d'hostilité, de crise politico-économique. Ainsi, toute découverte en cryptanalyse a une répercussion directe sur ces acteurs.

La prochaine section traite sur la motivation qui a donnée lieu à la réalisation de cette combinaison.

1.2 Motivation du choix effectué

La meilleure technique d'attaque contre un matériel de chiffrement, à savoir le *template Based DPA*, a beaucoup d'avantages tels que

1. L'efficacité de l'attaque
2. Ne nécessite aucune connaissance sur le message chiffré
3. Ne nécessite aucune connaissance sur le message clair
4. Prend en compte l'ensemble des informations dans le comportement du matériel de chiffrement
5. Réalisable contre l'ensemble des méthodes de chiffrement

Le but de ce mémoire est de montrer qu'une nouvelle attaque contre un matériel de chiffrement est réalisable. Cette dernière se veut, à la fois, novatrice et plus puissante que la meilleure technique d'attaque connue à ce jour.

Elle est une combinaison entre deux domaines : la cryptanalyse et l'apprentissage automatique. Outre les avantages de la meilleure technique, la nouvelle méthode semble en avoir d'autres. Comme pour tout modèle non paramétrique, tel que ceux mis à disposition par le domaine de l'apprentissage automatique, nous ne devons réaliser aucune hypothèse sur la distribution de probabilité des données contrairement au *template Based DPA*. Ce relâchement d'hypothèse est un grand avantage. En effet, pour que l'attaque *template Based DPA* soit efficace, il est nécessaire que les données suivent une distribution gaussienne. Dans le cas où cette condition est satisfaite, il est connu que ce type d'attaque est meilleur que si nous utilisions des modèles non paramétriques. Toutefois, ce modèle a des limites. Bien que dans des cas extrêmes, ce modèle est applicable, nous avons observé que les données ne suivent pas précisément une distribution gaussienne. De plus, nous constatons que nous pouvons facilement réaliser des techniques de *multi-channel attack*[9][1] lorsque nous intégrons l'apprentissage automatique dans le *template Based DPA* puisqu'à nouveau, il n'est plus important de connaitre la distribution des données pour en déduire une probabilité qu'un comportement soit lié à une clé. Bien que théoriquement il semble y avoir des raisons de choisir cette intégration, notons qu'à notre connaissance, aucune recherche

9. Le *multi-channel attack* est une attaque où nous prenons en compte plusieurs sources d'informations telles que le son, le courant et le temps de chiffrement.

n'a été réalisée pour combiner ces deux domaines. Il est donc intéressant de voir si une nouvelle idée peut être utilisée dans la cryptanalyse.

Cette combinaison n'est pas non plus totalement aléatoire. En effet, grâce au fait qu'une université telle que l'Université Libre de Bruxelles offre le regroupement de plusieurs experts de domaines différents sous le même toit, et donc accessibles en un même lieu, il fût possible de regrouper deux domaines à priori différents : l'apprentissage automatique et la cryptanalyse.

La combinaison entre la cryptologie, qui est la conjointure entre la cryptographie et la cryptanalyse, et l'apprentissage automatique a déjà été imaginé par Rivest en 1993[44]. Ce dernier a vu un lien très clair entre les deux domaines, tous deux reposant sur l'informatique. En effet, ces derniers partagent des notions et des inquiétudes semblables. Dans le cas de la cryptanalyse, nous cherchons une clé cachée dans un système alors que dans l'autre domaine, nous essayons de retrouver une fonction, tous deux pouvant être basés sur des données d'entrée/sortie [10] venant du système cachant la clé ou implémentant la fonction. De plus, l'espace des clés possibles peut être vu comme l'espace des fonctions possibles. Enfin, tous deux ont comme but de trouver des solutions raisonnables en temps et en mémoire de calcul.

Toutefois, ces deux domaines sont aussi parfois en contradiction ainsi, lorsque l'un suppose que l'ensemble des clés possibles est connu, l'autre ne peut supposer que l'ensemble des fonctions possibles, et plus précisément leur complexité, n'est connue qu'avec un certain jeu de données. De plus, lorsque l'un cherche la clé exacte, l'autre ne tente qu'une approximation.

Outre le fait que ces deux domaines sont parfois proches et parfois éloignés, Rivest remarque aussi que l'un peut aider l'autre. En effet, la cryptographie permet de montrer que certains problèmes d'apprentissage sont insolubles en supposant que les algorithmes cryptographiques sont complexes à apprendre. Un exemple cryptographique de problèmes complexes est la factorisation. Dans le cas où ces derniers sont possibles à apprendre, alors au moins un algorithme de chiffrement connu pour être résistant est cassé. Dans notre cas, si le problème de factorisation peut être appris par une technique d'apprentissage automatique, alors l'algorithme RSA [11] n'est plus résistant. Inversement, le domaine de l'apprentissage automatique peut aider la cryptanalyse en apprenant le comportement des fonctions cachées permettant de chiffrer des données sur base de valeurs d'entrées et de sorties.

La prochaine section réalise un apercu du mémoire, en montrant sa structure.

10. Dans le cas de la cryptanalyse, nous cherchons souvent le lien entre le texte en clair et celui chiffré.
11. Rivest Shamir Adleman.

1.3 Structure du mémoire

Ce mémoire est divisé en 8 parties :

1. La première partie est l'introduction où nous montrons le problème qui se pose en cryptanalyse, la motivation de la création d'une nouvelle méthode d'attaque, la présentation de la structure de ce mémoire, une liste des contributions réalisées grâce à ce mémoire et enfin l'ensemble des notations. Ce chapitre permettra d'avoir une vue globale sur le mémoire.
2. La seconde partie explique la cryptologie en général. Nous verrons quelques notions de base de cryptographie telles que les algorithmes symétriques et asymétriques. De plus, dans la section cryptanalyse, nous aborderons quelques techniques d'attaque de base. L'utilité de ce chapitre est d'avoir des notions de base en cryptologie.
3. La troisième partie explique, en détail, l'analyse de canaux auxiliaires - une technique de cryptanalyse - et plus précisemment deux grandes écoles : l'analyse simple de courant et l'analyse différentielle du courant. Le but de cette partie est de comprendre le fonctionnement de la meilleure technique d'attaque contre un matériel de chiffrement.
4. Le quatrième chapitre aborde l'apprentissage automatique. Ce chapitre permettra d'apprendre à construire un modèle d'apprentissage.
5. La cinquième chapitre montre la mise en œuvre de la nouvelle attaque proposée dans ce mémoire. Nous tenterons d'attaquer un matériel de chiffrement.
6. Le sixième chapitre réalise une comparaison entre la meilleure technique d'attaque et la nouvelle. Cette partie nous permettra d'évaluer la nouvelle attaque par rapport à ce qui se fait dans le monde de la cryptanalyse.
7. La septième partie sera la conclusion de ce mémoire. Ce chapitre résumera l'ensemble du mémoire.
8. La dernière partie rassemble les annexes.

Les lecteurs désirant une plus grande connaissance dans le domaine de l'apprentissage automatique sont invités à lire [7] et ceux le souhaitant dans le domaine de la cryptanalyse sont invités à lire [21]. Une grande partie du travail portant sur la cryptologie a été inspirée par [33, 21, 32].

La section suivante présente les contributions de ce mémoire.

1.4 Contributions du mémoire

Durant le mémoire, nous avons réalisé l'ensemble des contributions suivantes :

1. Etat de l'art de la cryptologie.
2. Etat de l'art de l'apprentissage automatique.
3. Proposition d'une nouvelle méthode de cryptanalyse.
4. Mise en œuvre de la nouvelle méthode sur un matériel de chiffrement de production.
5. Evaluation de la nouvelle méthode de cryptanalyse avec une connue.
6. Amélioration des techniques d'attaques contre un matériel de chiffrement.

La prochaine section traite des notations qui seront utilisées pendant le mémoire.

1.5 Notations

Notons quelques définitions de notations qui seront utilisées durant ce travail.

Probabilité

- $P(X)$: la probabilité d'avoir X.
- $P(X|Y)$: la probabilité d'avoir X , sachant que nous avons eu Y.
- Toute lettre $\mathbf{X}$ en gras représente une variable aléatoire. Une observation de celle-ci est notée par une lettre X non gras.
- σ: l'ecart type.
- σ^2: la variance.
- μ : la moyenne.

Vecteur et matrice

- $\mathbf{V}$: un ensemble d'éléments.
- $V_{1...e}$: un vecteur, un ensemble V contenant e éléments.
- V_i : l'$i^{ème}$ élément du vecteur V.
- $V_{1...m,1...e}$: un ensemble de m vecteurs ayant chacun e éléments.
- $V_{i,j}$: l'élément se trouvant à la $i^{ème}$ ligne et à la $j^{ème}$ colonne du tableau V.
- $V_{.,j}$: l'ensemble des éléments se trouvant à la $j^{ème}$ colonne du tableau V.
- $V_{i,.}$: l'ensemble des éléments se trouvant à la $i^{ème}$ ligne du tableau V.
- $\mathbf{T}$: l'ensemble des traces.
- T : une trace.
- T_i : la $i^{ème}$ trace.
- $T_{i,.}$: la $i^{ème}$ trace contenant l'ensemble de ses instants.
- $T_{.,j}$: toutes les traces au $j^{ème}$ instant.
- $T_{i,j}$: $i^{ème}$ trace au $j^{ème}$ instant.

Fonction, chiffrement et déchiffrement

- $F(a) = b$: fonction F qui prend en paramètre a et renvoie comme valeur b.
- $\mathbf{P}$: ensemble des messages clairs.
- P : un message clair.
- $\mathbf{C}$: ensemble des messages chiffrés.
- C : un message chiffré.
- $\mathbf{O}$: ensemble des clés.
- $|\mathbf{O}|$: nombre des clés.
- $\mathbf{O}_e$: ensemble des clés de chiffrement.
- $\mathbf{O}_d$: ensemble des clés de déchiffrement.
- O_i : la $i^{ème}$ clé.
- O_e : une clé de chiffrement.
- O_d : une clé de déchiffrement.
- O : une clé.
- $\mathbf{K}$: ensemble des pairs de clés possibles $\{O_e,O_d\}$.
- $E_A^O(P)$: Chiffrement d'un message P, avec l'algorithme A et la clé O.
- $E_A(P)$: Chiffrement d'un message P, avec l'algorithme A et une clé quelconque.
- $E^O(P)$: Chiffrement d'un message P, avec la clé O et un algorithme quelconque.
- $D_A^O(C)$: Déchiffrement d'un message C, avec l'algorithme A et la clé O.
- $D^O(C)$: Déchiffrement d'un message C, avec la clé O et un algorithme quelconque.
- $D_A(C)$: Déchiffrement d'un message C, avec l'algorithme A et une clé quelconque.
- $D_A(C)$: Déchiffrement d'un message C, avec l'algorithme A et une clé quelconque.
- $F_{E^{O_e}}$: machine qui implémente la fonction E^{O_e}.
- $F_{D^{O_d}}$: machine qui implémente la fonction D^{O_d}.
- $\mathbf{V}_{F_{E^{O_e}}}$: ensemble des observations physiques de $F_{E^{O_e}}$.
- $\mathbf{V}_{F_{D^{O_d}}}$: ensemble des observations physiques de $F_{D^{O_d}}$.
- $F_{FE} : O_e \times P \rightarrow V_{F_{E^{O_e}}}$: fonction qui renvoie le comportement de $F_{E^{O_e}}$.
- $F_{FD} : O_d \times C \rightarrow V_{F_{D^{O_d}}}$: fonction qui renvoie le comportement de $F_{D^{O_d}}$.

Complexité

- $O(n)$: Grand O de n.

Figure

- Dans les figures, un losange représentera une fonction et un rectangle, une donnée.

Chapitre 2

Cryptologie

La cryptologie peut se définir comme la science du secret. Elle se divise en deux grandes écoles :

1. La cryptographie : la mise en place d'un système pour cacher un message, grâce à une clé, une fonction, une connaissance.
2. La cryptanalyse : l'analyse du système mis en place par la cryptographie pour retrouver l'information cachée.

Ces deux parties sont à la fois en concurrence et unies par le fait que bien que la cryptanalyse essaye de casser la cryptographie, elle permet aussi une amélioration de cette dernière. Dans ce contexte, le mot « casser » signifie le déchiffrement de messages chiffrés bien que nous ne soyons pas le destinataire du message, ce que les cryptanalystes appellent un décryptage.

Pour comprendre ce large domaine, nous allons aborder quelques concepts et algorithmes de cryptographie et de cryptanalyse.

2.1 Notions de cryptographie

La cryptographie est un sous ensemble de la cryptologie. Le mot cryptographie vient du grec Kruptos et graphein, signifiant respectivement cacher et écrire. En d'autres mots, celle-ci essaye de protéger ou cacher les messages, en amenant une certaine confidentialité, intégrité et disponibilité des données. Toutefois, il est nécessaire de définir ces trois termes.

Confidentialité La confidentialité spécifie que seules les personnes autorisées à accéder à une certaine information ont la possibilité de l'atteindre. Ce service de sécurité concerne tant les données stockées que les données envoyées au travers d'un réseau. Pour l'assurer, nous utiliserons les contrôles d'accès et le chiffrement. La violation de la confidentialité peut se voir, par exemple, quand une information confidentielle est devenue publique,

grâce aux logs du système ou aux changements de comportement d'une certaine personne envers l'organisation.[37]

Intégrité L'intégrité spécifie que l'information ne peut être modifiée que par les personnes autorisées. Tout comme pour la confidentialité, ce service concerne tant les informations stockées que les données envoyées au travers d'un réseau. Sa protection est souvent la même que celle qui est garantie par la confidentialité. En effet, en contrôlant l'accès à une donnée, nous assurons aussi son intégrité. La détection d'une violation de cette dernière est, par exemple, la comparaison entre l'information et ses copies. Une réponse à cela est la réparation de cette donnée.[37]

Disponibilité La disponibilité est le fait qu'une information est obtenue quand nous en avons besoin. Une de ces attaques connues est le déni de service. Nous pouvons le limiter en modérant les ressources consommables du système par une personne. Une réponse à cela est la réduction de la charge ou l'augmentation des capacités du système.[37]

D'autres concepts de sécurité méritent également d'être cités. Le service de non répudiation permet d'échanger entre deux groupes de personnes, un ensemble de preuves qui montrent des opérations qui se sont effectuées durant une communication telle que la preuve de l'origine d'un message ou la réception de celui-ci. Le vote électronique permet de réaliser un suffrage rapide et de chez soi. La preuve à divulgation nulle de connaissance qui permet à une personne de prouver qu'elle est en possession d'une information sans pour autant la divulguer. Ces concepts sont détaillés dans [53].

La cryptographie met à disposition plusieurs algorithmes de chiffrement. Ces derniers représentent une boite, des fonctions mathématiques, prenant en entrée une clé et un message clair, et retournant un message chiffré. Le message chiffré est parfois appelé le chiffré. Cette procédure s'appelle le chiffrement.

Le cas inverse où nous avons un message chiffré, et donc ne contenant aucune information utile sans la clé de déchiffrement, et nous voulons retrouver le message claire grâce à l'utilisation d'une clé s'appelle le déchiffrement.

De manière plus formelle, inspirée de [50], nous pouvons définir le chiffrement et le déchiffrement ainsi :

P : l'ensemble fini de messages en clairs.

C : l'ensemble fini de messages chiffrés.

O : l'ensemble fini de clés possibles

$\mathbf{O}_e$: l'ensemble fini de clés de chiffrement.

$\mathbf{O}_d$: l'ensemble fini de clés de déchiffrement.

K : l'ensemble fini de pair de clés possibles $\{O_e, O_d\}$.

Pour chaque $K = \{O_e, O_d\}$ appartenant à **K**, il y a une fonction de chiffrement E^{O_e} et de déchiffrement D^{O_d} tel que $E^{O_e} : P \to C$ et $D^{O_d} : C \to P$ où $D^{O_d}(E^{O_e}(P)) = P$ pour tout P appartenant à **P**.

L'efficacité de toute cette procédure repose sur le secret de la clé, l'algorithme de chiffrement pouvant être publique tel que l'explique le principe de Kerckhoffs [1][53].

Nous constaterons par après que les opérations de base, que nous retrouverons dans les algorithmes de chiffrement contemporains, sont la transposition et la substitution. La transposition, ou la permutation, étant le changement de l'ordre des symboles alors que la substitution est le changement de symboles par d'autres.

Dans la cryptographie, il existe deux grandes écoles de chiffrement : le chiffrement asymétrique et symétrique.

2.2 Cryptographie asymétrique

Dans le chiffrement asymétrique, appelé aussi chiffrement à clé publique, deux groupes de personnes sont présents : ceux voulant chiffrer et ceux voulant déchiffrer. Chacun d'eux a une clé différente, l'une permettant de chiffrer et l'autre de déchiffrer. Selon les cas, l'une est une clé publique et l'autre privée. Cette école est moderne dans le sens où la recherche dans ce domaine est contemporaine. En effet, celle-ci a pris un essor considérable après la publication de W. Diffie et M.E. Hellman, en 1976, dans l'article « *New Directions in Cryptography* » [16]. L'avantage de cette découverte réside dans le fait que la clé de chiffrement est publique, la clé de déchiffrement ne pouvant être obtenue, déduite, estimée directement de celle de chiffrement, d'un message en clair, d'un message chiffré ou de l'algorithme de chiffrement utilisé. En d'autres termes, il est calculatoirement impossible à partir de la clé publique, d'un clair, d'un chiffré ou de l'algorithme de chiffrement de retrouver celle privée.

1. Ce principe d'Auguste Kerckhoffs est l'un de ses plus connus, publié dans son essai « *La cryptographie militaire* ».

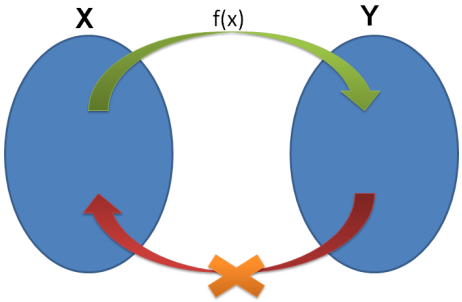


Figure 2.1 : Illustration d'une fonction $f(x)$ à sens unique.

La définition de calculatoirement impossible est [36]:

> « *Calculatoirement impossible signifie que les délais nécessaires afin de retrouver un message clair ou une clé dépassent la durée de validité du message et/ou de la clé, et ce sur base de la technologie actuelle et celle supposée par extrapolation pendant cette durée.* »

De plus, cette découverte a permis la réalisation de la signature digitale où nous inversons les rôles des clés. Ainsi, la clé de chiffrement est secrète et celle de déchiffrement est publique. Toutefois, ce dernier domaine ne sera pas traité dans ce mémoire.

Pour pouvoir comprendre la cryptographie asymétrique, il est nécessaire de définir deux concepts :

1. Une fonction est à sens unique si celle-ci ne peut être inversée. Autrement dit, une fonction $f(x)$ à sens unique permet de calculer le résultat y de la fonction, mais en prenant un résultat y, nous ne pouvons pas retrouver un paramètre x, parmi l'ensemble des paramètres x' tel que $f(x') = y$, qui l'engendre [2] dans un espace de temps ou de stockage raisonnable. Ce principe est illustré à la figure 2.1.
2. Une fonction $f(x)$ à sens unique est dite de trappe si une certaine information, appelée de trappe, permet de réaliser l'inverse de la fonction $f(x)$ en un temps et un espace mémoire polynomial. Autrement dit, grâce à une information, nous pouvons retrouver le paramètre x en ne connaissant que le résultat $f(x)$.

Dans le chiffrement asymétrique, Alice crée deux clés. L'une permet de chiffrer un message grâce à la fonction à sens unique $f(P, O_e)$. Cette dernière prend en paramètre le message P à chiffrer et la clé publique de chiffrement O_e. Pour inverser cette fonction et donc déchiffrer le message secret, Bob utilise une clé secrète O_d qui est donc une information trappe.

Il existe plusieurs algorithmes appartenant à cette école tels que le RSA et l'ElGamal.

2. Nous supposons évidemment que le paramètre x n'est pas connu avant d'essayer de réaliser l'inversion de la fonction à sens unique.

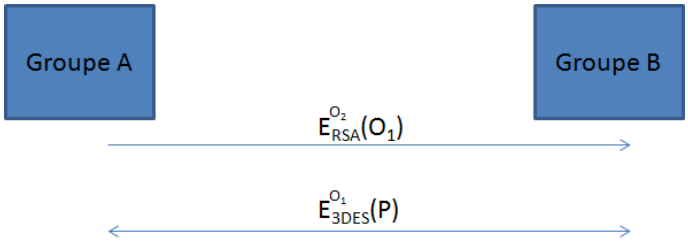


Figure 2.2 : Dans le cas où deux groupes, Alice (A) et Bob (B), veulent s'échanger des messages rapidement, ils s'échangent d'abord une clé O_1 chiffrée grâce à un algorithme asymétrique et une clé O_2 où le groupe B est l'unique personne pouvant la déchiffrer. Après avoir déchiffré le premier message, les deux groupes peuvent se baser sur un algorithme symétrique pour s'échanger des messages rapidement.

La prochaine section traite de la cryptographie symétrique, la seconde école de la cryptographie.

2.3 Cryptographie symétrique

Dans le chiffrement symétrique, ou encore le chiffrement à clé secrète, nous sommes face à deux informations ou deux clés ou deux secrets partagés par deux groupes de personnes : l'un voulant chiffrer et l'autre voulant déchiffrer. La clé de chiffrement et de déchiffrement est fortement dépendante, voir identique.

Cette école est la première qui a vu le jour dans le domaine de la cryptographie. Bien qu'elle soit ancienne, elle est encore utilisée. Un de ses avantages est la simplicité et la rapidité de ses algorithmes, en manipulant des opérations simples et rapides [3].

Néanmoins, cet avantage est contré par le fait que pour pouvoir s'échanger des messages secrets, deux groupes doivent s'échanger la clé de chiffrement et de déchiffrement au préalable de manière cachée. Ainsi, nous nous basons souvent sur un protocole où nous échangeons une clé de chiffrement pour un algorithme symétrique, grâce à un algorithme asymétrique tel que le schématise la Figure 2.2. Ce dernier montre un protocole pour s'échanger une clé de chiffrement symétrique. Pour cela, le groupe A envoie en premier lieu la clé de chiffrement symétrique O_1 chiffrée avec l'algorithme de chiffrement asymétrique RSA. Ainsi, la partie A utilise la clé publique O_2 de la partie B impliquant que seul le groupe B peut déchiffrer le message. Par après, pour accélérer le processus d'échange de messages, le groupe B et A peuvent chiffrer avec la clé d'algorithme de chiffrement symétrique O_1.

3. Selon [6], la rapidité d'exécution d'un algorithme symétrique est 1000 fois plus élevée que celle d'un algorithme asymétrique.

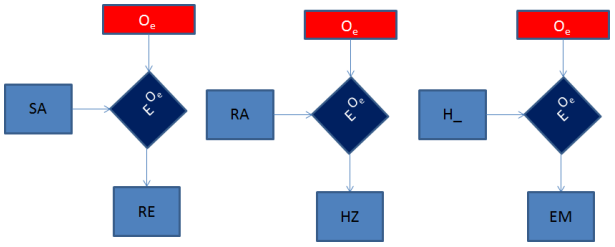


Figure 2.3 : Illustration de l'algorithme ECB où nous chiffrons un message contenant 3 blocs, avec un même algorithme et une même clé.

Les algorithmes de chiffrement symétrique peuvent se diviser en deux types de familles : le chiffrement par bloc et celui par flot. Le premier réalisant un découpage des messages à chiffrer en bloc de même taille. Le second réalise un chiffrement continu. Ainsi, dans ce dernier cas, nous utilisons une clé, appelée aussi graine, qui permet de générer une suite (pseudo)aléatoire. Cette suite sert à chiffrer durant toute une période donnée, telle que le fait une clé pour un chiffrement par bloc. Toutefois, la question qui se pose est « *Comment chiffrer ou déchiffrer par bloc des données qui ont plusieurs blocs ?* ». Autrement dit, comment chiffrer et déchiffrer avec un algorithme, qui prend un bloc, un message en contenant plusieurs. Devons nous utiliser des clés différentes pour chaque bloc ? La réponse est oui et non. Oui, nous pouvons en utiliser plusieurs mais ce n'est pas efficace puisqu'il faudrait en pratique un nombre important de clé, ce qui n'est évidemment pas aisé. Ainsi, il existe des méthodes permettant d'utiliser un algorithme de chiffrement par bloc sur un ensemble de blocs, tout en utilisant une seule clé. Le NIST [4] propose au moins 5 types de modes d'opérations pour cela, chacun ayant ses propres atouts :

1. Chiffrement à rétroaction de sortie (Output feedback mode) (OFB)
2. Enchaînement des blocs (Cipher block chaining mode) (CBC)
3. Dictionnaire de codes (Electronic codebook mode) (ECB)
4. Chiffrement à rétroaction (Cipher feedback mode) (CFB)
5. Chiffrement basé sur un compteur (CounTer) (CTR)

La figure 2.3 illustre un mode d'opération, le ECB, en chiffrant le mot SARAH_. Ce dernier contient six caractères. En supposant que l'algorithme de chiffrement E^{O_e} ne peut prendre que deux caractères pour chiffrer un bloc, nous découpons le message en trois. Chaque partie est chiffrée avec la même clé O_e. Ainsi, nous obtenons le chiffré REHZEM.

Nous renvoyons les lecteurs désirant avoir plus d'information sur ces modes d'opération à l'article publié par le NIST sur leur site internet, en 2001, qui s'appelle « *Recommendation for Block Cipher Modes of Operation* » [18].

4. National Institute of Standards and Technology.

Il existe beaucoup d'algorithmes de chiffrement symétrique par bloc tel que DES[5], 3DES[6] et AES[7]. Pour ce qui est du chiffrement par flot, il existe entre autre RC4.

Bien que chaque algorithme a des avantages et des inconvénients, nous allons nous focaliser sur l'un d'entre eux : le 3DES. Mais avant d'aborder ce dernier, nous devons nous pencher sur l'algorithme DES qui est un cas particulier de 3DES.

2.4 Data Encryption Standard

Le 15 mai 1973, le NIST a proposé un concours pour la réalisation d'un algorithme de chiffrement qui sera un standard [50]. Ce fut l'algorithme d'IBM[8], Lucifer, qui a été pris. Après un ensemble de modifications sur ce dernier, il fut officiellement le standard le 15 janvier 1977 sous le nom de DES [50].

Le DES est un algorithme de chiffrement symétrique par bloc. Pour le chiffrement et le déchiffrement, nous utilisons une clé unique. Cette dernière contient 56 bits, soit 8 bytes où le bit de parité n'est pas pris en compte. Ainsi, l'algorithme prend en entrée un bloc de 64 bits et une clé de 56 bits et transforme le tout en un bloc chiffré de 64 bits. L'ensemble de l'algorithme est structuré selon le schéma de Feistel qui est formalisé ainsi :

$P_0 = G_0.D_0$ où . est l'operateur de concaténation, P_0 est le message en clair, G_0 est la partie gauche du P_0 et D_0 l'autre partie du message P_0.

$P_i = G_i.D_i$ où $G_i = D_{i-1}$ et $D_i = G_{i-1} \oplus f(D_{i-1}, O_i)$ où $\oplus$ est l'opérateur XOR, f une fonction non linéaire, O_i une clé et $i \in \mathbb{N} \backslash 0$. L'ensemble des O_i sont généralement des dérivées d'une clé principale, et sont appelées les clés intermédiaires. Ils sont donc générées grâce à une fonction qui prend en entrée une clé et renvoie un ensemble de clés intermédiaires O_i.

$C_F = P_r$ où C_F est le message chiffré et $r \in \mathbb{N} \setminus 0$.

Ainsi, à partir d'un message en clair P_0, nous obtenons un message chiffré C_F. Comme dans tout schéma de Feistel, le déchiffrement se fait dans l'ordre inverse du chiffrement.

Notons que ce schéma est largement utilisé en cryptographie. En effet, nous pouvons le généraliser en changeant :

- le XOR par une autre fonction prenant deux paramètres
- la taille de chaque sous partie de P_0
- ...

5. Data Encryption Standard.
6. Triple Data Encryption Standard.
7. Advanced Encryption Standard.
8. International Business Machines.

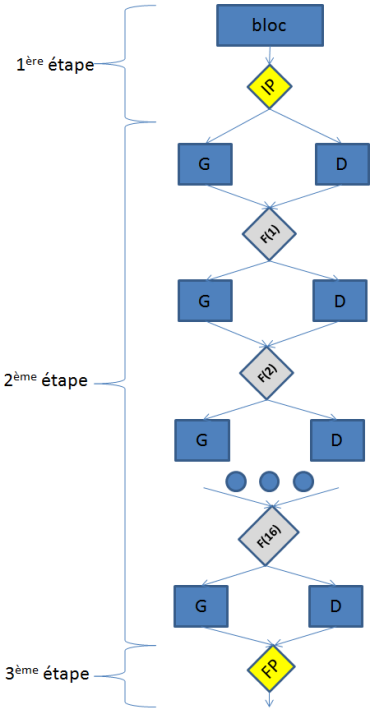


Figure 2.4 : illustration de l'algorithme DES. Nous y voyons les trois grandes parties de l'algorithme. La première où nous prenons un bloc de donnée que nous permutons avec la fonction IP et que nous divisons en deux sous blocs. La seconde où nous itérons 16 fois l'insertion de ces deux blocs dans une fonction $F(i)$. La dernière étape combine deux blocs et réalise une dernière permutation.

Pour chiffrer un bloc, DES implémente trois grandes étapes :

1. Permutation initiale du bloc (IP [9])
2. Réalisation de 16 étapes similaires
3. Permutation inverse du bloc (FP [10])

Le déchiffrement se fait en inversant l'ensemble des opérations. L'ensemble de ces étapes est illustré à la Figure 2.4 où nous voyons les 3 parties listées précédemment et qui seront détaillées par la suite.

9. Initial Permutation
10. Final Permutation

2.4.1 Permutation initiale du bloc

Regardons de plus près la première étape.
Soit un bloc de 64 bits à chiffrer :

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

La permutation du bloc ne fait que déplacer des bits du bloc d'un emplacement à un autre. Du point de vue cryptographique, notons qu'une permutation n'a aucune influence sur la sécurité d'un algorithme. Pour réaliser la permutation, nous utilisons la matrice suivante :

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Cette matrice montre que le $58^{ème}$ bit du bloc sera mis à la $1^{ère}$ position, que le $50^{ème}$ bit du bloc sera mis à la $2^{ème}$ position, ... que le $60^{ème}$ bit du bloc sera mis à la $9^{ème}$ position, ... que le $7^{ème}$ bit du bloc sera mis à la dernière position.

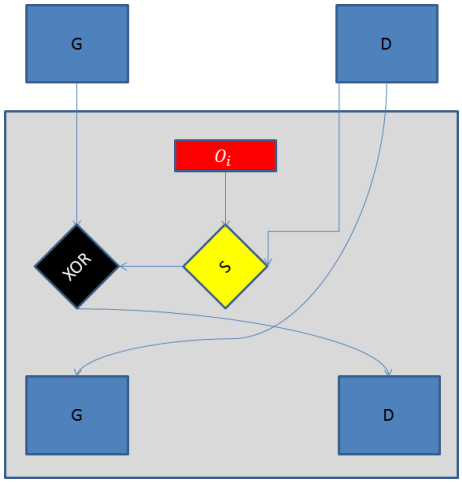


Figure 2.5 : Représentation d'un tour parmi les 16 de la $2^{ème}$ étape du 3DES.

2.4.2 Réalisation de 16 étapes similaires

Désormais, regardons de plus près la deuxième étape. Elle repose sur 16 tours, chacun d'entre eux utilisant une clé intermédiaire de 48 bits construite avec la clé initiale O grâce à une fonction $K(O)$ appelée key schedule. Cette dernière sera détaillée par la suite.

L'algorithme repose sur un schéma de Feistel. Pour cela, nous divisons le bloc en 2 sous-ensembles, contenant chacun 32 bits : G(sous-ensemble gauche) et D(sous-ensemble droit).

G :

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8

D :

57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

À chaque tour, nous avons deux sous-ensembles : G et D. Le sous ensemble G est combiné avec une version modifiée de D, grâce à la fonction XOR. Cette combinaison donne un sous ensemble G'. La version modifiée de D se construit grâce à une fonction non linéaire $S(D, O_i)$ qui prend en entrée D et la clé intermédiaire O_i et qui renvoie 32 bits. Au prochain tour, D et G' deviennent respectivement G et D. Ce processus est illustré à la Figure 2.5.

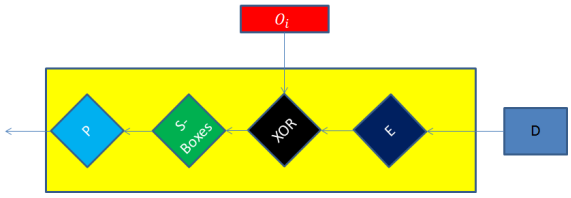


Figure 2.6 : Illustration de la fonction $S(D, O_i)$.

2.4.3 Permutation inverse du bloc (FP)

Enfin, finissons par une focalisation de la dernière étape de l'algorithme. Nous combinerons les deux sous-ensembles avant de réaliser une permutation finale. Cette dernière se fait grâce au tableau suivant :

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Nous remarquons que le $40^{ème}$ bit devient le 1^{er}, ... le $25^{ème}$ bit devient le dernier.

Nous venons de voir les trois grandes étapes de l'algorithme. Toutefois, nous n'avons pas encore détaillé les fonctions $K(O)$ et $S(D, O_i)$ utilisées à la deuxième étape de l'algorithme.

2.4.4 S(D,O_i)

Regardons d'abord la fonction $S(D, O_i)$. Celle-ci prend en entrée le sous-ensemble D et une clé intermédiaire O_i. Cette fonction est divisée en quatre étapes illustrées par la figure 2.6. Cette dernière montre l'ensemble des étapes qui sont présentées ci-dessous.

E

La fonction E s'appelle la fonction d'extension. Nous agrandissons la taille de la donnée D, de 32 bits à 48 bits, dans le but d'avoir une même taille que celle de O_i. Pour cela, nous utilisons le tableau suivant :

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Ce tableau montre que le dernier bit du sous-ensemble D devient le premier, le 1^{er} bit du sous-ensemble D devient le second,... Notons que nous dupliquons certains bits pour arriver à 48 bits.

XOR

La fonction XOR permet, après avoir agrandi la taille de D, de combine D à la clé intermédiaire O_i.

S-Boxes

Par la suite, nous insérons le résultat précèdent dans une nouvelle fonction non linéaire S-Boxes qui prend en entrée 48 bits et retourne 32 bits. Cette fonction, appelée substitution, sera détaillée par la suite. Toute la puissance de l'algorithme repose sur cette fonction, et plus précisément sur sa non linéarité.

P

Finalement, nous permutons le résultat précèdent grâce à la matrice suivante :

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Ce tableau montre que le $16^{ème}$ bit devient le 1^{er}, ..., le $25^{ème}$ bit devient le dernier.

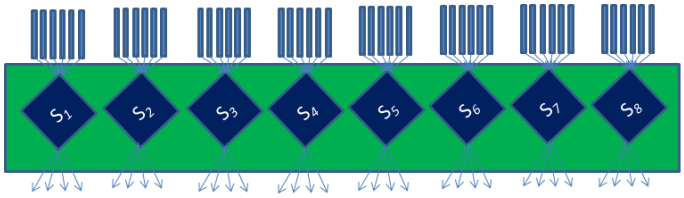


Figure 2.7 : Cette figure montre le contenu de la fonction S-Boxes. Celle-ci prend 6×8 bits et renvoie 4×8 bit grâce à 8 sous-fonctions S_i.

Dans l'algorithme précèdent, nous avons utilisé la fonction S-Boxes. Détaillons là désormais avant de détailler la fonction $K(O)$.

2.4.5 S-Boxes

La fonction non linéaire S-Boxes prend en entrée 48 bits et retourne 32 bits. Pour cela, celle-ci découpe les 48 bits en 8 morceaux, chacun d'eux ayant donc 6 bits. Chaque morceau est, par la suite, entré dans une sous-fonction S_i, appelée fonction de sélection. Le regroupement de ces fonctions S_i constituent S-Boxes. Chaque S_i est représentée par une matrice de taille 4×16, visible en Annexe 1, où la valeur renvoyée de 4 bits est celle qui se trouve dans la case choisie. Le premier et dernier bit du morceau détermine le numéro de ligne de la matrice, le reste détermine le numéro de colonne. La figure 2.7 illustre la S-Boxes.

2.4.6 Calcul des clés intermédiaires

Désormais, regardons la fonction $K(O)$. Celle-ci a comme but de donner une clé différente à chacune des 16 étapes de l'algorithme. Pour cela, elle se base sur la clé O donnée initialement pour chiffrer son bloc. L'algorithme de cette fonction se résume ainsi :

Permutation

Nous commençons par la permutation des bits de la clé O grâce à la matrice suivante :

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Cette matrice montre que le $57^{ème}$ bit devient le 1^{er}, ..., le $4^{ème}$ bit devient le dernier.

Décomposition

Par après, nous décomposons la clé en deux parties G et D :

G :

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

D :

63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Rotation

Nous réalisons une ou deux rotations de chaque partie vers la gauche. Le nombre de rotation est déterminé par le nombre de clés intermédiaires déjà prises. Si nous sommes à la $1^{ère}$, $2^{ème}$, $9^{ème}$ ou $16^{ème}$ clé intermédiaire, il faudra réaliser 1 rotation de chaque partie. Sinon, nous devons réaliser 2 rotations pour chaque partie.

Regroupement

Nous regroupons les deux parties pour n'en former qu'une seule de 56 bits.

Permutation

Nous réalisons une dernière permutation grâce à la matrice suivante :

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Pour chaque clé intermédiaire, nous réalisons une fois cet algorithme.

Pour les lecteurs désireux d'avoir plus d'informations sur cet algorithme, nous les invitons à voir la publication par le FIPS [11] du standard de DES datant du 25 octobre 1999 (FIPS PUB 46-3). Toutefois, dû à la petite taille de la clé dans DES qui peut être retrouvée avec un ordinateur moderne, la plupart des organisations utilisent non pas DES mais 3DES.

11. Le FIPS (Federal Information Processing Standards) sont des standards publiés par le gouvernement des Etats-Unis d'Amérique .

2.5 Triple Data Encryption Standard

L'algorithme DES vu précédemment a été sujet à beaucoup de critique [12]. L'une d'entre elles portait sur les 8 boites qui constituent la fonction S-Boxes. Cette dernière est l'unique partie non linéaire. De ce fait, elle représente l'unique sécurité [13] et a pourtant été créée sans réelle démonstration de son efficacité. Selon [50], cela a amené des soupçons sur une éventuelle porte cachée permettant de décrypter tout message chiffré avec DES. Toutefois, selon la même source, certains tests actuelles montrent qu'il n'existait pas de portes cachées.

Sa grosse faiblesse de nos jours est la petite taille de sa clé. En effet, selon [50], un ensemble d'ordinateurs spécialisés ont été créés. Ces derniers réalisent une recherche exhaustive de la clé d'un DES, en n'ayant que le chiffré et le clair. Grâce à cela, une clé DES peut être retrouvée en moins d'une journée. Pour contrer cela, nous pouvons agrandir la taille de la clé. Ainsi, nous itérons désormais 3 fois l'algorithme DES sur le bloc à chiffrer en utilisant une, deux ou trois clés différentes et indépendantes O_1, O_2, O_3. L'ordre d'utilisation de ces clés est choisi par l'utilisateur. Ainsi, si nous travaillons avec deux clés différentes, nous pouvons, par exemple, réaliser un O_1, O_2, O_1 où nous utilisons la deuxième clé pour la seconde itération et la première clé pour la première et la dernière itérations. Toutefois, notons qu'utiliser une seule clé, où $O_1 = O_2 = O_3$, avec le 3DES n'améliore pas la puissance de cette algorithme, et ne le fait que ramener au problème de DES qu'est la longueur de la clé. De surcroît, comme nous le verrons par la suite, le 2DES, où nous itérons deux fois DES, n'améliore pas la complexité par rapport au DES.

Pour ce qui est du chiffrement, le standard propose de chiffrer, déchiffrer et rechiffrer (EDE [14]). Inversement, pour déchiffrer un message, il est proposé de déchiffrer, chiffrer et redéchiffrer (DED [15]). Une opération de chiffrement est illustrée à la figure 2.8 où nous y voyons 3 itérations de l'algorithme DES.

Notons que nous pouvons réaliser un EEE [16] pour le chiffrement et un DDD [17] pour le déchiffrement. Toutefois, cela n'est pas dans le standard de 3DES.

Désormais que nous savons comment chiffrer un message, nous allons regarder comment casser un algorithme de chiffrement. En d'autres termes, nous allons essayer de décrypter des messages grâce à la cryptanalyse.

12. Des attaques puissantes contre DES ont été présentées dans [5].
13. Nous verrons par la suite comment des algorithmes ne contenant pas de partie non linéaire sont rapidement cassables grâce à une analyse statistique.
14. Encryption, Decryption, Encryption.
15. Decryption, Encryption, Decryption.
16. Encryption, Encryption, Encryption.
17. Decryption, Decryption, Decryption.

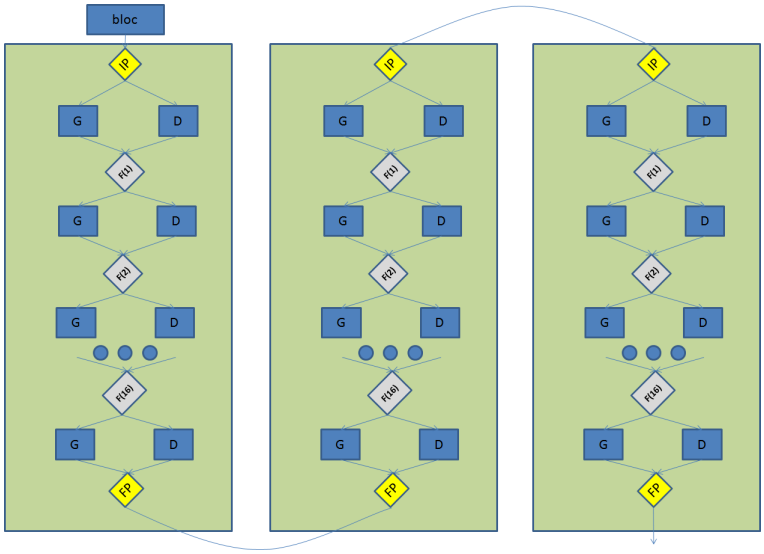


Figure 2.8 : Cette illustration montre l'algorithme 3DES où nous prenons un bloc en entrée, nous itérons trois fois l'algorithme DES et nous renvoyons le résultat en sortie.

2.6 Notions de cryptanalyse

Un algorithme de chiffrement est considéré comme sûr si ce dernier résiste à des attaques dont le besoin en temps ou en espace de mémoire est raisonnable. En d'autres termes, ces derniers ne permettent pas de découvrir la clé secrète, ni le message en clair. Dans la plupart des cas, le chiffrement est d'autant plus sûr que la clé a une grande taille[18]. Par conséquent, la taille de la clé est un paramètre essentiel de la sureté d'un algorithme de chiffrement.

Dans la littérature de la cryptanalyse, il existe un vaste choix de méthode d'attaque pour retrouver la clé secrète. Ces attaques différent par le coût, le temps de calcul, la connaissance du matériel, l'algorithme de chiffrement utilisé,... Ainsi, cette section commencera par détailler une technique de classification de différents types d'attaques connues ainsi que de définir quand une attaque est concidérée comme reussite, pour par la suite en détailler quelques-unes. Notons que pour être complet, il fut nécessaire de présenter les techniques de bases pour par la suite comprendre comment ont évoluées les techniques d'attaques, et pourquoi la technique proposée dans ce mémoire semble puissante en théorie. Toutefois, nous renvoyons, vers [21], les lecteurs désirant connaitre plus d'informations sur d'autres attaques, comme l'attaque différentielle ou linéaire.

18. Notons que bien que la taille joue un rôle important, le choix de la clé l'est tout autant.

2.6.1 Technique de classification

Selon [53], les techniques d'attaques peuvent être classifiées dans 4 catégories :

Attaque à texte chiffré connu Cette famille d'attaques fait comme hypothèse de base que l'attaquant a uniquement les messages chiffrés.

Attaque à texte clair connu Cette famille fait comme supposition qu'Eve a les textes clairs et les chiffrés de ces clairs.

Attaque à texte chiffré choisi Cet ensemble d'attaques fait comme hypothèse que l'attaquant peut choisir les messages chiffrés et recevoir leur clair.

Attaque à texte clair choisi Cet ensemble d'attaques suppose qu'Eve peut choisir les textes clairs et obtenir leur chiffré. Cette attaque est largement réalisable dans les algorithmes asymétriques où la clé de chiffrement est publique. Ainsi, l'attaquant peut chiffrer l'ensemble des messages qu'il choisit. Le but final étant de choisir les bons couples de clair/chiffré pour retrouver une information utile.

La section suivante essayera de répondre à la question : "*Quand un algorithme de chiffrement est considéré comme cassé ?*"

2.6.2 Réussite de l'attaque

Il existe plusieurs découvertes qui permettent de dire qu'un algorithme de chiffrement est cassé. La liste suivante regroupe des exemples :

1. Si nous retrouvons la clé de chiffrement
2. Si nous retrouvons le message clair
3. Si nous retrouvons un sous-ensemble du message clair
4. Si nous retrouvons une information du message clair

Il existe un ensemble très vaste d'attaques. En effet, il existe autant de techniques d'attaque que d'algorithmes de chiffrement et certaines de ces méthodes sont généralisées pour une famille d'algorithmes de chiffrement. Ni la liste exhaustive de ces attaques, ni de leur contremesure ne pourrait être donnée ici. Pour les lecteurs désirant apprendre des contremesures sur les attaques présentes dans ce document, [21] est un bon choix. Toutefois, essayons de détailler quelques-unes de ces attaques pour voir d'où a débuter la cryptanalyse jusqu'à arriver à l'attaque utilisée dans ce travail avec l'aide de l'apprentissage automatique.

2.6.3 Analyse fréquentielle

Une des plus anciennes méthodes de cryptanalyse est celle se basant sur des propriétés statistiques et probabilistes de base telle que l'analyse fréquentielle. Cette dernière amène à retrouver une clé, en ne se basant que sur la fréquence des lettres dans un message chiffré. De ce fait, c'est une attaque à texte chiffré connu et est particulièrement efficace sur des algorithmes de chiffrement où la seule opération mathématique est une substitution mono-alphabétique où chaque lettre est remplacée par une autre. Elle fut inventée par Al-Kindi au $9^{\text{ème}}$ siècle [21]. Pour des chiffrements poly-alphabétiques, où un symbole peut être remplacé par un ensemble d'autres symboles, nous utilisons un outil similaire et est détaillé dans [21]. Prenons l'exemple de la phrase suivante :

« *SPYHU CVBZ WYLZLUAL SH JYFWAHUHSFZL*

HCLJ S HPKL KB THJOPUL SLHYUPUN »

Cette phrase peut être décryptée en regardant la fréquence de chaque lettre de l'alphabet française. De manière originale, la fréquence de chaque lettre peut être obtenue en regardant un document écrit en français. Le tableau suivant affiche l'ensemble des fréquences de chaque lettre calculée dans [33].

A	7.03	**N**	8.14
B	0.77	**O**	5.43
C	3.26	**P**	3.54
D	4.40	**Q**	1.13
E	15.37	**R**	6.02
F	1.25	**S**	9.38
G	0.94	**T**	8.12
H	0.77	**U**	5.09
I	7.18	**V**	0.99
J	0.08	**W**	0.05
K	0.17	**X**	0.49
L	5.39	**Y**	0.51
M	2.87	**Z**	0.01

Ainsi, théoriquement, la lettre « E » a une fréquence de 15.87, la lettre « A » une fréquence de 9.48,... Lors de la phase d'attaque, nous regardons quelle lettre de la phrase chiffrée à une fréquence proche de 15.87 pour en déduire que celle-ci soit le « E » et ainsi de suite.

Cette technique peut aussi s'appliquer sur tout un ensemble de lettres ou plus exactement une combinaison de lettres, où la fréquence de cette réunion est connue. Cette

attaque est réalisable sur l'ensemble des algorithmes de chiffrement où nous ne réalisons que des substitutions pour chiffrer les messages, telles que le chiffrement de César.

L'avantage conséquent de l'attaque fréquentielle est sa facilité de compréhension et de mise à l'œuvre. Toutefois, elle a aussi énormément de limites telles que le fait que si un message chiffré n'a pas la même fréquence théorique que celle de la langue théoriquement utilisée, la clé ne pourra être rapidement retrouvée. De plus, les algorithmes de chiffrement contemporains réalisent des opérations détruisant la possibilité de ce type d'attaque.

2.6.4 Force brute

La force brute n'est nul autre que la technique essayant toutes les clés possibles pour retrouver la bonne. Cette attaque est réalisable dans le cas où l'ensemble des clés possibles est faible, ou de manière analogue où l'algorithme de recherche de la clé est efficace. Ce qui veut dire que les ordinateurs contemporains peuvent réaliser cette recherche en un temps et un espace mémoire raisonnable. La loi de Moore est particulièrement intéressante puisque celle-ci énonce que la rapidité de calcul des processeurs double chaque 18 mois. De ce fait, lors de la mise en place d'un algorithme de chiffrement, il est nécessaire de prendre en compte la durée de temps qu'il faudra pour qu'un ordinateur puisse réaliser cette attaque.

Il existe plusieurs variantes de cette technique :

Attaque par dictionnaire Nous avons une table avec toutes les clés les plus probables qu'Alice et Bob ont utilisées et nous réalisons une force brute sur ce sous ensemble de clé.

Codebook attacks Nous listons dans un tableau un ensemble de chiffrés et un ensemble de clairs qui correspondent aux premiers. Nous cherchons alors quel message chiffré coïncide au message échangé entre Alice et Bob.

Cette méthode est rarement utilisée dû essentiellement à sa lenteur d'exécution. En effet, les algorithmes de chiffrement modernes prennent en compte que les ordinateurs sont puissants pour pouvoir réaliser ce genre d'attaque. Ainsi, les cryptographes mettent tout en œuvre pour que si cette technique est utilisée par un attaquant, celui-ci ne puisse retrouver la clé pendant un temps et un espace de stockage raisonnable. Toutefois, cette attaque est parfois utilisée pour retrouver un sous ensemble de la clé lorsqu'une majorité de l'ensemble de la clé fût trouvée grâce à une attaque moderne. Cette dernière étape permet souvent d'accélérer le processus d'attaque. C'est pourquoi la stratégie proposée en fin de ce mémoire utilisera en dernier lieu cette attaque.

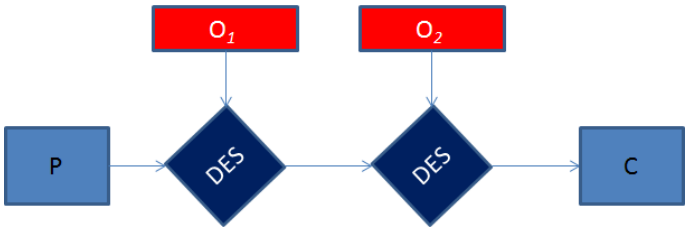


Figure 2.9 : Illustration de l'algorithme 2DES.

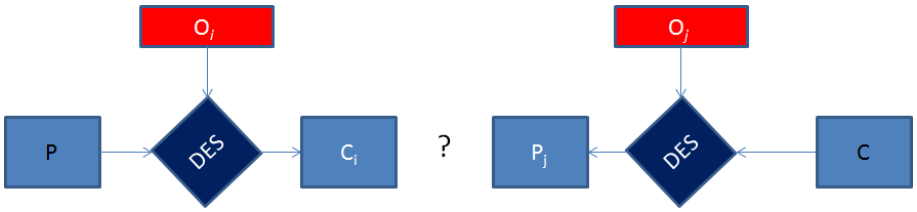


Figure 2.10 : Cette figure illustre l'attaque du *meet in the middle attack* où nous avons un ensemble de paires (C_i,P_j) et nous cherchons celles dont les deux parties sont égales.

2.6.5 Meet in the middle attack

Le *meet in the middle attack* est une technique décrite par Ralph Merkle et Martin Hellman en 1981 [39]. Celle-ci est réalisée sur la plupart des algorithmes de chiffrement par bloc se composant d'une succession de sous-algorithmes, tels que 2DES. Notons qu'à la base, le 2DES fut inventé pour augmenter la taille de la clé et ainsi accroître drastiquement le temps de recherche d'une clé. En effet, le DES ne repose que sur 56 bits de clé. Ce dernier fût rapidement attaquable grâce à des ordinateurs contemporains via la force brute. Toutefois, le *meet in the middle attack* réduit la complexité en grand O de 2DES à un simple DES.

Celle-ci est une attaque passive et plus précisément une attaque à message clair connu. Dans le cas de 2DES, nous chiffrons une donnée en clair avec une clé O_1 et nous rechiffrons la donnée avec une clé O_2. Le 2DES est illustré à la figure 2.9.

De manière plus formelle, nous avons :

$C = E_{DES}^{O_1}(E_{DES}^{O_2}(P))$

De ce fait, nous avons que $E_{DES}^{O_1}(P) = D_{DES}^{O_2}(C)$

Nous supposerons dans cette attaque que nous avons un ensemble de message en clair P et le chiffré correspondant C (P, C). L'attaque, qui a été proposée dans [15], consiste en plusieurs étapes, illustrées par la figure 2.10 et détaillé dans l'algorithme 2.1.

Ainsi, au lieu d'avoir un algorithme de chiffrement avec $|\mathbf{O}| \times |\mathbf{O}|$ clés possibles, nous en avons un de $|\mathbf{O}| + |\mathbf{O}|$. En d'autre terme, nous passons d'un $O(|\mathbf{O}| \times |\mathbf{O}|)$ à un $O(|\mathbf{O}|)$. Toutefois, il est nécessaire de préciser que pour réaliser ce type d'attaque, il faut stocker un nombre conséquent de données en mémoire ce qui peut être une limite.

Algorithme 2.1 : Le *meet in the middle attack* contre 2DES.

1. Nous chiffrons P avec toutes les clés possibles : $O_1, O_2, ..., O_{|\mathbf{O}|}$. Ce qui nous donne un vecteur de chiffré temporaire : $(C_1, C_2, ..., C_{|\mathbf{O}|})$
2. Nous déchiffrons C avec toutes les clés possibles : $O_1, O_2, ..., O_{|\mathbf{O}|}$. Ce qui nous donne un vecteur de message clair temporaire : $(P_1, P_2, ..., P_{|\mathbf{O}|})$
3. Nous regardons quel C_i est égale à P_j. Si C_i est égale à P_j, c'est qu'il est fort probable que les clés utilisées sont O_i et O_j. Or, lors de la mise en œuvre de l'attaque, nous retrouverons plusieurs paires (O_i,O_j). De ce fait, nous devons essayer de regarder d'autres paires (P, C).
4. Comme expliqué précédemment, il est nécessaire de regarder une autre paire (P, C) pour retrouver la bonne paire (O_i,O_j) de l'ensemble des paires. Ainsi, plus nous avons de paires (P, C), plus nous augmentons la probabilité de retrouver la bonne clé.

2.6.6 Inversion de fonction supposée à sens unique

Certains algorithmes de chiffrement, tels que RSA, font l'hypothèse que pour retrouver la clé de déchiffrement ou de chiffrement, il est nécessaire de réaliser une opération qui nécessite un temps ou un espace mémoire abusifs par rapport à l'offre réalisée dans le domaine de l'informatique contemporaine. Autrement dit, nous supposons que nous travaillons avec des fonctions à sens unique.

Dans le cas de l'algorithme RSA, ce dernier suppose que la factorisation d'un grand nombre est complexe. Toutefois, dans le cas où cette hypothèse est cassée, comme le suggèrent les algorithmes qui retrouvent les facteurs premiers d'un grand nombre et qui se basent sur un ordinateur quantique [19], l'algorithme de chiffrement n'est plus sûr. Pour les lecteurs désirant plus d'informations sur ce cassage, l'article [46] est un bon choix.

2.6.7 Analyse de canaux auxiliaires

L'analyse de canaux auxiliaires essaye de retrouver de l'information sur la clé de chiffrement, en ne regardant que les mesures [20] sur le comportement du matériel physique. Ce dernier se nomme canal auxiliaire. Les attaques se basant sur ce dernier se nomment des attaques de canaux auxiliaires. Nous supposerons donc que le comportement du matériel de chiffrement est corrélé avec les données qu'il traite, et plus précisément avec la clé secrète qu'il utilise [21].

19. Notons que pour l'instant, cette suggestion n'est que théorique.
20. Selon ce que nous regardons, l'unité de mesure peut être en volt, en seconde, ...
21. Une donnée cachée et non traitée par le matériel n'influence donc pas son comportement et ne peut donc pas être obtenue avec cette technique.

Formellement, nous avons :

$F_{E^{O_e}}$: la machine qui implémente la fonction E^{O_e}.

$F_{D^{O_d}}$: la machine qui implémente la fonction D^{O_d}.

$\mathbf{V}_{F_{E^{O_e}}}$: l'ensemble des observations physiques de $F_{E^{O_e}}$ telles que le temps de chiffrement.

$\mathbf{V}_{F_{D^{O_d}}}$: l'ensemble des observations physiques de $F_{D^{O_d}}$ telles que le courant.

$F_{FE} : O_e \times P \rightarrow V_{F_{E^{O_e}}}$: avec un clair P et une clé O_e, la fonction F_{FE} nous renvoie le comportement [22] de $F_{E^{O_e}}$.

$F_{FD} : O_d \times C \rightarrow V_{F_{D^{O_d}}}$: avec un chiffré C et une clé O_d, la fonction F_{FD} nous renvoie le comportement de $F_{D^{O_d}}$.

L'idée est venue du fait que l'attaquant peut obtenir beaucoup plus d'informations que les simples messages chiffrés ou clairs durant l'exécution d'un algorithme de chiffrement ou de déchiffrement.

Selon [27], les deux conditions nécessaire pour que cette attaque puisse être réalisable sont que :

1. il doit être possible de récolter des données liées à la clé de chiffrement en un temps et un espace polynomial.
2. il doit être possible de retrouver la clé en se focalisant sur les données récoltées en un temps et un espace polynomial.

En d'autres termes, pour prouver qu'une machine est résistante à cette attaque, il faut qu'au moins une des deux conditions décrites ci-dessus ne soit pas satisfaite par Eve.

Ce type d'attaque se résume en deux grandes étapes : le moment où nous collectons des données $V_{F_{E^{O_e}}}$ ou $V_{F_{D^{O_d}}}$, après avoir demandé l'exécution de $F_{E^{O_e}}$ ou de $F_{D^{O_d}}$, et la période où nous les analysons. Lors de cette dernière étape, nous cherchons la clé O_i qui maximise la probabilité suivante : $P(O_i|T)$ où T est une mesure sur le comportement du matériel et $P(O_i|T)$ est la probabilité d'avoir O_i, sachant que nous avons eu T.

Nous nous focaliserons surtout sur la partie d'analyse qui est spécifique à un matériel de chiffrement. Ainsi, un désavantage de ce type d'attaque, en comparaison aux autres attaques vues jusqu'à maintenant, est que nous n'aurons pas de solutions d'attaques pour tout matériel de chiffrement ou pour un algorithme de chiffrement. En d'autres termes, nous aurons une attaque moins générique que celles vues précédemment. En effet, un même algorithme peut être implémenté d'une infinité de manières différentes. Toutefois, pour réduire le coût de fabrication, la majorité des constructeurs réalisent un même matériel de chiffrement en plusieurs millions d'exemplaires. Ainsi, une attaque spécifique à un matériel sera réutilisable pour un ensemble de matériaux de chiffrement.

22. Un exemple de comportement est le temps de chiffrement.

De plus, il y a une grande différence entre une attaque telle que nous avons vu jusqu'à maintenant, et celle-ci : le côté pratique. En effet, nous venons de voir un ensemble d'attaques qui ne sont pas toujours réalisables en pratique lorsqu'elle suppose, par exemple, pouvoir stocker plus de 2^{40} clairs nécessitant donc plusieurs terabytes de mémoire. Le reste de ce travail se focalisera sur ce problème. Nous n'accepterons qu'une solution réalisable dans un cas réaliste.

2.7 Conclusion

Ce chapitre a eu comme but de montrer les deux domaines qui forment la cryptologie, à savoir la cryptographie et la cryptanalyse. Le premier nous a permis de connaitre quelques algorithmes de chiffrement de base. Le second nous a permis d'apprendre les techniques d'attaque de ces derniers, en passant par les plus simples jusqu'aux plus complexes.

Le prochain chapitre détaillera l'analyse de canaux auxiliaires.

Chapitre 3

Analyse de canaux auxiliaires

Ce chapitre a comme but d'avoir une vue plus détaillée du monde de l'analyse de canaux auxiliaires. Nous allons commencer par définir quelques termes. Ils nous permettront de classifier l'ensemble des attaques de cette famille. Cette classification nous sera utile pour comprendre l'histoire de cette famille. Par après, nous regarderons les deux grandes écoles qui forment ce domaine, à savoir l'analyse simple du courant et l'analyse différentielle du courant.

3.1 Technique de classification

Selon [21], les attaques de cette famille peuvent être divisées en deux parties : les attaques actives ou passives.

Attaques actives et passives

Dans l'attaque passive, le matériel de chiffrement est utilisé de la manière dont sa spécification le définit. L'unique propriété exploitée est le temps de chiffrement, la consommation d'énergie, le bruit, la température du matériel,... Dans le cas de l'attaque active, le matériel est modifié dans le but que ce dernier révèle la clé utilisée.

Selon [21], les attaques peuvent aussi se diviser en trois autres parties : les attaques invasives, semi-invasives et non invasives.

Attaques invasives, semi-invasive et non invasives

Lors d'une attaque invasive, tout est permis sur le matériel de chiffrement. Ainsi, nous pouvons, par exemple, ouvrir le matériel de chiffrement et y insérer des sondes.

Dans le cas où le matériel fonctionne de manière habituelle, nous parlerons d'attaque passive.

Pour les attaques semi-invasives, nous pouvons ouvrir le matériel de chiffrement, mais nous ne réalisons aucun contact direct avec le matériel. Dans le cas où l'attaque semi-invasive est active, nous réalisons des perturbations sur le matériel sans contact direct avec ce dernier.

Pour les attaques non invasives, l'unique interface accessible sans aucune modification du matériel est utilisée. Ainsi, il est théoriquement complexe de repérer ce type d'attaque puisque ce dernier ne réalise aucun changement du matériel.

3.2 Histoire

Selon [43], le début de l'histoire des analyses de canaux auxiliaires se situe entre la fin du $XIX^{ème}$ siècle et le début du $XX^{ème}$ siècle. En effet, dès la $1^{ère}$ guerre mondiale, des problèmes de bruit furent utilisés dans le but d'espionner des communications.

En 1918, H. Yardley et son équipe ont montré qu'il était possible de découvrir des informations privées s'échappant de matériels électriques. Ces fuites ont été considérées importantes dans les années trentes sur des machines à écrire d'IBM.

Après cela, durant la deuxième guerre mondiale, nous avons pu remarquer une forte volonté de réduire tout rayonnement sur les appareils sensibles. Toutefois en ne se focalisant que sur des fuites d'informations,nous avons aperçu, tout au long du temps, des techniques d'espionnage organisées par différents pays. Le plus connu étant le renseignement d'origine électromagnétique (ROEM [1]) Echelon.

Ce n'est que dans les années cinquante où une grande série de normes militaires américaines et OTAN [2] ont vues le jour sur la limitation des rayonnements électromagnétiques et l'utilisation de blindage électromagnétique. Ainsi, le programme TEMPEST [3], utilisé par la défense américaine, est un ensemble de techniques et de contre-mesures pour protéger ou masquer des signaux et des rayonnements électromagnétiques. L'ensemble de ces techniques d'attaques fût souvent secret, bien que le public aurait pu voir des interférences entre divers appareils électriques.

Ce n'est qu'en 1984 où la NSA [4] publia le concept de zone de sécurité autour d'un point de mesure. Un an après, IBM construit le premier PC [5] blindé contre les fuites électromagnétiques.

D'année en année, des recherches ont eu lieu pour essayer de réduire les fuites d'informations, tant pour la sécurité de l'information que pour celle des personnes utilisant le matériel où une trop grande fuite peut causer des dommages de santé.

1. En anglais, nous parlons de Signals Intelligence (SIGINT).
2. Organisation du traité de l'Atlantique Nord.
3. Telecommunications Electronic Material Protected from Emanating Spurious Transmissions.
4. National Security Agency.
5. Personal Computer.

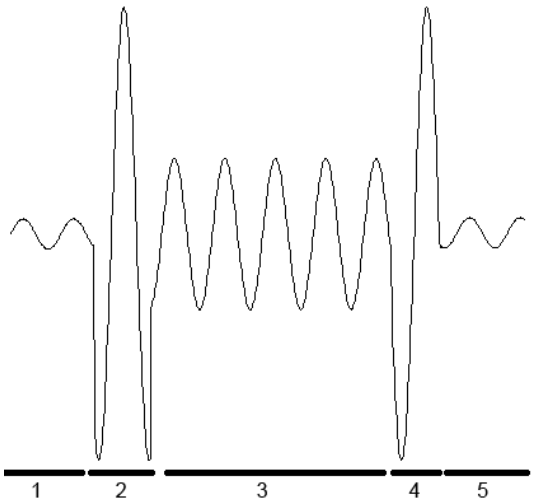


Figure 3.1 : Cette figure illustre qu'une trace contient 5 périodes : le moment d'attente, le moment où le matériel reçoit le message à chiffrer et la clé, le moment de chiffrement, le moment où le matériel renvoie le chiffré et enfin le moment où le matériel attend un nouvel ordre.

En 1996, P. Kocher publia des articles sur le lien entre le temps de calcul et l'information manipulée, ce que nous nommons par attaque temporelle. Par après, des recherches sur l'analyse de consommation par P. Kocher furent aussi réalisées [30].

3.3 Idée générale

Le regard peut donc se porter sur le temps des opérations, le son, la température du matériel, l'émanation électromagnétique ou/et la consommation d'énergie. Notons que le temps des opérations est une information largement utilisable de nos jours dû essentiellement à internet qui permet le chiffrement à distance. De plus, les émanations électromagnétiques permettent de réaliser des attaques à distance. Bien que la solution générale puisse être applicable avec tous ces différents regards, ou une combinaison de ces derniers, ce mémoire se focalisera sur le courant. Ce dernier sera modélisé par une trace $T_{i,1...N}$ qui est un vecteur colonne contenant N valeurs réelles. Nous avons donc un vecteur de variables aléatoires $\mathbf{T}_{.,1...N}$ représentant l'ensemble des traces. Chaque élément du vecteur représente un temps différent. La valeur qui est présente dans cet élément est la consommation d'énergie, en volt, à un instant donné. Comme l'illustre la figure 3.1, la consommation d'énergie pour un matériel chiffrant ont 5 ensembles de temps : le moment d'attente, le moment où le matériel reçoit le message à chiffrer et la clé, le moment de chiffrement, le moment où le matériel renvoie le chiffré et enfin le moment où le matériel attend un nouvel ordre.

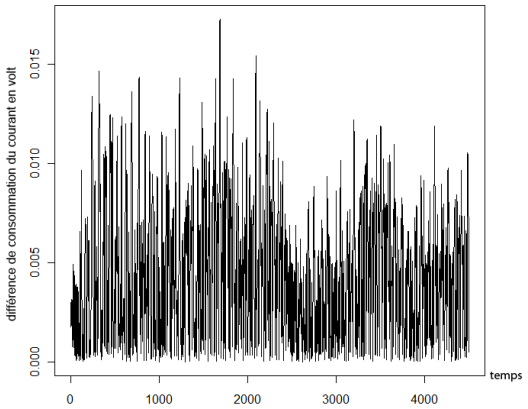


Figure 3.2 : Cette figure montre la différence en valeur absolue de traces lorsqu'un matériel de chiffrement utilise deux clés différentes : 00 et FF. Ces traces proviennent du jeu de donnée réalisé par Stéphane Fernandes Medeiros[21]. Les deux traces utilisées sont celles où nous varions le dernier byte de la première clé 3DES et le reste est inchangé. Notons que nous nous sommes focalisés sur la période de temps où le matériel chiffre les données.

La valeur de la clé utilisée influence directement l'allure de cette trace. Une méthode simple de le vérifier est de réaliser la différence entre deux traces d'un même matériel de chiffrement, l'une avec l'utilisation de la clé où la seule valeur du byte qui varie vaut 00 [6], et une autre où cette valeur est FF [7]. Le résultat se trouve dans la figure 3.2 où le graphique montre la différences des deux traces en valeur absolu. Ces différences montrent clairement qu'avec une clé de chiffrement différente, nous obtenons des traces différentes.

Les attaques se basant sur la consommation d'énergie peuvent se diviser en deux parties :

1. L'analyse simple du courant
2. L'analyse différentielle du courant

3.4 Analyse simple du courant (SPA [8])

L'attaque par SPA essaye de retrouver la clé en se focalisant sur les traces. Elle se base sur le fait que des clés différentes impliqueront l'utilisation d'appels de fonctions différente [9] ou de temps de calcul différent [10], impliquant des consommations d'énergie

6. Ceci est une notation hexadécimale.
7. Ceci est une notation hexadécimale.
8. En anglais, cette analyse s'appelle Simple Power Analysis.
9. Pour accélérer des calculs, des techniques d'optimisation font varier l'utilisation de fonction selon la clé utilisée.
10. Avec des valeurs de paramètres différents, des instructions peuvent avoir un temps de calcul différent.

Figure 3.3 : Cette figure montre les différences entre SPA et DPA.

différentes. Ainsi, cette attaque ne demande pas de connaitre le message chiffré ou le clair.

L'hypothèse de base, lors de l'utilisation de cette technique, est qu'il est nécessaire de bien connaitre le fonctionnement interne de la machine attaquée, ce qui peut être une limitation et est une des raisons qui ont poussé à une focalisation plus détaillée sur d'autres techniques, telle que l'analyse différentielle du courant. De plus, selon [38], les techniques SPA sont plus simples à contrer. Toutefois, si l'hypothèse est satisfaite, cette attaque permet théoriquement, en n'utilisant qu'une seul trace, de récupérer la clé de chiffrement puisque nous supposons connaitre parfaitement le matériel de chiffrement, son implémentation, quand est-ce qu'il fait des jump ou un appel d'une certaine fonction,....

3.5 Analyse différentielle du courant (DPA [11])

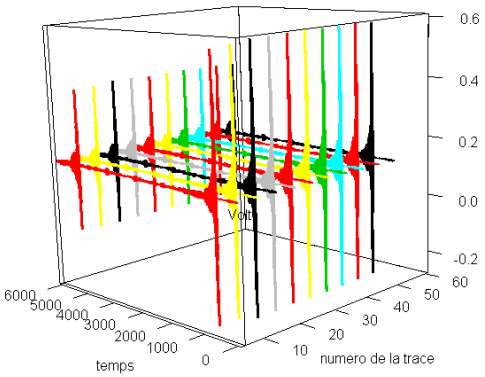
Comme il a déjà été fait mention précédemment, la technique DPA ne présuppose aucune connaissance détaillée sur le fonctionnement interne de la machine. Celle-ci essaye de déterminer un lien entre les données traitées et la consommation d'énergie. Comme nous le verrons par après, une variante de cette technique ne nécessite même pas une connaissance sur l'algorithme de chiffrement utilisé. Toutefois, contrairement à SPA, elle nécessite suffisamment de données pour comprendre le lien effectif entre les données manipulées et le comportement de la machine. En statistique, ce lien s'appelle une dépendance entre deux ensembles de variables, ceux représentant la consommation d'énergie $T_{i,.}$ et ceux représentant la clé O_j. Pour quantifier $P(T_{i,.}|O_j)$, l'attaquant essaye de donner des messages en clair ou en chiffré au matériel de chiffrement et réalise une hypothèse sur la clé utilisée par ce dernier.

La figure 3.3 reprend les différences entre SPA et DPA.

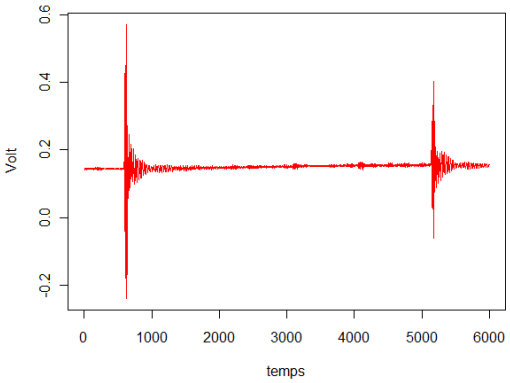
Les graphiques de la figure 3.4 résume l'idée générale des différences entre SPA et DPA. Nous y voyons, au point a. l'ensemble des 10 traces récoltées où chacune d'entre elles a 6000 points, au point b. une focalisation telle que le fait SPA sur les 6000 points d'une seule trace et au point c. une focalisation telle que le fait DPA sur un même instant t ($1 \leq t \leq 6000$) de l'ensemble des 10 traces.

Il existe énormément de variantes de ce type d'attaque telle que le DPA simple, celui généralisé et le *template Based DPA*.

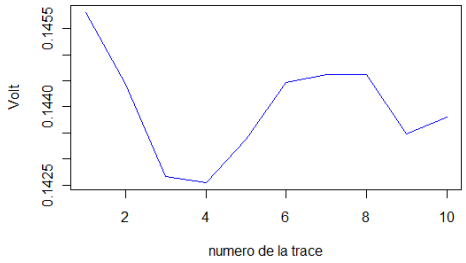
11. En anglais, cette analyse s'appelle Differential Power Analysis.



a. Ensemble de traces



b. Une trace



c. Un instant de toutes les traces

Figure 3.4 : Au point a. il figure l'ensemble des 10 traces récoltées où chacune d'entre elles a 6000 points. Au point b. il figure une focalisation telle que le fait SPA sur les 6000 points d'une seule trace. Au point c. il est illustré une focalisation telle que le fait DPA sur un même instant t ($1 \leq t \leq 6000$) de l'ensemble des 10 traces. Notons que l'ensemble de ces traces sont ceux du $2^{ème}$ byte de la $1^{ère}$ clé 3DES récoltée lors du stage décrit par après.

3.5.1 DPA simple

La première idée venant de Kocher et al [30] est la technique de DPA implémentée contre l'algorithme DES. Cette attaque se base sur quatre étapes :

1. Choix d'une fonction de sélection
2. Collecte de donnée
3. Calcule d'une trace différentielle
4. Attaque par brute force des bits non accessibles

Détaillons chacune des étapes :

Choix d'une fonction de sélection

Nous utilisons une fonction dite de sélection $D(C, b, O_s)$. Il existe énormément d'exemples de telle fonction. Prenons celui où cette dernière calcule la valeur du $b^{ème}$ bit ($0 \leqslant b < 32$) du registre G [12] au début de la $16^{ème}$ étape pour le texte chiffré C [13]. Notons que ce bit b est appelé bit de sélection ou cible. La valeur de O_s ($0 \leqslant O_s < 2^6$) représente les 6 bits de la clé d'entrée de la fonction S-boxes qui ont un lien direct avec le $b^{ème}$ bit. O_s est la clé que nous supposons que le matériel de chiffrement utilise. Elle est donc fixée. De plus, nous supposerons que la valeur du bit de sélection est dépendante de la clé O_s et du texte C. Notons que dans le cas où la clé supposée est effectivement la clé utilisée par le matériel, alors la fonction $D()$ renverra toujours la bonne valeur du $b^{ème}$ bit. Dans le cas contraire, $D()$ renverra une fois sur deux la bonne valeur. La figure 3.5 montre comment nous pouvons remonter, grâce à un chiffré C, jusqu'à la valeur du bit b. Nous y voyons 12 étapes où nous commençons par le chiffré, nous passons par les fonctions FP, E, Xor, S-Boxes, P et enfin Xor pour retrouver le bit b.

Collecte de données

La deuxième étape est la collecte de données. Pour cela, nous utilisons un oscilloscope qui va récolter les traces. Nous collectons n traces, chacune d'elles ayant N points : $T_{1...n,1...N}$. De plus, nous prenons aussi l'ensemble des chiffrés : $C_{1...n}$.

12. Pour rappel, G est illustré à la figure 2.5.
13. Un autre exemple de bit de sélection est la valeur en sortie d'un S-Boxes tel qu'est présenté dans [26].

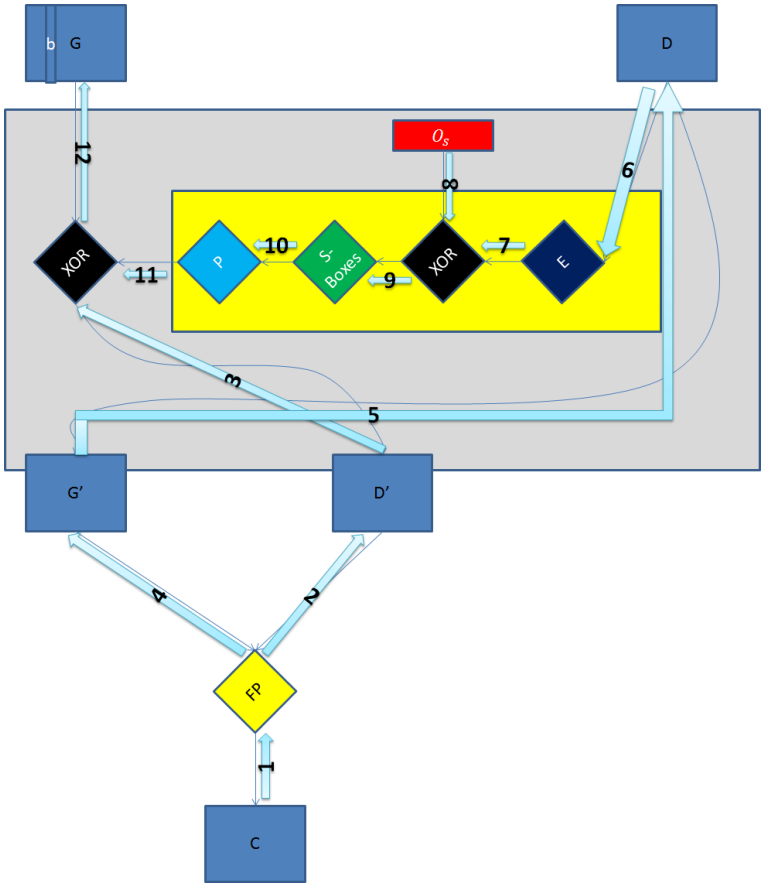


Figure 3.5 : Cette figure illustre les différentes étapes qui permettent de remonter jusqu'à la valeur du bit b. Nous commençons par le chiffré, nous passons par les fonctions FP, E,Xor, S-Boxes,P et enfin Xor pour retrouver le bit b. Les numéros sur les flèches représentent le moment où nous passons d'un lieu à un autre.

Calcul d'une trace différentielle

Par la suite, nous calculons une trace différentielle $\triangle D_{1...N}$. Cette trace représente la différence entre la moyenne $\mu_{1,1...N}$ des traces où $D(C, b, O_i)$ vaut 1 et la moyenne $\mu_{0,1...N}$ des traces où $D(C, b, O_i)$ vaut 0. Formellement, nous avons :

S_0 : l'ensemble des traces où $D(C, b, O_i)$ vaut 0
S_1 : l'ensemble des traces où $D(C, b, O_i)$ vaut 1
$\mid S_0 \mid$: le cardinal de l'ensemble des traces où $D(C, b, O_i)$ vaut 0
$\mid S_1 \mid$: le cardinal de l'ensemble des traces où $D(C, b, O_i)$ vaut 1
$\mid S_0 \mid + \mid S_1 \mid = n$
$\mu_{0,j} = \frac{1}{|S_0|} \sum_{T_i \in S_0} T_{i,j} \quad \forall j \in [1, N]$
$\mu_{1,j} = \frac{1}{|S_1|} \sum_{T_i \in S_1} T_{i,j} \quad \forall j \in [1, N]$
$\triangle D_j = \mu_{1,j} - \mu_{0,j} \quad \forall j \in [1, N]$

Pour vérifier que le choix de la valeur de O_s est exacte, en d'autres termes que notre choix de O_s est aussi le choix du matériel de chiffrement, nous regardons la trace différentielle. Comme nous le verrons par après, la présence de pics dans une trace indique que notre choix de O_s est correct et que de plus, le choix de la fonction de sélection est approprié. Dans le cas contraire, nous devrions avoir une absence de pics, ou plus exactement une courbe plate à un bruit près.

De plus, ces pics montrent où le bit b influence le courant, dans le cas où nous connaissons la valeur de b avec une probabilité 1. En effet, nous supposons qu'au moment où nous manipulons le bit b, nous aurons une consommation proportionnelle à la valeur de ce bit. En d'autres termes lorsque nous avons trouvé la bonne clé, nous supposons que quand ce bit est manipulé et est égal à 1 ou à 0, nous aurons une consommation d'énergie de A ou B respectivement. Plus la différence entre A et B est grande, plus nous aurons un pic visible. Inversement, lorsque nous ne manipulons pas ce bit ou que nous n'avons pas trouvé la bonne clé, nous supposons que la consommation est indépendante de la valeur de ce bit et amène à une différence des $\mu_{i,j}$ proche de zéro et ce, d'autant plus que nous collectons de traces selon [28]. Notons que [28] affirme aussi que $\triangle D_j$ ne convergera pas toujours vers zéro lorsque b n'est pas manipulé, mais ne donnera pas un pic autant que lorsque b est utilisé.

Attaque par force brute des bits non accessibles

Un ensemble de 48 bits de la clé initiale est retrouvable puisque nous cherchons les bytes de la dernière clé intermédiaire. Le reste des bits, soit 8 bits, est cherché grâce à une attaque par force brute.

L'attaque est applicable à tout algorithme de chiffrement manipulant des S-Boxes [14]. Dans le cas précis du DES, cette attaque est applicable pour des clairs ou des chiffrés connus, et ce lors du chiffrement ou du déchiffrement d'un message. Selon les données à disposition, nous pourrons donc réaliser une attaque à texte chiffré connu, à texte clair connu, à texte chiffré choisi ou à texte clair choisi. Pour les lecteurs désirant connaitre la méthode avec des textes en clair et non avec des chiffrés, nous les renvoyons vers [23] qui propose, en plus, de ne se focaliser plus sur un bit mais sur un ensemble. Ce type d'attaque où nous passons d'un bit cible à un ensemble est appelé multi-bit DPA par [24]. Ainsi, au lieu de se focaliser sur un bit b, il se focalise sur 4 bits, amenant à 2^4 groupes possibles. L'intérêt d'une telle méthode est une amélioration des performances, au détriment d'un nombre accru de traces à collecter selon [28].

Selon [1], il est possible de réaliser un *multi-side channel DPA attack* en combinant S sources d'information, tel que le courant, le son, le temps de chiffrement,... Leur résultat est qu'il est plus efficace de prendre en compte un ensemble de sources d'information qu'une seule. Toutefois, si nous restons dans le cas de base, nous constatons des hypothèses à respecter. Selon [12], nous pouvons en constater trois :

1. Les bits du même byte que le bit cible sont supposés avoir la même influence moyenne dans les deux sous-ensembles possibles, l'ensemble où le bit cible vaut 1 et celui où il vaut 0. Ainsi, il n'est pas important de se soucier de ces bits.
2. Lorsque nous réalisons un choix erroné de la clé, la valeur du bit cible ne dépend pas, au sens statistique, de la valeur associée à la supposition correcte de ce bit.
3. Il n'y a aucune dépendance entre la consommation d'énergie et le bit cible quand ce dernier n'est pas manipulé.

Un des désavantages à tout cela est la présence de ce que [12] appelle des pics fantômes. Notons toutefois que [12] suppose que la fonction $D()$ ne renvoie qu'une seule valeur, celle du bit cible. Les pics fantômes sont des pics non négligeables qui pourraient faire croire qu'une certaine clé est celle utilisée par le matériel de chiffrement, alors qu'une autre clé est la bonne. Ce problème amène à une erreur d'identification de la bonne clé. Selon [12], ceci est dû à trois erreurs :

1. Il est inexacte de supposer que le partitionnement des traces, dans chaque sous-ensemble pour une supposition de clé incorrecte, est indépendant de celle correspondant à la bonne supposition. Ainsi, certaines partitions qui ne sont pas bonnes

14. DES et AES sont des exemples d'algorithme qui manipulent des S-Boxes.

ont des similitudes avec la bonne partition. C'est pourquoi nous observons des pics, appelé pics fantômes.

2. Dans la consommation du courant, c'est une erreur d'ignorer les autres bits du même byte que celui du bit de sélection. En effet, la fonction $D()$ ne renvoie qu'une seul valeur, celui du bit cible. Toutefois, la contribution des autres bits accompagnant le bit de sélection peut contrebalancer l'influence, dans la consommation, du bit sélectionné.
3. Il arrive que nous voyons des pics bien que la donnée cible n'est pas manipulée. En effet, certaines données peuvent être corrélées avec le bit cible.

Mise à part ces hypothèses, aucune autre sur la distribution des données doit être réalisée, il suffit que sa consommation diffère, en moyenne, lorsqu'un certain bit de la clé est mis à zéro et lorsqu'il est mis à un.

Désormais, focalisons-nous sur le DPA généralisé qui est une variante de la technique que nous venons de voir.

3.5.2 DPA généralisé

Pour avoir une procédure indépendamment du choix de l'algorithme de chiffrement, [35] montre un algorithme généralisé réalisable en cinq étapes :

1. Choix d'une valeur intermédiaire
2. Collecte des données
3. Calcul des valeurs intermédiaires
4. Lien entre consommation d'énergie et valeur intermédiaire
5. Regarder les résultats possibles avec ceux observés

Détaillons chacune des étapes.

Choix d'une valeur intermédiaire

La première étape demande la recherche d'une fonction F, tel que S-Boxes dans le cas du DES, prenant deux paramètres : une valeur d connue et une valeur O_i. Cette dernière représente une (partie de la) clé. La valeur d représente le texte soit en clair soit en chiffré. Nous allons supposer que l'ensemble des clés possibles est $\mathbf{O}$ de taille $|\mathbf{O}|$ et que nous avons donc $O_{1...|\mathbf{O}|}$.

Collecte des données

La deuxième étape est la demande d'exécution du matériel de chiffrement sur un ensemble de données différentes $P_{1...n}$, en utilisant la clé que nous recherchons. Ces différentes exécutions sont enregistrées sous forme de traces $T_{i,1...N}$ où $1 \leq i \leq n$. Chaque

trace $T_{i,1...N}$ contient N points. En regroupant chaque $T_{i,1...N}$ en un tableau de taille $n \times N$, nous obtenons la matrice $T_{1...n,1...N}$.

Calcul des valeurs intermédiaires

Désormais que nous avons les traces, nous devons construire une matrice $v_{1...n,1...|\mathbf{O}|}$ de taille $n \times |\mathbf{O}|$. Cette matrice contient la valeur $v_{i,j} = F(P_i, O_j)$ où $i = 1...n$, $j = 1...|\mathbf{O}|$. Cette matrice permet de connaitre la valeur que le matériel manipule dans le cas où il a manipulé la valeur de la fonction F. Une des colonnes de $v_{.,.}$ utilise la fonction F avec la même clé qu'utilise le matériel de chiffrement. Cette dernière sera nommée O_{ck}.

Lien entre consommation d'énergie et valeur intermédiaire

Pour réaliser le lien entre la consommation d'énergie et la valeur intermédiaire, il est nécessaire d'utiliser un modèle de consommation M. Ce dernier représente la consommation d'énergie théorique dans le cas où le matériel manipule la valeur intermédiaire. Ainsi, nous obtenons une nouvelle matrice $h_{1...n,1...|\mathbf{O}|}$ où $h_{i,j} = M(v_{i,j})$. Un exemple de modèle M est le bit de poids faible, la distance de Hamming[9, 10] ou le poids de Hamming.

Ainsi, pour modéliser la consommation, nous pouvons réaliser la distance de Hamming entre la donnée écrite sur un emplacement mémoire et la donnée, appelée aussi état de référence, qui était présente avant sur le même emplacement. Ce basculement d'une valeur à une autre amène à une certaine consommation dépendant de ces deux valeurs et peut être calculée grâce à la distance de Hamming. De manière formelle, nous pouvons définir la consommation d'énergie au temps t par :

> $T_t = e \times H(P_{t-1}, P_t) + E$ où e et E sont des constantes dont les valeurs sont dépendantes du matériel analysé, P_{t-1} et P_t sont les données respectivement avant et après l'écriture d'une donnée et H est la distance de Hamming.

L'article [10] donne une distance de Hamming encore plus détaillée en prenant en compte la différence de consommation lorsqu'un bit passe d'une valeur zéro à un et d'une valeur un à zéro :

> $T_t = \sum_{i=1}^{m}(P_{i,t}(1 - P_{i,t-1})c_{01} + P_{i,t-1}(1 - P_{i,t})c_{10} + C)$ où m est le nombre de bit manipulé ; c_{xy} représente la consommation d'un bit qui passe d'une valeur x à une valeur y ; $P_{i,t}$ représente la valeur du $i^{ème}$ bit au temps t et C est une constante représentant la consommation indépendante de la donnée manipulée.

Ce choix de modèle vient du fait que selon [38], le courant est plus dépendant du changement de valeur d'un lieu de la mémoire que de la valeur qu'il contient. La question est de savoir quelle est la valeur qui se trouvait avant l'écriture de la donnée. Nous renvoyons les lecteurs désirant avoir la solution à lire le chapitre 3 dans [38].

Dans le cas où l'état de référence est mis à zéro avant chaque écriture, il n'y a aucune influence entre deux écritures. Dans ce cas, nous utilisons le poids de Hamming où ce dernier est l'ensemble des éléments différents de zéro dans une suite finie. Nous pouvons définir de manière formelle la consommation d'énergie au temps t par :

$T_t = e \times h(P_t) + E$ où e et E sont des constantes dont les valeurs sont dépendantes du matériel analysé et h le poids de Hamming de la donnée P_t manipulée au temps t.

Notons que ce T_t peut être vue en terme de distance de Hamming en :

$T_t = e \times H(P_t, 0) + E$ où H est la distance de Hamming et P_t est la donnée manipulée au temps t.

Grâce au fait que nous tenons compte de plusieurs bits, les pics fantômes vue dans le DPA simple sont atténués.

Regarder les résultats possibles avec ceux observés

Pour finir, il est nécessaire de regarder le lien $r_{i,j}$ entre $h_{.,i}$ avec $T_{.,j}$. En d'autres termes, nous regardons le courant théorique et pratique. Le résultat est mis dans une nouvelle matrice $r_{1...|\mathbf{O}|,1...N}$, de taille $|\mathbf{O}| \times N$, contenant les différents $r_{i,j}$. Le but du jeu est de retrouver le plus grand $r_{i,j}$ représentant la clé recherchée. En effet, un grand $r_{i,j}$ représente une dépendance entre une partie de chaque trace observée et le modèle de consommation qui utilise la clé O_i. Si ce lien est fort, c'est que notre modèle de consommation utilise la bonne clé O_i. Ce résultat nous amène à supposer que O_{ck} est en réalité O_i. Notons que $r_{i,j}$ peut par exemple être un coefficient de corrélation linéaire ou non linéaire.

Une manière simple de visualiser la matrice $r_{.,.}$ est de réaliser des graphiques où la présence de pics montre une forte corrélation entre le modèle de consommation et les traces calculées. Ainsi, nous réalisons $|\mathbf{O}|$ graphiques, une par clé potentielle.

A présent, passons au *template Based DPA*.

3.5.3 Template Based DPA attacks

Le DPA a une variante qui s'appelle le *template Based DPA*. Elle a été présentée dans [11] en 2002. Cette dernière est la plus puissante parmi l'ensemble des attaques connues, si nous respectons certaines hypothèses détaillées par la suite. De plus, elle ne nécessite ni chiffré ni clair, uniquement les traces, ou tout comportement du matériel de chiffrement, lorsqu'une certaine clé est utilisée. Elle essayera de prendre en compte l'ensemble des informations qui se trouvent dans chaque trace pour en déduire la fonction F_{FE} ou F_{FD} vue précédemment.

Cette attaque se base sur le bruit, défini comme étant la différence entre une consommation moyenne de courant et celui pris lors de l'attaque, généré par le matériel de chiffrement pour retrouver la clé utilisée. Ainsi, nous créons un template pour chaque clé

possible. Ce template représente le comportement du matériel lorsque celui-ci utilise une certaine clé. Sachant que le bruit généré par le matériel de chiffrement varie selon la clé utilisée, ce template représente la distribution de probabilité P des valeurs du bruit pour une certaine clé utilisée. Suivant le théorème central limite [15], nous supposons que le bruit produit par la machine suit théoriquement une distribution normale. C'est pourquoi dans la littérature, P suit une distribution normale [16]. Toutefois, notons que certaines données ne suivent pas forcement une distribution normale. De plus, dans des phénomènes réels, un ensemble de données suit rarement totalement une distribution gaussienne. Cette dernière n'étant qu'un artifice pour modéliser, avec très peu de paramètres, un comportement aléatoire. De ce fait, il est toujours nécessaire de vérifier que celle-ci suit une distribution gaussienne avec des tests statistiques tels que Shapiro-Wilk[45].

La méthode habituelle pour réaliser une technique de *template Based DPA* se résume en deux grandes phases [11] :

1. Phase d'apprentissage
2. Phase d'attaque

Détaillons chacune de ces périodes.

Phase d'apprentissage

La première phase est celle d'apprentissage où nous allons chercher à modéliser le comportement du matériel. Elle se fait en six étapes.

- Collecter un certain nombre de traces pour une même clé, et ce pour toutes les clés possibles : $O_{1...|\mathbf{O}|}$. Le but étant de mesurer le comportement du matériel avec une certaine clé.
- Calculer la moyenne des traces d'une même clé, et ceux pour toutes les clés : $\bar{T}_{1...|\mathbf{O}|,.}$. La finalité de cette étape est le calcul du premier paramètre des distributions normales multivariées.
- Réduction éventuelle du nombre de points pris en compte sur la trace avec des techniques qui seront détaillées par la suite. L'intérêt de cette étape est une réduction de la taille du problème. Au final, nous avons des traces avec N instants de chiffrement $(T_{.,1...N})$.
- Pour chaque clé O_i possible, calculer la valeur de la fonction $N_i(T_{j,1...N})$ pour chaque trace $T_{j,.}$ liée à la clé O_i. Pour cela, nous calculons la différence entre une trace $T_{j,.}$ et la moyenne $\bar{T}_{i,.}$. Ainsi, nous obtenons un ensemble de vecteurs $N_i(T_j)$, où $N_i(T_j) = (T_{j,1} - \bar{T}_{i,1}, T_{j,2} - \bar{T}_{i,2}, \ldots, T_{j,N} - \bar{T}_{i,N})$, par clé O_i.

15. Le théorème central limite énonce que tout comportement, causé par un ensemble de phénomènes indépendants et identiquements distribués, est modélisé par une distribution normale.
16. Cette distribution peut être mono variée ou multi variée.

– Pour chaque clé O_i possible, calculer la covariance Cov_i du vecteur des variables aléatoires obtenues grâce à la fonction $N_i(T_{j,1...N})$. Le but de cette étape est la mesure du deuxième paramètre des distributions.
– Avec la covariance et la moyenne, nous créons un modèle paramétrique basé sur une distribution normale multivariée pour chaque clé. Ainsi, pour avoir la probabilité $P_i(T_{j,.})$ de voir une certaine trace $T_{j,.}$ appartenant à la clé O_i, nous calculons la probabilité suivante :

$$P_i(T_{j,.}) = \frac{1}{\sqrt{(2\Pi)^N |Cov_i|}} e^{-\frac{1}{2}(T_{j,.}-\bar{T}_{i,.})^t Cov_i^{-1}(T_{j,.}-\bar{T}_{i,.})^t}$$

Phase d'attaque

Lorsque nous voulons vérifier à quelle clé est rattachée une trace quelconque $T_{j,.}$, nous procédons ainsi :

– Pour chaque clé possible O_i, calculer $P_i(T_{j,.})$
– Choisir la clé O_i ayant la plus grande probabilité $P_i(T_{j,.})$

Nous constatons donc qu'il suffit d'avoir une seul trace pour déterminer la clé de chiffrement.

Exemple

Prenons un exemple simple pour illustrer l'algorithme. Notons que nous avons simplifié l'exemple au maximum pour avoir une compréhension facile. En effet, les données que nous manipulerons ne suivent pas une distribution normale et la quantité d'informations est très faible.

Supposons avoir 2 clés possibles, avec 4 traces collectées pour chaque clé utilisée.

1. Récolte de données

Lors de l'utilisation de la première clé O_1, nous obtenons les traces suivantes :

1	10
2	11
3	9
2	12

Ces traces sont illustrées à la figure 3.6 où nous voyons un graphique pour chaque trace.

Lors de l'utilisation de la deuxième clé O_2, nous obtenons les traces suivantes :

10	1
15	3
9	2
11	4

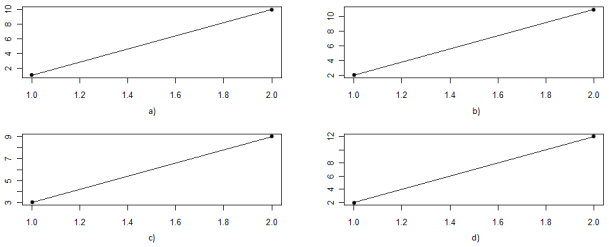


Figure 3.6 : les traces a), b), c) et d) représentent la 1^{er}, $2^{ème}$, $3^{ème}$ et $4^{ème}$ traces lors de l'utilisation de la clé O_1. Notons que les valeurs se trouvant entre les points collectés ont été prédites sur base de ces derniers.

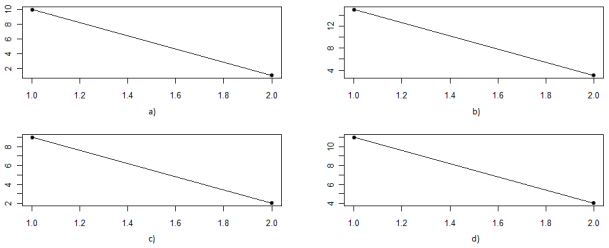


Figure 3.7 : les traces a), b), c) et d) représentent la 1^{er}, $2^{ème}$, $3^{ème}$ et $4^{ème}$ traces lors de l'utilisation de la clé O_2. Notons que les valeurs se trouvant entre les points collectés ont été prédites sur base de ces derniers.

Les quatre traces sont affichées dans la figure 3.7.

Remarquons qu'il semblerait facile de détecter à quelle clé appartient une trace. En effet, les traces liées à O_1 ont des faibles valeurs dans la première variable aléatoire et des fortes valeurs dans la seconde. Les traces liées à O_2 ont des valeurs contraires, de faibles valeurs pour la seconde variable et de fortes valeurs pour la première.

2. Calcul de la moyenne

La moyenne des traces de O_1 est :

2	10.5

La moyenne des traces de O_2 est :

11.25	2.5

3. Réduction de la dimension

N'ayant pas beaucoup de dimensions, nous n'allons pas en réduire.

4.Calcul de $N_i(T_{j,1...N})$

Les valeurs de la fonction $N_i(T_{j,1...N})$ sur les traces de la clé O_1 sont :

-1	-0.5
0	0.5
1	-1.5
0	1.5

Les valeurs de la fonction $N_i(T_{j,1...N})$ sur les traces de la clé O_2 sont :

-1.25	-1.5
3.75	0.5
-2.25	-0.5
-0.25	1.5

5. Calcul de la covariance

La covariance de la clé O_1 est :

0.666666	-0.333333
-0.333333	1.6666667

La covariance de la clé O_2 est :

6.916667	1.50000
1.50000	1.666667

6. Phase d'attaque

Désormais, pour vérifier notre modèle, testons-le avec une trace de test. Soit une trace $T_{1,1...2}$ appartenant à une clé inconnue :

0	10

A première vue, elle semble appartenir à la clé O_1.

Calculons la probabilité de voir apparaitre cette trace $T_{1,1...2}$ si nous utilisons la clé O_1 :

$$P_1(T_{1,1...2}) = 0.003742966$$

Désormais, voyons ce que vaut la probabilité si nous pensons que la trace vient de la clé O_2.

Calculons la probabilité de voir apparaitre cette trace $T_{1,1...2}$ si nous utilisons la clé O_2 :

$$P_2(T_{1,1...2}) = 5.631576 \times 10^{-22}$$

La probabilité que la trace appartienne à la première clé est plus élevée, ce qui confirme notre impression de départ.

Discussion

Nous venons de voir un exemple d'application du *template Based DPA*. Toutefois, dans le monde réel, les données sont beaucoup plus complexes, contenant beaucoup plus de dimensions ainsi que du bruit amenant à la construction d'un mauvais template dans le cas pratique. Le bruit peut être dû à l'oscilloscope, à la cage de Faraday,... C'est pourquoi il est souvent nécessaire de récolter un ensemble de trace, d'un même appareil de chiffrement utilisant une même clé. Ainsi, avec cet ensemble, nous pouvons réduire le bruit en réalisant une moyenne [17]. Cette moyenne est par la suite utilisée comme donnée d'apprentissage, de validation et de test pour construire un modèle. Ce type d'attaque a été réalisé avec succès comme le présente [20].

Toutefois, il est nécessaire de relativiser leur résultat. Ils ont obtenu des bons résultats lorsqu'ils faisaient varier les 6 bits de la clé qui entrent dans la $1^{ère}$ S-Boxes, en laissant le reste des bits de la clé à zéro. En d'autres termes, ils cherchaient la valeur des 6 bits de la clé qui entraient dans la $1^{ère}$ S-Boxes. Néanmoins, lorsque ceux-ci faisaient varier l'ensemble de la clé, les résultats devenaient très mauvais. Pour améliorer les résultats, comme le montre [2], il est possible de combiner le DPA et le *template Based DPA*. Pour cela, ils utilisent le DPA pour construire les templates. En effet, suite à certaine constatation, ils ont remarqué que les pics observés par l'attaque DPA peuvent être utilisés par le *template Based DPA* pour être plus précis lors de la prédiction de chaque bit. Ils ont réussi à réduire l'entropie [18] des 48 bits de la clé d'un simple DES de 16 bits amenant à prédire 32 bits à 1.54%, en ne se focalisant que sur les 32 bits en sortie des S-Box, du premier round.

[2] expose donc qu'il est possible de faire que le DPA va montrer quelles clés sont potentiellement intéressantes. En d'autres termes, nous choisissons les k clés dont les pics sont les plus grands. De plus, nous utilisons le résultat de DPA pour savoir quelles sont les variables intéressantes pour le *template Based DPA*. En effet, nous choisirons les variables où les pics sont présents. De plus, puisque le modèle peut être résumé en une fonction qui renvoie la clé selon la trace que nous lui donnons en paramètre, nous pouvons généraliser ce concept, pour passer à un « *Multi-Channel Attacks* »[1], en mettant en entrée non seulement des traces, mais aussi le temps de chiffrement, le message chiffré, le message claire, le son,...

Pour mettre cela en pratique, [1] utilise le maximum de vraisemblance pour inférer les paramètres de la distribution normale multivariée des données, selon la clé utilisée. Leurs résultats sont très bons, moins de 10% d'erreurs. Toutefois, ils ne s'attaquent qu'à un bit pour montrer que leur technique est meilleure que lorsque nous utilisons une seule source d'informations.

17. [28]montre des méthodes plus avancées, basée sur le *North filter*, pour réduire le bruit.

18. La perte de l'entropie d'un ensemble de clés peut être vu comme une réduction du coût de calcul d'une recherche par force brute.

3.6 Conclusion

Ce chapitre nous a permis d'avoir une vue détaillée sur l'analyse de canaux auxiliaires, et plus précisemment sur le DPA et le *template Based DPA*. La suite du travail montrera en quoi est-ce utile de dériver une nouvelle technique de ce dernier où nous le combinons avec de l'apprentissage automatique. C'est pourquoi le prochain chapitre traite de l'apprentissage automatique pour mettre l'ensemble des lecteurs au même niveau.

Chapitre 4

Apprentissage automatique

Ce chapitre a été inspiré par [7]. Il a comme but d'expliquer et d'analyser le domaine de l'apprentissage automatique pour tirer de l'information à travers des données. Nous allons d'abord présenter le problème en général de l'apprentissage automatique, puis le formaliser, et enfin passer l'ensemble des étapes pour la construction d'un modèle. Il nous permettra de construire un modèle pour attaquer un matériel de chiffrement lors des prochains chapitres.

4.1 Problème

Nous voulons déduire de l'information à partir de données. Les données manipulées viennent d'un problème. Un exemple simple de problèmes est le fait de pouvoir inférer, estimer, déterminer la clé utilisée, en se focalisant sur la consommation d'énergie par le matériel de chiffrement. Notons que lors de la mesure de la consommation d'énergie, les éléments se trouvant aux alentours du matériel de chiffrement influencent les données prises, ce qui est vue comme du bruit sur ces dernières. La donnée est donc une combinaison de deux variables aléatoires : la trace et la clé utilisée.

Le problème est de déterminer le lien, l'information, la dépendance, la loi entre ces deux variables aléatoires. Pour cela, il faut se focaliser sur la racine du problème. Nous pouvons remarquer que l'ensemble des applications de l'apprentissage automatique ont un problème en commun : l'incertitude de la prise de décision. En effet, au vue des variables d'entrée telle que la trace, appelée aussi variable descriptive ou observée, nous ne pouvons être certains sur la variable de sortie telle que la clé, appelée aussi variable cible ou d'intérêt.

Il ne suffit donc pas de faire du pattern matching entre une trace $T_{1,1...N}$, où la clé est inconnue et une autre $T_{2,1...N}$, où la clé est connue pour savoir si $T_{1,1...N}$ a la même clé que $T_{2,1...N}$. Toutefois, il est nécessaire de prendre une décision, de déterminer quelle clé a été utilisée. Pour représenter l'incertitude, nous allons utiliser le formalisme du domaine de

la probabilité tel qu'il est donné par l'apprentissage automatique. Ainsi, il faudra définir les différentes possibilités de réalisation et la loi qui les gouvernent. La connaissance de l'incertitude sera définie par la distribution conditionnelle entre les variables cibles et celles observées. Pour prendre une décision, qui est représentée par la variable d'intérêt, nous prendrons la variable cible qui maximise cette probabilité.

Pour comprendre ce mécanisme, nous allons formaliser l'apprentissage automatique au prochain chapitre.

4.2 Formalisation du problème

Pour réaliser un processus d'apprentissage d'un problème, nous utilisons un ensemble de n observations du phénomène $D_n = \{(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)\}$ où nous connaissons les variables observées ($\mathbf{X}$) et celles cibles ($\mathbf{Y}$). Nous allons supposer que les données sont finies, indépendantes et identiquement distribuées (i.i.d). En d'autres termes, ils ont la même loi de probabilité et sont mutuellement indépendants. Cette hypothèse est nécessaire dans le domaine étudié de l'apprentissage automatique. De plus, les variables descriptives seront considérées comme multidimensionnelles, et celles cibles comme scalaires.

Il ne reste plus qu'à créer un objet, un modèle, un estimateur qui va apprendre et décrire formellement le problème en se focalisant sur les données d'apprentissage D_n. Il permet, selon les données que nous observons, de prédire celles inconnues ainsi que de mieux comprendre et d'analyser celles existantes. Dans le cas d'une prévision, le modèle va au mieux généraliser le problème, en représentant ou en approximant les données connues. La dépendance, entre les deux ensembles de variable $\mathbf{X}$ et $\mathbf{Y}$, est supposée influencée par une loi $\mathbf{Y} = f(\mathbf{X})$ inconnue mais réelle.

Le but de l'apprentissage automatique est donc de comprendre, d'approximer cette loi sur base de données, qui sont souvent bruitées, dans le but d'une prise de décision sur des données inconnues lors de la construction d'un modèle $\mathbf{Y} = h(\mathbf{X}, \alpha)$ où α est le paramètre du modèle. Le bruit [1] w est toute valeur non prévisible avec les informations collectées et qui fait varier la valeur sortie d'un phénomène pour une même entrée à celui-ci. En d'autres termes, nous avons que $\mathbf{Y} = f(\mathbf{X}) + \mathbf{w}$. Il est dû au fait que nous ne pouvons qu'approximer la réalité. Nous aurons donc que la prévision ne sera pas toujours la bonne amenant à un taux d'erreurs non nul. Ce dernier sera aussi dépendant de l'hypothèse de base qui est fait par le modèle et par le problème.

L'apprentissage automatique étant un domaine très large, nous allons nous focaliser sur un type spécifique de problème. Ce type, appelé problème de classification supervisée non paramétrique, sera celui nécessaire pour comprendre le reste de ce travail. Ce groupe réunit l'ensemble des problèmes tel que les variables cibles sont des valeurs finies dans les

1. Il est commode de supposer que le bruit a une moyenne nulle, et est indépendant de la variable X.

entiers (classification) connus lors de la construction du modèle (supervisée) et dont la forme de loi qui gouverne les données est inconnue (non paramétrique).

Le cycle de vie d'une implémentation de l'apprentissage automatique est la suivante :

1. Obtention des données et prétraitement
2. Réalisation du modèle
3. Phase d'apprentissage
4. Phase de validation
5. Phase de test
6. Phase d'exécution

4.3 Obtention des données et prétraitement

La première étape à réaliser est donc l'obtention de données en suffisance, représentatives du problème à résoudre. Ceci n'est pas toujours aisé. Certaines informations sont plus coûteuses à obtenir que d'autres. Par exemple, le courant d'un matériel qui chiffre les données est plus complexe à obtenir que le temps de chiffrement.

Après la récolte, nous réalisons le nettoyage, appelé aussi prétraitement, de la donnée récoltée, c'est-à-dire une réduction de ce qui est strictement intéressant, ainsi que leur traduction. Le but de cette étape est une meilleure précision du modèle, une optimisation de son temps d'exécution et de son apprentissage ainsi que de sa taille. Voici quelques exemples de nettoyage :

1. Normaliser un ensemble de nombres
2. Discrétiser un ensemble de nombres
3. Ajouter des valeurs qui ont été calculées à partir des données récoltées
4. Sélectionner un résumé des informations

Dans le cas d'une mise en pratique d'une attaque se focalisant sur le courant, l'oscilloscope calculant ce dernier renvoie un fichier contenant tout un ensemble de données non nécessairement importantes telles que la date et l'heure de la prise de mesure. De plus, dans le cas où les calculs se font dans un langage particulier, tel que R, il est nécessaire de formater les données pour que ces dernières soient lisibles par le langage. Ce nettoyage est souvent très compliqué à mettre en œuvre et demande une bonne connaissance des données à traiter. C'est pourquoi des techniques automatiques ont fait l'objet de recherches pour diminuer la dimension des données : ce sont les techniques de sélection de variables.

Sélection de variables

Les techniques de sélection de variables ont comme but de réduire le nombre total de variables descriptives à prendre en considération, d'augmenter la facilité de visualisation

et de la compréhension des variables, de réduire la quantité de données, de réduire le temps général du problème tout en gardant une certaine qualité d'estimation. En d'autres termes, nous allons diminuer la dimensionnalité du problème. Toutefois, nous remarquons une augmentation de la lenteur du temps total de calcul pour certains cas, puisqu'il faut rajouter la phase de sélection de variables.

Il existe essentiellement trois écoles. Nous retrouvons d'abord les « *Filter methods* » [2], qui ne se basent que sur l'utilité d'une variable sans tenir compte de son impact dans le modèle. Son avantage est un unique calcul à réaliser, et ce indépendamment du choix ultérieur du modèle d'apprentissage. Ensuite nous avons les « *Wrapper methods* » [3] qui tiennent compte de l'algorithme d'apprentissage pour déduire l'apport des variables d'observation sur celles cibles. Ces techniques sont généralement lourdes. Toutefois, ils permettent de tenir plus facilement compte de l'ensemble de variables d'entrée, en d'autres termes les dépendances entre celles-ci. Enfin, nous distinguons aussi les « *Embedded methods* » [4] qui sont spécifiques à un modèle et sont exécutées lors de la procédure d'apprentissage. Nous ne pouvons donc pas généraliser une de ces méthodes pour un ensemble de modèles. Toutefois, ils sont généralement moins gourmands en calcul que les « *Wrapper methods* ».

Notons que ces méthodes peuvent être combinées pour obtenir de meilleurs résultats. De plus, il est conseillé de ne pas utiliser les mêmes données pour les phases de sélection de variables et d'évaluation, pour éviter un biais au niveau des performances estimées du système.

La première idée pour faire de la sélection de variables est de voir la variation d'une variable, selon la clé utilisée. Ainsi, nous prenons les variables qui varient le plus, et ont donc une grande variance, lorsque nous prenons des clés différentes. Toutefois, il existe des techniques automatiques pour réaliser le choix de variables.

2. Tel que la méthode « Principal Component Analysis » (PCA), le *clustering* (SOM, K-means,...) ou le *ranking*.

3. Tel que la méthode « *Forward selection* », la méthode « *Backward selection* » et la méthode « *Stepwise selection* ».

4. Tel que les arbres de décision et les forêts aléatoires.

Le mémoire ayant utilisé que 4 types différents, nous n'en détaillerons que les suivants :
- PCA [5]
- mRMR [6]
- Self Organizing Map
- Ranking

PCA

Le PCA fût inventé par Karl Pearson[40] et essaye de réduire la dimension du problème en réalisant une projection vers une nouvelle base orthonormée de plus petite taille où chaque nouvelle variable v_i est appelée composante principale et n'est corrélée à aucune autre nouvelle variable. Toutefois, ces dernières ne sont qu'une combinaison linéaire des anciennes variables. Elles représentent des nouveaux axes qui sont définis selon la valeur des vecteurs propres de la matrice de covariance des données qui sont préalablement normalisées [7]. Ces vecteurs propres peuvent être vus comme des hyperplans qui coupent au mieux les données [8]. La projection d'une ancienne observation $x_{j,1...N}$ contenant N variables sur une nouvelle variable v_i où le vecteur propre correspondant est $v'_{i,1...N}$ et la valeur propre est l_i, se fait selon la formule suivante : $v_i = \frac{x_{j,1} \times v'_{i,1} + x_{j,2} \times v'_{i,2} + ... + x_{j,N} \times v'_{i,N}}{\sqrt{l_i}}$. Cette projection se fait sur l'ensemble des nouvelles variables. Ces dernières sont ordonnées : la première contient la plupart de l'information permettant de distinguer les traces alors que la dernière en contient le moins. En d'autres termes, les premières ont la plus grande variance par rapport aux dernières : $\sigma^2_{v_1} \geq \sigma^2_{v_2} \geq ... \geq \sigma^2_{v_N}$. Ce tri se réalise en ordonnant les valeurs propres représentant cette quantité d'informations. Ainsi, en prenant les k plus grandes valeurs propres, nous avons une certaine perte d'informations qui se réduit lorsque k augmente et surtout lorsque les valeurs propres deviennent très petites. Et c'est en prenant une faible valeur pour k que nous pouvons réduire la taille du problème.

Par exemple, supposons que nous avons des données à 3 dimensions. Nous cherchons la matrice de covariance (Cov) de ces données ainsi que les vecteurs propres et les valeurs propres de Cov. Supposons que les valeurs propres sont 1.454, 1.232 et 0.314. Le premier vecteur propre explique 48 % ($=\frac{1.454}{1.454+1.232+0.314}$) des variations totales des données, le deuxième explique 41 % des variations et le reste est expliqué par le dernier vecteur propre. Ainsi, nous remarquons qu'en projetant nos données à trois dimensions vers deux dimensions, nous avons déjà 89 % de l'information initiale qui pourrait être suffisante pour résoudre le problème.

5. Principal Component Analysis.

6. minimum Redundancy Maximum Relevance.

7. C'est-à-dire que les données sont centrées et réduites telles que pour tout $X_{i,j}$, nous obtenons un nouvelle donnée valant $\frac{X_{i,j}-\mu_j}{\sigma_j}$ où μ_j représente la moyenne de la colonne (ou variable) j et σ_j représente l'écart type de la colonne j.

8. Pour couper au mieux les données, nous essayons de maximiser la variance de chaque nouvelle variable.

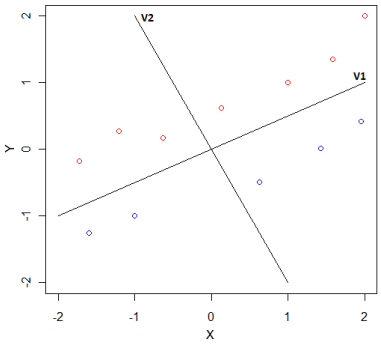


Figure 4.1 : Cette figure montre un ensemble de points où chaque point appartient à une certaine classe déterminée par la couleur du point. Les axes V1 et V2 sont respectivement le premier et deuxième nouvel axe calculé grâce à PCA.

Un de ses désavantages est qu'il ne tient pas compte de l'algorithme d'apprentissage du modèle ni de la classe de chaque donnée, représentée par la clé, contrairement à mRMR que nous'on verrons par la suite. En effet, comme nous pouvons le constater sur la figure 4.1, la meilleure variable selon PCA, celle qui a la plus grande variance, est la première (V1) alors que c'est la deuxième variable (V2) qui permet une distinction des données de classes différentes.

Par ailleurs, la méthode permet de stocker les nouvelles données dans un espace mémoire plus petit. Le PCA peut donc être vu à la fois comme un compresseur de données et un réducteur de bruit où ce dernier est un nombre trop important de variables pour représenter notre problème au modèle d'apprentissage. Notons que beaucoup d'articles parlent d'une combinaison entre le PCA et le DPA [9] appelé aussi *principal subspace-based TA* (PSTA) par [3]. L'article [25] réalise une comparaison entre trois types de réducteurs de dimension - le PCA, une technique se basant sur la moyenne et la corrélation de Spearman - contre l'algorithme AES-128 [10] avec la *template Based DPA*. PCA est montré comme amenant les meilleures prévisions.

mRMR

Le mRMR, défini dans [41], est une technique qui trie les variables de celles ayant le plus d'informations avec la cible, à celles en ayant le moins, tout en retirant celles qui sont redondantes. Ainsi, les k premières variables sont celles ayant le minimum de corrélation entres elles, permettant d'avoir le moins de redondance, et le maximum de dépendance avec la classe. Pour déterminer ce lien, cette dépendance, nous utilisons l'information

9. Pour les lecteurs intéressés, les articles suivant parlent de cette combinaison : [20].
10. L'AES-128 est l'AES où nous utilisons une clé de 128 bits.

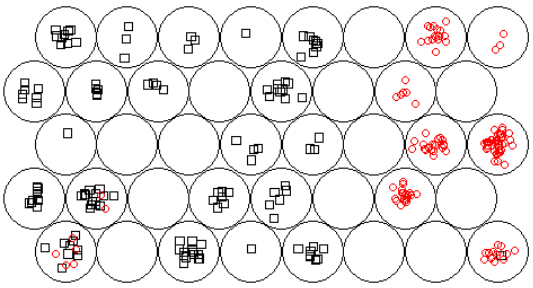


Figure 4.2 : Un exemple de SOM de taille 8×5 où le modèle essaye de partitionner les traces. Il se réalise sur le 1^{er} bit du dernier byte de la $1^{ère}$ clé du 3DES récolté par [21]. Ainsi, nous obtenons deux types de traces, ceux ayant le bit à zéro et ceux ayant le bit à un. Chaque type est différentiable grâce à une couleur et une forme différentes.

mutuelle [11]. Cette recherche se fait en $O(N^k)$ [17] où N est le nombre de variables à analyser. En pratique, nous pouvons utiliser des algorithmes heuristiques pour trouver une solution approchée qui en fait un algorithme rapide pour une petite valeur de k.

SOM

Le SOM [12] ou SOFM [13] est un réseau de neurones non supervisé feedforward à une couche, permettant de mieux visualiser les informations données en entrée sur une carte. Nous obtenons une vue respectant la distance entre les informations initiales. C'est-à-dire qu'un ensemble d'informations différentes sera visiblement séparé sur la carte alors que des données semblables seront visiblement proches. En d'autres termes, nous réalisons une projection des données sur de nouvelles dimensions. Ce modèle est souvent utilisé pour cartographier des données de grandes dimensions en une, en deux ou en trois dimensions. La figure 4.2 résume cette idée où nous projetons des traces en deux dimensions.

L'apprentissage est de type compétitif, ce qui signifie que les neurones sont en compétition pour représenter les données. Ainsi, ils se spécialisent pour certains types de données. L'augmentation du nombre de neurones peut améliorer la précision du réseau mais accroît aussi drastiquement le temps d'exécution. De plus, un de ses points faibles est son temps d'exécution pour des données de grandes dimensions puisque la distance entre tous les neurones doit être calculée pour chaque input. Par ailleurs, lors de l'apprentissage, les connexions du neurone gagnant et des neurones avoisinants doivent être modifiées.

11. Notons que le mRMR utilisé dans ce mémoire fait l'hypothèse de linéarité des données et de ce fait utilise la corrélation.
12. Self Organizing Map.
13. Self Organizing Feature Maps.

RANKING

La méthode de ranking choisit les k meilleures variables d'observation. Ces variables sont celles ayant la plus grande dépendance, corrélation monovariée avec les variables cibles. Cette méthode est rapide et s'exécute en $O(N)$ où N est le nombre de variables. Toutefois, elle ne tient pas compte des redondances entre les k meilleures variables ni du fait qu'une combinaison de variables permet parfois une meilleure estimation de celles cibles qu'une toute seule. Ce dernier est un problème général des techniques monovariées.

4.4 Réalisation du modèle

Cette étape consiste en une recherche de la meilleure structure $h(\mathbf{X}, \alpha)$ ainsi que l'ensemble des paramètres à initialiser dedans. La complexité, et plus précisément la qualité, du modèle aura une influence directe sur la précision de la généralisation des données. Plusieurs types de modèles vont être présentés dans la suite. Avant cela, il reste à déterminer la prochaine étape. Après avoir construit le modèle, il est nécessaire de le configurer selon le problème à traiter grâce à la phase d'apprentissage. En d'autres termes, nous cherchons la meilleure valeur de α durant la phase d'apprentissage.

4.5 Phase d'apprentissage

Un sous-ensemble de l'ensemble des informations nettoyées forme les données d'entrainement, permettant d'exécuter la phase d'apprentissage du modèle. Ceci permet d'ajuster le paramètre α du modèle. Cette étape s'appelle aussi l'identification paramétrique. Elle peut être unique ou périodique, bien que nous allons nous focaliser sur le premier.

Il existe plusieurs sortes d'algorithmes d'apprentissage. Dans la majorité des cas, l'entraînement a comme but de réduire l'erreur de prédiction sur les données d'apprentissage. Elle se fait en utilisant des méthodes mathématiques, tel que les algorithmes génétiques.

Après la phase d'apprentissage vient celle de validation.

4.6 Phase de validation

Désormais que nous avons un ensemble de modèles appris avec un ensemble d'apprentissage, nous devons pouvoir les évaluer pour choisir le meilleur, ce que nous appelons l'identification structurale. Le meilleur modèle étant celui qui généralise au mieux le problème, qui prédit au mieux les cibles inconnues. Nous sommes donc face à deux étapes lors de cette phase : une étape où nous évaluons la qualité du modèle et une autre où nous comparons les modèles entre eux.

4.6.1 Evaluation de la qualité du modèle

Théoriquement, pour une certaine valeur X en entrée, nous pouvons calculer l'erreur moyenne, appelé Mean Square Error (MSE) que le modèle réalise. Toutefois, en pratique, nous ne sommes pas interressés de minimiser le MSE mais le Mean Integrated Square Error (MISE) ou erreur de généralisation. Ce dernier calcule l'erreur moyenne sur l'ensemble des X.

En revanche, nous n'avons que rarement l'ensemble des X possibles. Nous n'avons qu'un seul ensemble fini D_n en pratique. Par conséquent, nous allons essayer d'évaluer le MISE. Pour certain problème simple, le MISE peut être évalué avec précision. Néanmoins, quand nous travaillons avec un problème complexe, nous ne possédons plus de formes analytiques pour l'évaluer. Ainsi, nous pourrions utiliser l'erreur que le modèle réalise sur les données qu'ils connaissent, ce que nous appelons l'erreur empirique. Cette technique permet de voir si le modèle a correctement appris ce que nous lui avons donné. Toutefois, ce n'est pas une bonne méthode pour estimer avec précision l'efficacité de nos modèles. [7] montre que cette valeur est biaisée pour estimer le MISE. L'intuition de cela est qu'un modèle qui a appris par cœur les données d'apprentissage sera supposé comme ayant une faible erreur. Or, ce n'est pas forcément un bon estimateur avec des exemples qu'il ne connait pas. Cela peut se voir avec l'analogie de l'élève qui apprend par cœur les exemples d'un cours. Il semble évident que ce n'est pas en réalisant cela qu'il réussira l'examen.

Nous constatons donc qu'il ne faut ni apprendre, par cœur, les données, ni être dans une situation d'anarchie où le modèle n'est pas influencé par les données d'apprentissage. Le premier cas s'appelle le sur-apprentissage et le second le sous-apprentissage. Dans ces deux situations, nous avons un MISE élevé dû au fait que le modèle n'a pas compris la dépendance entre les variables d'entrée et de sortie.

Pour pouvoir être en sur-apprentissage et donc apprendre, par cœur, les données, nous devons avoir un modèle de complexité élevée. En d'autres termes, il doit prendre en compte l'ensemble des variables d'entrée. Inversement, pour être en situation de sous-apprentissage, il faut prendre en compte aucune variable d'entrée tout en ayant la complexité la plus faible. Il en résulte qu'il faut faire un compromis entre prendre en compte trop ou trop peu de variables d'entrée pour atteindre le minimum du MISE, ce qui est illustré à la figure 4.3.

Pour réduire le MISE, les techniques de sélection de variables peuvent donc être appliquées. En effet, cette réduction amène à une réduction de la complexité du problème. Les sous figures de la figure 4.4 résume l'ensemble de ces idées. La sous-figure de gauche montre un modèle qui a appris par cœur les données alors que celui de droite n'a presque rien appris. Dans les deux cas, dû au bruit sur les données d'apprentissage, aucun des deux modèles sera précis pour représenter le problème, à savoir représenter la fonction $Y = \sin(X)$.

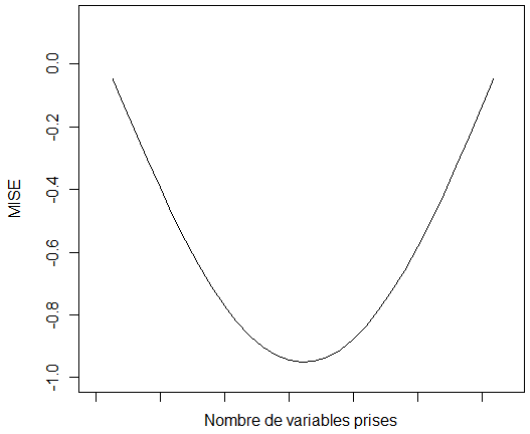


Figure 4.3 : Illustration du fait qu'il faut faire un compromis entre prendre en compte trop ou trop peu de variables d'entrée pour atteindre le minimum du MISE.

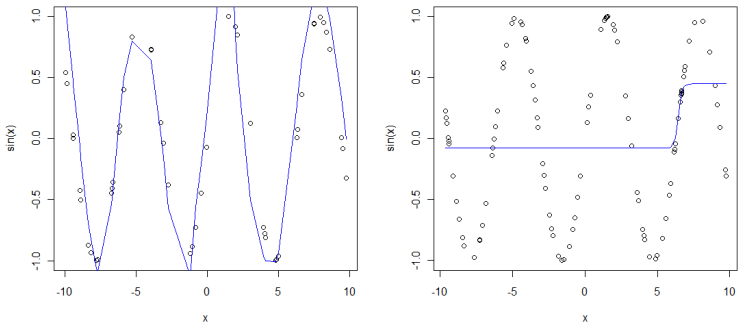


Figure 4.4 : Supposons vouloir apprendre la fonction $Y = \sin(X)$. Les points représentent les données bruitées à apprendre. Les modèles sont représentés par une ligne bleu. La figure de gauche montre un modèle qui a sur-appris alors que celui de droite a sous-appris.

Pour évaluer le MISE, la phase de validation divise aléatoirement nos données D_n en deux groupes distincts. L'un permet de réaliser la phase d'apprentissage et l'autre celle de validation permettant d'évaluer la précision du modèle. En d'autres termes, cette dernière permet d'évaluer la capacité de généralisation de l'estimateur.

Voici quelques méthodes pour la validation :

1. *Test* : Nous prenons un ensemble de données D'_n indépendantes de celle d'apprentissage et nous regardons le taux d'erreurs sur D'_n. Toutefois, nous avons rarement un tel ensemble en pratique.
2. *Hold-out* : Nous coupons une fois aléatoirement l'ensemble des informations en deux groupes mutuellement exclusifs : groupe d'apprentissage et groupe de validation. Cette méthode a comme désavantage que les données d'apprentissage vont influencer directement le taux de bonne prévision lors de la phase de validation.
3. *Leave-one-out* : Cette méthode, appelée aussi *jacknife*, sort de l'ensemble des informations une donnée en particulier et la laisse de côté, puis construit le modèle avec celles restantes et enfin évalue la structure avec l'exemple laissé de côté. Nous répétons le processus pour chacune des données de l'ensemble de données. Pour une meilleure compréhension, l'algorithme 4.1 représente ce processus de validation.

Algorithme 4.1 : Le *leave-one-out*.
Soit $\mathbf{T} = T_{1...n}$, l'ensemble des traces pour l'apprentissage
Soit $i = 1$
Temps que $i \leq n$, faire :
  $\mathbf{T} = \mathbf{T} \setminus T_i$
  Apprendre au modèle l'ensemble $\mathbf{T}$
  Obtenir et comparer le résultat prédit par le modèle sur la trace T_i
  $\mathbf{T} = \mathbf{T} \cup T_i$
  $i = i + 1$

Nous pouvons montrer que ce type de méthode est très efficace pour déterminer la qualité d'un modèle avec beaucoup de données tout comme avec peu de données. Toutefois, il nécessite un temps de calcul conséquent.

4. Validation-croisée : Cette méthode réalise un partitionnement des données de manière aléatoire en k groupes mutuellement exclusifs et approximativement [14] de même taille. Nous utilisons une partition comme un ensemble de validation et le reste pour former celui d'entraînement. Comme précédemment, nous appliquons un algorithme à l'ensemble d'entraînement et nous évaluons le modèle résultant sur celui de validation. Nous répétons ce processus pour chaque partition et nous regardons l'erreur moyenne. Notons que le validation-croisée est le cas général du leave-one-out.

14. Une approximation est nécessaire lorsque n modulo k n'est pas nul.

4.6.2 Comparaison des modèles entre eux

Apres avoir choisi et évalué notre modèle, il est intéressant de pouvoir quantifier ses performances. Pour cela, nous comparons plusieurs modèles sur un même jeu de données. En effet, certains sont plus adaptés pour certains problèmes. Pour cette phase, nous nous intéressons au pourcentage de vrais positifs [15], de vrais négatifs [16], de faux négatifs [17] et enfin de faux positifs [18]. Le meilleur modèle à choisir est le modèle ayant le moins d'erreurs, qui minimise le MISE mesuré lors de la phase de validation. Cette technique est appelé le Winner-Take-All.

Toutefois, ce choix n'est pas l'unique. Au lieu de choisir le meilleur modèle, nous pouvons réaliser une combinaison d'estimateurs dans le but d'améliorer le résultat final. Cette réduction crée une diminution du MISE. Ce type de raisonnement est la base d'une méthode qui s'appelle le bagging ou bootstrap aggregating. Il permet non seulement de réaliser du calcul parallèle mais aussi d'améliorer les performances de structures instables. Une structure est dite instable quand un changement de l'ensemble des données d'apprentissage amène à un changement des réponses émises par cette structure pour le même problème. Pour l'implémenter, il faut créer un ensemble de modèles où chaque composante a un ensemble d'apprentissage bootstrapé. Ce dernier représente un tirage aléatoire avec remplacement parmi l'ensemble d'apprentissage initial. Ainsi, chaque modèle aura un ensemble d'apprentissage légèrement différent. L'ensemble des modèles forme l'estimateur final où nous appliquons un système de vote entre les modèles pour déterminer la classe d'une observation.

Après avoir appris et choisi le meilleur modèle, il reste encore à évaluer avec une meilleure précision ce dernier.

4.7 Phase de test

Au final, il faut être capable d'évaluer la performance du modèle choisi avec de nouvelles données.

En effet, nous ne sommes pas à l'abri d'un sur-apprentissage lors du choix du modèle. C'est pourquoi il faut utiliser un nouvel ensemble de données, appelé ensemble de tests, et évaluant avec plus de précision le modèle final.

Après cette étape, il reste à mettre en production le modèle, à passer à la phase d'exécution.

15. Le modèle a prédit correctement zéro.
16. Le modèle a prédit correctement un.
17. Le modèle a prédit zéro alors qu'il aurait dû prévoir un.
18. Le modèle a prédit un alors qu'il aurait dû prévoir zéro.

4.8 Phase d'exécution

Lors de cette dernière phase, nous reprenons l'ensemble des données, ceux d'apprentissage, de validation et de tests pour construire le modèle choisi lors de la phase de validation. Par après, nous l'utilisons pour de nouveau cas en pratique, des cas non forcement vus lors des phases précédentes.

4.9 Types de modèle

Il existe deux grands types de modèle : ceux qui sont paramétriques et ceux qui ne le sont pas. Les estimateurs paramétriques sont ceux qui réalisent une hypothèse sur la distribution de données. Les autres n'en font pas. Nous n'allons étudier que des modèles non paramétriques. De plus, nous pouvons encore faire une distinction entre les modèles supervisés et ceux qui ne le sont pas. Les premiers sont ceux où la variable cible est connue lors de la période d'apprentissage, les autres où cette information est inconnue. Dans notre cas, puisque nous connaissons la variable cible, nous nous focaliserons sur les modèles supervisés. En outre, nous pouvons structurer les modèles supervisés non paramétriques en modèle de classification ou de régression. Le premier étant un problème où la variable cible est un entier, l'autre étant celui où cette variable est un réel. Sachant que nous devons décider à quelle clé appartient une trace, nous nous trouvons dans un problème de classification. Finalement, pour une raison de temps, nous allons nous concentrer sur les modèles non linéaires puisque ceux linéaires ne résolvent qu'une faible quantité de problèmes. De ce fait, pour ne pas surcharger ce document, nous nous focaliserons uniquement sur le modèle non linéaire, non paramétrique, supervisé et de classification. La liste exhaustive de ces modèles ne peut se faire au sein d'un mémoire. Toutefois, nous allons en détailler quatre. Les modèles étudiés sont :

1. Réseaux neuronaux
2. Machine à vecteur de support
3. Arbre de décision
4. Forêt aléatoire

4.9.1 Réseau neuronal

Les réseaux neuronaux sont inspirés par le cerveau humain. Ils permettent d'approximer toutes les fonctions f, ainsi que l'énonce le principe d'approximation universelle. Ils sont largement utilisés grâce à leur capacité de traiter de grandes quantités d'informations et à leur stabilité face au bruit.

Les réseaux neuronaux sont un ensemble de petit processeurs, appelés neurones, travaillant en pseudo-parallèle. Ces neurones reçoivent des données et les renvoient, après

une transformation [19] appelée fonction d'activation, à d'autres neurones ou à une source externe. Les canaux de communication entre eux contiennent chacune une certaine valeur représentant la capacité de transfert d'informations. Ces valeurs sont modifiées lors de la phase d'apprentissage.

L'ensemble de neurones est partitionné par couche : une d'entrée, des cachées et enfin une de sortie.

Il existe plusieurs manières de différencier les réseaux neuronaux. Ils peuvent l'être par leur côté *feed-forward* ou *feed-back*, le deuxième permettant de réaliser un comportement plus dynamique et complexe que le premier en prenant en compte l'écoulement du temps [20] et en permettant d'avoir des boucles dans le réseau. Le réseau *feed-forward*, appelé aussi *multi-layer perceptron*, semble être plus efficace pour des problèmes de classification et de reconnaissance de forme.

L'algorithme d'apprentissage permet aussi de différencier les réseaux neuronaux. Il y a l'apprentissage par descente du gradient et par l'algorithme de *Widrow-Hoff* pour les réseaux *feed-forward*. Pour les réseaux *feed-back*, il y a la rétropropagation récurrente, la rétropropagation dans le temps (BPTT) et l'apprentissage temporel récurrent en temps réel (RTRL). Il existe d'autres types d'apprentissage selon que le réseau est supervisé ou non. Pour le cas non supervisé, nous pouvons utiliser le *Hebbian Learning Rule* ou le *Competitive Learning Rule*. Le *Hebbian Learning Rule* augmente les valeurs des connexions entre deux neurones qui s'activent simultanément [21] et les diminue dans le cas inverse. Le *Competitive Learning Rule* définit des neurones spécialisés pour certains cas d'input. Pour le cas supervisé, nous pouvons utiliser des algorithmes tels que la rétropropagation ou le *Boltzman learning rule*.

Le choix du nombre de neurones ainsi que du nombre de couches est empirique et est dépendant de plusieurs facteurs tels que le volume de données à traiter, la puissance du processeur, la précision voulue, le temps d'apprentissage maximum,... Nous renvoyons les lecteurs vers [7, 42] pour plus d'informations.

4.9.2 Machine à vecteur de support

Les SVM [22] sont des généralisations de classificateurs linéaires. Le but est de séparer un espace grâce à un hyperplan. Dans le cas où un hyperplan séparateur n'existe pas, nous passons l'ensemble à une dimension supérieure.

Il repose sur deux notions de bases. La première notion est celle de marge maximale. La marge est la distance entre la frontière de deux groupes d'observation appartenant à

19. Cette transformation peut être la fonction seuil, la sigmoïde, la tangente hyperbolique,...

20. Les réseaux *feed-back*, contrairement aux réseaux *feed-forward*, sont donc dépendants de l'ordre des informations que nous leur donnons lors des différentes phases (phase d'apprentissage, d'exécution,...).

21. C'est-à-dire qui reçoivent des données en input simultanément

22. Machine à vecteur de support.

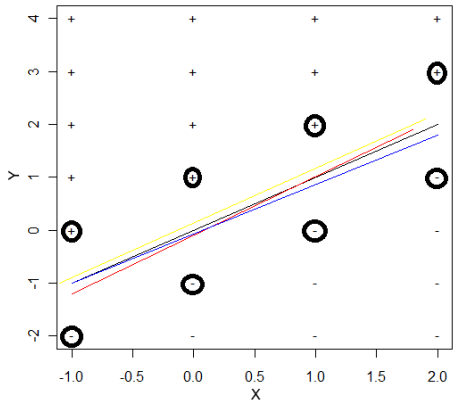


Figure 4.5 : Graphique affichant deux types d'observation, celles appartenant à une certaine classe et qui sont représentées par une croix, et les autres représentées par un moins. Les observations entourées représentent les vecteurs supports. La marge qui maximise la distance, est représentée par une droite en noir, les autres droites ne la maximisent pas.

des classes différentes et les plus proches observations de cette frontière. Ces observations sont appelées vecteurs supports. Le but est de trouver la frontière qui maximise la marge. La figure 4.5 illustre cela en affichant deux types d'observation, celles appartenant à une certaine classe et qui sont représentées par une croix, et les autres représentées par un moins. Les observations entourées représentent les vecteurs supports. La marge qui maximise la distance, est représentée par une droite en noir, les autres droites ne la maximisent pas.

En d'autres termes, nous nous trouvons devant un problème d'optimisation quadratique qui a été exploré par [54]. La raison de cette maximisation vient du fait que, bien qu'il existe une infinité d'hyperplans qui permettent de réaliser une séparation, celui qui maximise la marge est celui qui minimise l'erreur de généralisation.

Cette notion est exploitable dans le cas où les données sont linéairement séparables. Toutefois, dans le cas contraire, il est nécessaire de regarder une autre notion clé des SVM, celle du passage à une dimension supérieure des données. Avec cet agrandissement, nous espérons arriver à une dimension où les données sont linéairement séparables. Pour faciliter certains calculs de la recherche du meilleur hyperplan, la transformation non linéaire se réalise grâce à une fonction appelée noyau.

Nous renvoyons les lecteurs vers [13] pour plus d'informations.

4.9.3 Arbre de décision

Pour illustrer un arbre de décision, appelé aussi arbre de classification, nous allons prendre l'exemple du ballon. Supposons vouloir connaitre la couleur d'un ballon. Dans notre cas, la couleur est déterminable grâce au diamètre du ballon. En effet, nous consta-

tons que les ballons jaunes ont 1 cm, les rouges 3 cm et les noirs 10 cm de diamètre. Ainsi, nous allons mettre en place un système composé d'une planche et de 3 trous. Le premier trou fait tomber toutes les balles qui ont une taille plus petite que 2. Le deuxième fait tomber les balles ayant au maximum 4 cm de diamètre. Le dernier trou laisse passer toute les autres balles. Ainsi, suivant où la balle tombe, nous pouvons déterminer sa couleur. Nous venons de construire un arbre de décision. Sa simplicité permet de voir visuellement comment l'arbre fait pour classifier des données au vue des règles qu'il construit. Ainsi, ça facilite l'interprétation de la classification contrairement à d'autres techniques qui sont vue comme une boite noire.

En effet, nous pouvons le comprendre comme un ensemble de règles du type « si nous avons telle caractéristique, nous sommes catégorisés comme tel groupe ». Pour condenser ces règles qui permettent de catégoriser chaque observation, nous réalisons une hiérarchie de questionnement du type suivant :

Si nous avons telle caractéristique A
  Si nous avons telle caractéristique B
    Alors nous sommes catégorisés de type *ab*
  Sinon
    Si nous avons telle caractéristique C
    ...

Nous voyons apparaitre un arbre hiérarchique, autrement dit un arbre de décision, où chaque feuille est liée à une catégorie. La construction se fait par itération. Nous commençons par prendre l'ensemble des observations de l'ensemble d'apprentissage, et le numéro de classe auquel chaque observation est liée. Au début, nous nous trouvons sur la racine de l'arbre, avec toutes ces observations pour essayer de finir avec un ensemble de feuilles le plus homogène [23] possible qui contiennent chacune une partition, mutuellement exclusive des autres partitions et de l'ensemble des observations.

A chaque nœud, nous coupons l'ensemble en deux ou plusieurs parties, et ce en fonction des variables cibles et une ou des variables descriptives. Ce découpage crée plusieurs fils, chacun prenant un certain sous-ensemble. Ce partitionnement se fait en considérant chaque variable. Le meilleur pris est celui qui maximise l'homogénéité des fils. Notons qu'au lieu de considérer une seule variable, il est possible de le faire sur un ensemble de variables. Ainsi, nous choisissons une certaine variable et nous réalisons le partitionnement en fonction de celle-ci. Puis, nous en prenons une autre et nous réalisons le découpage en fonction de l'autre. Ainsi de suite, jusqu'à trouver la meilleure variable, celle qui amène à la plus grande homogénéité des classes des sous-ensembles. Ce découpage se poursuit pour chaque nœud, jusqu'à avoir une certaine homogénéité des observations se trouvant dans

23. Notons que nous allons parler d'homogénéité pour faciliter la compréhension. Toutefois, il aurait été plus exacte de parler d'erreur de prédiction.

l'ensemble traité ou qu'une certaine règle soit violée telle que la grandeur maximale de l'arbre ou le gain d'homogénéité d'une nouvelle découpe. Lorsque cette dernière survient, le nœud devient une feuille de l'arbre.

Nous arrêtons le processus lorsque nous avons que des feuilles, et des nœuds intermédiaires. En réalisant un tel arbre, nous nous rendons compte qu'au final, nous aurons un grand arbre avec des feuilles très homogènes. Ce découpage est donc très sensible à l'ensemble d'apprentissage. En d'autres termes, nous aurons sur-appris. C'est pourquoi, après cette première étape, nous réalisons une phase d'élagage où nous allons supprimer des feuilles les moins intéressantes. Cela signifie que nous allons éliminer les feuilles qui engendrent le plus d'erreurs avec des données non vues lors de la première étape. Pour évaluer cela, nous pouvons, par exemple, découper l'ensemble d'apprentissage en deux : celui qui permet de construire l'arbre et celui qui permet de trouver les moins bonnes feuilles. Le but ultime est donc d'avoir un petit arbre où les feuilles sont les plus homogènes possibles. Pour atteindre ce but, plusieurs méthodes existent, qui différent par la manière de définir la meilleure partition, la règle qui décide quand un nœud devient une feuille et enfin quel classe nous assignons à une feuille contenant des éléments de classe différents. Toutefois, ces techniques ne seront pas détailler dans ce document.

Nous renvoyons les lecteurs vers [4] pour plus d'informations.

4.9.4 Forêt aléatoire

La forêt aléatoire (Random Forest ou RF) est un ensemble, un regroupement, une forêt de f arbres de décision $\{F_1, F_2, ..., F_f\}$. Il est majoritairement utilisé pour la classification de données. Celui-ci prend en entrée des données, les renvoie à chaque arbre F_i, et renvoie la classe qui est majoritairement sortie par l'ensemble des arbres. Chaque arbre est construit suivant l'algorithme 4.2.

Nous obtenons donc un ensemble d'arbres qui ont sur-appris. Toutefois, en les regroupant, nous obtenons un bon estimateur [24]. Notons que nous perdons le côté interprétable des règles que nous avions avec un arbre de décision.

4.10 Conclusion

Nous venons de voir l'ensemble des étapes pour construire un modèle permettant de prédire des variables cibles, en ne se focalisant que sur des variables observées. Par conséquent, désormais que nous avons le bagage suffisant dans le domaine de la cryptanalyse et de l'apprentissage automatique, nous allons le mettre en pratique au chapitre suivant.

24. L'union fait la force.

Algorithme 4.2 : Construction d'un arbre pour une forêt aléatoire.

1. Soit D_n, l'ensemble des observations de taille n, et N le nombre de variables d'observation.
2. Nous tirons n observations, avec remplacement, de manière aléatoire pour former l'ensemble d'entrainement. Le reste, appelé aussi « *out of bag sample (OOB)* », sera utilisé pour l'ensemble de validation ou pour évaluer l'ensemble des variables.
3. Soit $v' \leq N$, un entier disant combien de variables seront prises en considération à chaque nœud de l'arbre.
4. Pour chaque nœud, nous allons prendre aléatoirement v' variables qui permettraient de faire une prise de décision à ce niveau. Ce choix aléatoire permet d'avoir une forêt d'arbres différents, indépendante.
5. Choisir le meilleur partitionnement de l'ensemble d'apprentissage sur base des v' variables prises à l'étape précédente pour en déduire les règles du nœud.
6. Nous agrandissons l'arbre jusqu'à la fin, sans aucun élagage.

Chapitre 5

Mise en œuvre de l'attaque

Pour passer d'un cadre théorique à un cadre pratique, des recherches ont été réalisées où nous avons combiné l'apprentissage automatique et la cryptanalyse. Elles ont étés effectuées durant un stage réalisé du 1^{er} octobre 2009 au 30 avril 2010 avec le mois de janvier comme congé. Le rapport de stage se trouve en Annexe 3.

Le but était donc d'attaquer en pratique un matériel de chiffrement avec l'aide de l'apprentissage automatique. En d'autres termes, nous avons cherché la clé de chiffrement du matériel. Ce chapitre présente les résultats obtenus.

Il se divise en trois grandes parties, l'une traitant sur le contexte du stage, une autre sur l'attaque à proprement parler et enfin une qui se focalise sur une découverte qui a eu lieu durant le stage et qui ne porte pas directement sur l'attaque du matériel mais qui peut, à long terme, être utile.

Notons que ce chapitre, ainsi que le suivant présentent notre plus grande contribution de ce mémoire aux domaines de la cryptanalyse et de l'apprentissage automatique.

5.1 Contexte

Pour pouvoir évaluer la proposition de combiner deux domaines différents, des recherches ont été effectuées dans le cadre d'un stage chez Atos Worldline, une filiale d'Atos Origin. Cette dernière est une entreprise spécialisée entre autres dans le traitement des transactions électroniques, dans les services de paiement et le traitement de cartes. Leur domaine d'activité est par conséquent un excellent choix pour mettre en œuvre la stratégie d'attaque proposée dans ce mémoire. En effet, ils mettent tout en œuvre pour proposer un produit de chiffrement qui devrait être robuste à une attaque, qu'elle soit physique ou algorithmique. Outre le stage, une présentation du travail effectué a eu lieu en mars 2010 chez Atos Worldline. Les slides de cette présentation sont disponibles en Annexe 2.

Le travail effectué est une suite à celui réalisé par Stéphane Fernandes Medeiros[21] sur le même matériel de chiffrement mais en ne se focalisant que sur les DPA et les SPA.

L'appareil chiffrait, avec l'algorithme 3DES [1], un même message en utilisant des clés différentes où seuls 8 bits parmi les 64×3 de la clé variaient. Ainsi, mis à part 8 bits, le reste de la clé ne changeait pas [2].

Pour une question de facilité et de rapidité, l'ensemble des calculs a été réalisé dans le centre de calcul de l'ULB en utilisant le langage informatique R.

Les méthodes d'apprentissage automatique, plus précisément celles de classifications supervisées, ont été utilisées. Ces méthodes demandent la mise en place de deux phases :

1. Phase d'entrainement : nous créons un modèle, qui va nous permettre de modéliser le matériel de chiffrement. Notons que lors de cette phase, nous supposons avoir un contrôle total sur le FPGA [3], sur le matériel, qui chiffre les données, comme par exemple la possibilité d'envoi d'une certaine clé avec une certaine donnée à chiffrer.
2. Phase d'attaque : nous utilisons le modèle de l'étape 1 pour retrouver n'importe quelle clé sur un matériel similaire à celui acquis précédemment. Nous nommons cette phase le problème de classification de la trace. Puisque nous construisons un modèle où le message à chiffrer ne varie pas, nous supposons donc pouvoir déterminer quel message nous voulons chiffrer lors de cette phase.

En ce qui concerne les données récoltées d'un matériel de chiffrement représentant des traces, elles ont été prises grâce à un oscilloscope [4] et trois probes. L'une était liée au matériel de chiffrement pour savoir quand ce dernier chiffrait réellement les données. Deux autres ont été utilisées pour obtenir la consommation d'énergie par unité de temps. De plus, un ordinateur était relié à l'oscilloscope et au matériel de chiffrement pour envoyer l'ordre de chiffrement et récolté les traces.

5.2 Prétraitement

L'ensemble des traces récoltées ont dû passer par un prétraitement pour réduire autant que possible le bruit. Dans l'ensemble des cas, nous avions à faire à 400 traces liées à une clé. La réduction de bruit s'est donc faite en calculant la moyenne de ces 400 traces comme illustrée à la Figure 5.1.

La raison de ce choix vient de la supposition que chaque trace $T_{j,1...N}$, où N est le nombre d'instants de la trace, est la somme de la consommation d'énergie $i_{j,1...N}$ du matériel et d'un bruit [5] $B_{j,1...N}$ de moyenne nulle : $T_{j,t} = i_{j,t} + B_{j,t}$. Le bruit représente l'ensemble des variations inexpliquées mais réelles de la trace, qui n'ont pas été prises

1. L'algorithme 3DES est encore fort utilisé dans le milieu bancaire pour des raisons de conservatisme.
2. Le fait d'essayer toutes les clés possibles n'a pas été réalisable en pratique.
3. Cette considération n'est pas limitative puisque le matériel utilisé est en vente libre.
4. L'oscilloscope est un Agilent infiniium 1GHz 4GSa/s.
5. Le bruit est appelé aussi bruit blanc. Ce dernier suit théoriquement une loi normale (cf : Théorème central-limite).

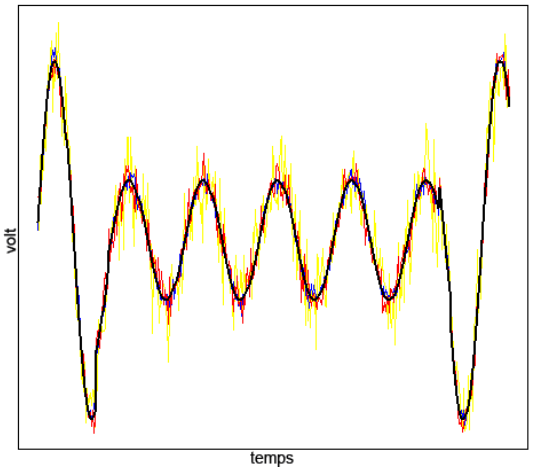


Figure 5.1 : Illustration de différentes traces de la consommation d'énergie par unité de temps de la machine de chiffrement utilisant une même clé sur un même message. La ligne noire, étant le traitement statistique pour réduire le bruit, représente leur moyenne.

en considération dans le modèle de prédiction. Ainsi, $\frac{1}{n}\sum_{j=1}^{n} T_{j,t}$ donne la consommation d'énergie au temps t sans le bruit où n est le nombre de traces collectées [6].

Au total, nous étions donc en face de 128 traces nettes par byte attaqué lorsque nous ne prenons pas en compte le bit de parité.

Les traces récoltées par [21] comportaient 20000 points, symbolisables en dimension 20000. Elles représentent le $8^{ème}$ byte de la première clé. Les autres en comportaient 6000. La raison de cette différence vient du fait que [21] a utilisé un oscilloscope [7] permettant d'atteindre jusqu'à 20000 points, alors que nous avions à disposition un matériel arrivant jusqu'à 6000 instants.

5.3 Visualisation graphique

Après le prétraitement, l'étape suivante à réaliser lorsque nous attaquons un matériel est d'avoir une certaine visualisation des traces dans une dimension visible à l'œil humain, ce qui veut dire une dimension ne dépassant pas la $3^{ème}$. Cette réduction signifie que chaque trace sera définie grâce à trois coordonnées. Cette visualisation permet d'avoir une première idée du regroupement des traces utilisant la même valeur d'un certain bit.

Une technique de réduction de dimension est le PCA présentée dans le chapitre 4, section 3. Notons que la diminution de la dimension du problème engendre un certain

6. Le nombre de traces que nous pouvons récolter est une limitation du problème. En effet, si nous supposons que le matériel ne tolère qu'un certain nombre d'opérations avec la même clé, la diminution du bruit sera d'autant plus complexe que ce nombre est faible.

7. L'oscilloscope de [21] était un Agile infiniium DSO80204B 2Ghz 40GSa/s.

taux de perte dû à cette réduction.

Pour une raison de taille, nous n'allons réaliser cette visualisation que pour le $8^{ème}$ byte de la première clé. L'ensemble des sous-figures se trouvent à la figure 5.2. Chaque sous figure montre, grâce à des couleurs différentes, la valeur d'un bit. Grâce à ces visualisations, nous constatons une complication croissante de la distinction des lieux où le bit vaut une valeur différente. En d'autres termes, plus nous nous approchons du bit de parité, plus il est complexe d'en prédire la valeur. Nous pouvons donc prédire une plus grande précision de la prévision des premiers bits, pour le $8^{ème}$ byte de la première clé.

Nous pouvons aller encore plus loin dans la visualisation en regardant quelles sont les clés qui sont proches entre elles. Ce nouveau regard permet d'avoir une certaine évaluation des clés qui pourraient être confondues, amenant à une plus grande complexité de retrouver les clés qui sont proches les unes des autres.

Pour une raison de clarté, la figure 5.3 représente cette réduction à une dimension 2 et non 3. Nous y voyons la projection de quelques traces sur deux dimensions. Sur le graphique, chaque projection d'une trace est représentée par un numéro. Ce dernier est la clé associé à la trace.

Nous voyons clairement que certaines valeurs du byte forment des groupes. En regardant plus précisément sur les valeurs, nous pouvons remarquer que les traces, qui ont une ordonnée plus petite que 0, sont liées en majorité à un byte ayant une valeur nulle dans le 1^{er} bit. Il va de soi qu'au vu des figures précédentes, les clés ayant des valeurs fortement différentes dans les derniers bits et faiblement dans les premiers sont complexes à différencier. En effet, prenons l'exemple de clés :

1. 0 0 0 0 1 0 1 0
2. 1 0 0 0 1 1 1 0

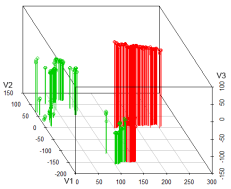
Ces deux clés sont facilement différenciables en regardant le premier bit, qui est par ailleurs facilement prévisible au vu des graphiques précédents. Toutefois, les clés suivantes sont complexes à différencier :

1. 0 0 0 0 0 0 0 1
2. 0 0 0 0 0 0 0 0

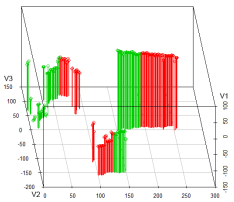
En effet, le dernier bit semble être complexe à prédire.

5.4 Description de l'approche

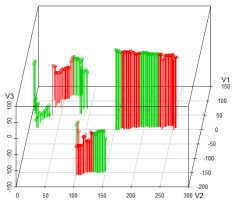
Désormais que nous avons réduit nos doutes sur le lien effectif entre la consommation d'énergie et la clé utilisée, il est nécessaire de réaliser la première phase de l'apprentissage automatique. Pour choisir le meilleur modèle, nous avons utilisé le jeu de données collecté par [21] dans le but de l'analyser.



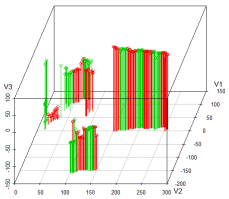
(a) La différence de couleur des points indique la valeur différente du 1^{er} bit du byte.



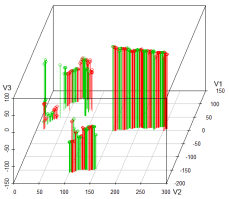
(b) La différence de couleur des points indique la valeur différente du $2^{ème}$ bit du byte.



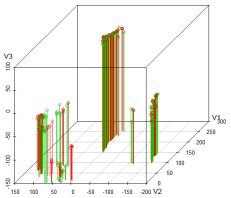
(c) La différence de couleur des points indique la valeur différente du $3^{ème}$ bit du byte.



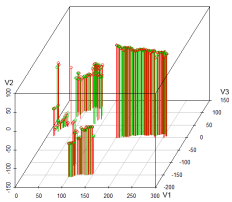
(d) La différence de couleur des points indique la valeur différente du $4^{ème}$ bit du byte.



(e) La différence de couleur des points indique la valeur différente du $5^{ème}$ bit du byte.



(f) La différence de couleur des points indique la valeur différente du $6^{ème}$ bit du byte.



(g) La différence de couleur des points indique la valeur différente du $7^{ème}$ bit du byte.

Figure 5.2 : Cette figure montre les traces de la première clé, au $8^{ème}$ byte. Les points rouges représentent un bit valant 1, les autres symbolisent un bit valant 0.

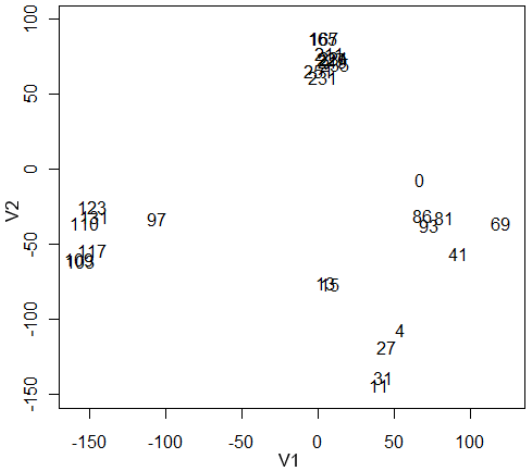


Figure 5.3 : Cette figure montre les valeurs en décimal du $8^{ème}$ byte de la $1^{ère}$ clé. Nous y voyons la projection de quelques traces sur deux dimensions. Sur le graphique, chaque projection d'une trace est représentée par un numéro. Ce dernier est la clé associé à la trace.

Sachant que l'information importante pour la cryptanalyse se trouvait entre les points 8000 et 17399 [8], seul un sous-ensemble de chaque trace fut utilisé. Ces deux points ont été obtenus grâce à des flags montrant quand le FPGA commence et termine le chiffrement.

Néanmoins, l'ajout de flags ne réduit pas la possibilité de mise en pratique de l'attaque. En effet, ceci permet uniquement de gagner du temps. Nous verrons par la suite qu'il existe des techniques permettant de trouver quelle partie d'une trace corrèle le mieux avec la valeur de la clé. Ainsi, il suffit de prendre les points qui corrèlent le plus pour simuler l'implémentation de ces flags.

La première idée fut de construire une structure modulaire, c'est-à-dire un modèle supervisé pour chacun des 8 bits à attaquer. Ainsi, nous avons créé 8 modèles, chacun d'entre eux prédisant un parmi les 8 bits à attaquer (Figure 5.4). Nous pouvons le définir comme un type d'attaque « *diviser pour régner* » où nous nous attaquons à une partie de la clé à la fois.

A cause d'un petit nombre de données disponibles, l'ensemble des modèles présenté dans ce chapitre a été validé par la technique du *leave-one-out*. Cette méthode fut appliquée pour toutes les traces. Ainsi, lorsque nous ne prenons pas en compte le bit de parité, nous avons 128 étapes pour chaque modèle. Dans le cas contraire, nous en avons 256.

L'ensemble de ces validations permet de connaître le pourcentage de bonnes prédictions réalisées par un modèle construit sur un ensemble de traces et ainsi d'avoir une idée de la capacité du modèle de généraliser le problème.

8. [21] a supposé que l'information utile à la cryptanalyse ne se trouvait que lorsque le FPGA chiffrait. Nous verrons par la suite que ceci n'est pas totalement exact.

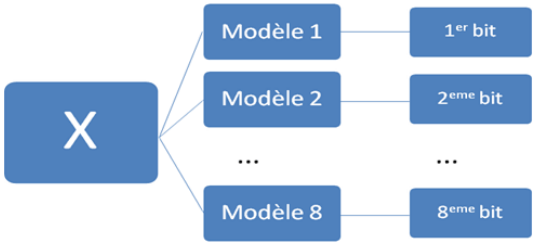


Figure 5.4 : L'ensemble des traces (X) est donné à chaque modèle pour qu'il prédise un bit de la clé.

5.5 Choix du modèle

Le choix du modèle se base sur la maximisation d'une quantité. Pour cela, nous allons calculer les vrais positifs [9], les vrais négatifs [10], les faux négatifs [11] et les faux positifs [12]. Le but est de chercher le modèle qui maximise la somme des vrais positifs et des vrais négatifs.

SOM de taille 8×5

Le premier SOM [13] réalisé a une taille de 8×5 [14]. Voici, pour chaque bit, les vrais positifs (TP), les vrais négatifs (TN), les faux négatifs (FN) et les faux positifs (FP) :

SOM(8×5)	1	2	3	4	5	6	7	8
TP	125	115	115	101	69	58	55	49
TN	121	116	109	89	68	65	59	44
FN	7	12	19	39	60	63	69	84
FP	3	13	13	27	59	70	73	79
% de bonnes réponses	96.09	90.23	87.50	74.22	53.52	48.05	44.53	36.33

La première remarque que nous pouvons faire sur la capacité de généralisation du modèle est que nous observons des résultats semblables à ceux de [21] : les premiers bits sont mieux prédits que les derniers. Ceci pourrait venir du fait que certains bits de la clé sont plus utilisés que d'autres lors de l'algorithme cryptographique. En effet, l'algorithme 3DES génère 16×3 sous clés de 48×3 bits à partir de la clé initiale pour en donner une différente à chaque étape. Chaque sous clé contient certains bits de la clé initiale. Ainsi, en

9. Le modèle a prédit correctement zéro.
10. Le modèle a prédit correctement un.
11. Le modèle a prédit zéro alors qu'il aurait dû prévoir un.
12. Le modèle a prédit un alors qu'il aurait dû prévoir zéro.
13. Le SOM supervisé fut construit grâce à la méthode bdk() de la librairie kohonen[P4].
14. Les résultats du SOM 8×5 sont reproductibles grâce au fichier code.R se trouvant dans le DVD joint à ce mémoire.

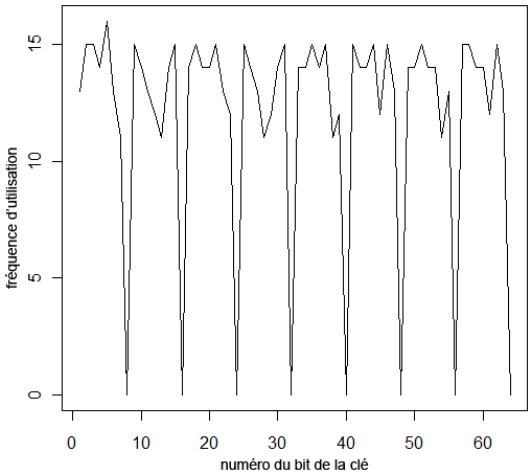


Figure 5.5 : Ce graphique montre en abscisse le numéro du bit de la clé et en ordonnée la fréquence d'utilisation du bit. Notons que ce graphique ne montre que la fréquence d'utilisation de chaque bit de la première clé de l'ensemble des trois clés.

calculant la fréquence d'utilisation de chaque bit de la clé initiale, nous nous apercevons que certains bits sont plus employés que d'autres (Figure 5.5).

La deuxième remarque que nous pouvons faire est que la capacité de prédiction du byte est meilleure que celle du modèle de [21].

Etonnamment, nous constatons une prédiction en dessous des 50 % pour les $6^{ème}$, $7^{ème}$ et $8^{ème}$ bits, contrairement aux résultats de [21] où les derniers bits sont prédits avec +/-50% de chance.

Une idée triviale serait de prendre l'inverse de ce que le modèle nous prédit [15] pour augmenter légèrement sa capacité de prévision à 51.95, 55.47 et 63.67 % pour les trois derniers bits. Néanmoins, du point de vue de l'algorithme d'apprentissage automatique, la raison de ce phénomène semblerait venir de la complexité d'apprentissage de ces bits, autrement dit ces bits seraient trop complexes à déterminer avec les données que nous fournissons au modèle. Ainsi, l'idée triviale n'est pas applicable.

Pour confirmer cette théorie, des recherches ont eu lieu pour comprendre ce phénomène. Tout d'abord, le dernier bit, celui de parité, n'est pas pris en compte dans le chiffrement d'où sa difficulté d'apprentissage. En effet, puisque ce bit n'a aucun lien avec le chiffrement, il n'y a aucune raison pour qu'il influence la quantité d'énergie consommée par la machine de chiffrement. Néanmoins, selon le protocole d'Atos Worldline, ce bit est généré aléatoirement et n'est donc pas utilisé comme bit de parité. Ainsi, il devrait être prédit à environs 50 % ce qui n'est pas le cas en pratique. La raison de ce phénomène semblerait venir du peu de données à disposition. En effet, il n'est pas logique qu'un al-

15. En d'autres mots, lorsque le modèle nous prédit 1, nous choisissons 0, et lorsque celui-ci prévoit 0, nous prenons 1.

gorithme d'apprentissage à qui nous demandons de faire le lien entre une trace et un bit, comprenne le contraire et qui plus est, arrive à prédire une donnée aléatoire. Dans notre cas, il semblerait donc logique de rejeter tous les modèles prédisant des données avec moins de 50 % de bonnes réponses. Cette règle amène un rejet d'une autre idée, celle triviale où nous prenions l'inverse de ce que le modèle prédisait pour améliorer le pourcentage de bonnes réponses. En effet, dans le cas où nous augmenterions le nombre de données, nous devrions ramener ce pourcentage à 50 %. Pour confirmer cette règle, nous avons regardé le pourcentage de bonnes réponses parmi l'ensemble des données d'apprentissage. Ainsi, nous obtenons 59.53 % et 57.88 % de bonnes réponses pour les données d'apprentissage pour les $7^{ème}$ et $8^{ème}$ bits. En voyant leur taux d'erreurs, nous pouvons comprendre la raison du taux d'échecs pour les données de validation. En effet, avec cette grandeur aux alentours de 50 %, le taux de prédiction ne peut pas être bon.

Ainsi, nous pouvons supposer que le problème pour les bits 7 et 8 est trop complexe en n'utilisant que les données mises à disposition.

Finalement, avec ce modèle, nous pouvons dire que nous avons 7.53 % [16] de chances d'avoir l'ensemble du byte alors qu'il n'est que de 0.78 % dans le cas où nous tirons au hasard un byte.

Pour avoir une meilleure précision, la deuxième idée a été d'augmenter la complexité du modèle. Ainsi, trois autres modèles ont été construits : un ayant une taille de 9×5, un autre de 8×6 et un dernier de 9×6. Les résultats sont affichés ci-dessous.

SOM de taille 9×5

SOM(9×5)	1	2	3	4	5	6	7
TP	126	116	113	95	79	66	60
TN	123	120	102	83	69	65	48
FN	5	8	26	45	59	63	80
FP	2	12	15	33	49	62	68
% de bonnes réponses	97.27	92.19	83.98	69.53	57.81	51.17	42.19

En conclusion avec ce modèle [17], nous pouvons dire que nous avons 7.74 % de chances d'avoir l'ensemble du byte.

16. Ce pourcentage ainsi que tous les pourcentages qui sont dans ce document, sauf ceux se trouvant dans un tableau, ne prennent pas en compte le bit de parité puisque celui-ci n'est pas pris en compte par le matériel. De plus, dans le cas où le pourcentage de bonnes réponses est inférieur à 50 %, nous mettons par défaut 50 % puisque c'est le pourcentage minimum pour une prévision.

17. Les résultats du SOM 9×5 sont reproductibles grâce au fichier code2.R se trouvant dans le DVD joint à ce mémoire.

SOM de taille 8×6

SOM(8×6)	1	2	3	4	5	6	7
TP	127	112	106	94	76	58	52
TN	120	117	108	94	71	59	49
FN	8	11	20	34	57	69	79
FP	1	16	22	34	52	70	76
% de bonnes réponses	96.48	89.45	83.59	73.44	57.42	45.70	39.45

En conclusion avec ce modèle [18], nous pouvons dire que nous avons 7.61 % de chances d'avoir l'ensemble du byte.

SOM de taille 9×6

SOM(9×6)	1	2	3	4	5	6	7
TP	125	118	110	104	76	70	56
TN	120	120	110	97	73	62	53
FN	8	8	18	31	55	66	75
FP	3	10	18	24	52	58	72
% de bonnes réponses	95.70	92.97	85.94	78.52	58.20	51.56	42.58

En conclusion avec ce modèle [19], nous pouvons dire que nous avons 9.01 % de chances d'avoir l'ensemble du byte.
Ces données sont résumées sur la figure 5.6 où l'abscisse est le numéro du bit que nous attaquons et l'ordonnée représente le pourcentage de bonnes réponses.

La première constatation générale est que certains bits, tels que le $5^{ème}$, peuvent être mieux prédits en augmentant légèrement la complexité du modèle alors que d'autres, tels que le $3^{ème}$, le sont en la diminuant légèrement.

Parallèlement à l'utilisation du modèle SOM, deux autres ont été utilisés [20] : SVM, avec un noyau linéaire, et le RF [21]. Ces derniers sont décrits dans le chapitre 4, section 9, sous-section 2 et 4. La dimension des données étant très grande pour nos modèles, nous avons utilisé des techniques de sélection de variables pour réduire leur taille. Les deux techniques utilisées sont : Nosel et Rank. Le premier renvoie l'ensemble des variables et le second utilise la covariance pour calculer les 20 variables les plus intéressantes [22]. Les résultats sont disponible ci-dessous.

18. Les résultats du SOM 8×6 sont reproductibles grâce au fichier code3.R se trouvant dans le DVD joint à ce mémoire.

19. Les résultats du SOM 9×6 sont reproductibles grâce au fichier code4.R se trouvant dans le DVD joint à ce mémoire.

20. Les algorithmes utilisant ces modèles ont été implémentés par Prof. Bontempi qui a lui-même utilisé des méthodes déjà faites.

21. Random Forest.

22. Des techniques de sélection de variables plus avancées seront utilisées ultérieurement.

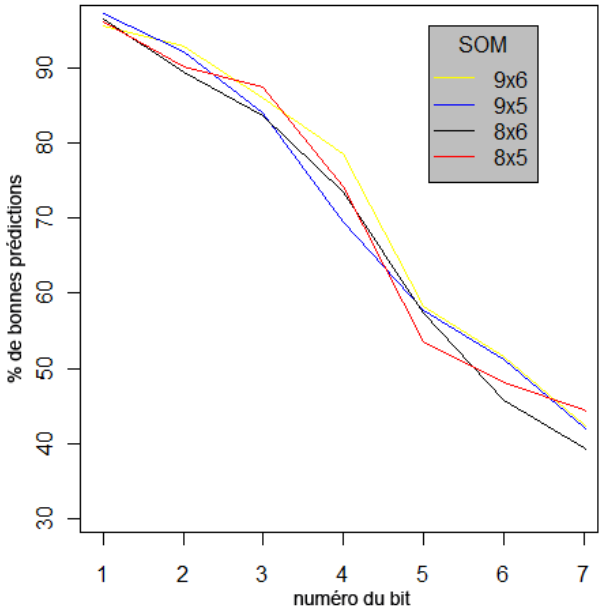


Figure 5.6 : La courbe jaune, bleu, noir et rouge montrent la performance, pour chaque bit, pour le modèle SOM ayant une taille respectivement de 9×6, 9×5, 8×6, et 8×5.

SVM/Rank

SVM/Rank	1	2	3	4	5	6	7
TP	119	102	94	79	40	60	56
TN	123	104	92	81	78	70	33
FN	9	26	34	49	88	68	72
FP	5	24	36	47	50	58	95
% de bonnes réponses	94.53	80.47	72.66	62.5	46.09	50.78	34.77

En conclusion avec ce modèle [23], nous pouvons dire que nous avons 4.39 % de chances d'avoir l'ensemble du byte.

23. Les résultats du SVM/Rank sont reproductibles grâce au fichier class.R se trouvant dans le DVD joint à ce mémoire.

SVM/Nosel

SVM/Nosel	1	2	3	4	5	6	7
TP	125	116	107	93	79	70	56
TN	122	115	105	94	85	67	54
FN	3	12	21	35	49	58	72
FP	6	13	23	34	43	61	74
% de bonnes réponses	96.48	90.23	82.81	73.05	64.06	53.52	42.97

En conclusion avec ce modèle [24], nous pouvons dire que nous avons 9.03 % de chances d'avoir l'ensemble du byte.

RF/Rank

RF/Rank	1	2	3	4	5	6	7
TP	126	106	104	99	77	74	45
TN	124	109	105	99	80	73	44
FN	2	22	24	29	51	54	83
FP	4	19	23	29	48	55	84
% de bonnes réponses	97.66	83.98	81.64	77.34	61.33	57.42	34.77

En conclusion avec ce modèle [25], nous pouvons dire que nous avons 9.12 % de chances d'avoir l'ensemble du byte

RF/Nosel

RF/Nosel	1	2	3	4	5	6	7
TP	124	115	119	114	79	69	51
TN	122	122	109	101	74	73	41
FN	4	13	19	14	49	59	77
FP	6	6	9	27	54	55	87
% de bonnes réponses	96.09	92.58	89.06	83.98	59.77	55.47	35.94

En conclusion avec ce modèle [26], nous pouvons dire que nous avons 11.03 % de chances d'avoir l'ensemble du byte.

L'ensemble de ces résultats est résumé sur la figure 5.7.

24. Les résultats du SVM/Nosel sont reproductibles grâce au fichier class.R se trouvant dans le DVD joint à ce mémoire.

25. Les résultats du RF/Rank sont reproductibles grâce au fichier class.R se trouvant dans le DVD joint à ce mémoire.

26. Les résultats du RF/Nosel sont reproductibles grâce au fichier class.R se trouvant dans le DVD joint à ce mémoire.

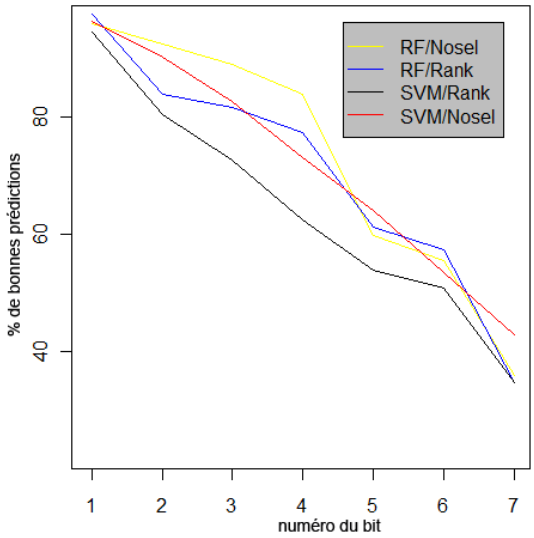


Figure 5.7 : La courbe jaune, bleu, noir et rouge montrent la performance, pour chaque bit, pour le modèle RF/Nosel, RF/ Rank, SVM/Rank et SVM/Nosel.

Nous constatons dans ces derniers cas que le modèle Random Forest combiné avec Nosel donne généralement les meilleures prévisions pour la majorité des bits. Du point de vue pratique, l'ensemble de ces tests ont été long, entre 24h et 48h, par rapport aux méthodes présentées par après. La lenteur d'exécution est majoritairement due à la phase de validation. En effet, dès qu'un modèle était construit, il suffisait de moins d'un $100^{ème}$ de seconde pour calculer une prévision.

5.6 Choix du modèle avec une sélection de variables

Après avoir vu que le modèle Random Forest était de loin le meilleur, il nous est venu comme idée de le combiner avec un modèle de sélection de variables évolué pour réduire la taille des données. En effet, un des plus grands problèmes était la dimension des données qui engendrait un temps d'apprentissage conséquent. Ainsi, bien que la réduction de la taille des données n'amène aucune nouvelle information, il était devenu nécessaire de trouver une technique permettant de réduire la taille du problème en espérant que cette réduction donnerait :

1. Une diminution du temps de calcul total puisque nous réduisons la taille du problème.
2. Une augmentation de la précision de la prédiction puisqu'en réduisant la taille du problème, nous diminuons le bruit causé par un surplus d'informations.

Random Forest/SOM

Le premier modèle utilisé était un SOM non supervisé [27] représentant un premier modèle puissant de sélection de variables. Il a été présenté dans le chapitre 4, section 3. La raison de ce choix fut influencée par le fait que son utilisation était très proche du modèle SOM supervisé qui était déjà abordé en première partie du stage ce qui en fait une implémentation facile et rapide. Pour augmenter l'efficacité de la réduction, la taille du SOM variait d'un bit à l'autre, en fonction de la meilleure taille obtenue dans les résultats précédents. Ainsi, pour le premier bit, la taille du SOM était de 9×5 ; pour le deuxième bit, la taille était de 9×6,... Pour un premier test, le modèle RF [28] comportait 500 arbres au total. De plus, la validation se faisait toujours grâce au *leave-one-out*. Les résultats [29] sont disponibles ci-dessous.

RF/SOM	1	2	3	4	5	6	7
TP	127	109	109	102	81	53	62
TN	120	119	103	93	75	62	55
FN	8	9	25	35	53	66	73
FP	1	19	19	26	47	75	66
% de bonnes réponses	96.48	89.06	82.81	76.17	60.94	44.92	45.70

Une première constatation lors de la mise en pratique de cette technique est une augmentation drastique du temps de prédiction. Ce dernier était aux alentours de 1.5 secondes/prédiction.

En conclusion pour notre premier modèle combinant une sélection de variables et un modèle d'apprentissage, nous pouvons dire que nous avons 8.26 % de chances d'avoir l'ensemble du byte. De plus, nous pouvons constater qu'il y a une légère dégradation des résultats par rapport à RF/Nosel et une augmentation drastique du temps de calcul. Ces deux découvertes sont en contradiction avec les résultats souhaités d'une sélection de variables ce qui nous pousse à essayer un autre modèle de filtre.

Random Forest et PCA

La deuxième sélection de variables testée était le PCA [30].

Pour des raisons de clarté, les résultats détaillés du modèle sont situés en annexe A [31].

Ces données sont représentées sur la figure 5.8 où l'abscisse indique le nombre de variables pris en compte et l'ordonné, la probabilité de retrouver le byte en entier.

27. Le SOM non supervisé fut construit grâce à la méthode som() de la librairie kohonen[P4].

28. Le Random Forest a été implémenté grâce à la méthode randomForest() de la librairie randomForest[P2].

29. Les résultats du RF/SOM sont reproductibles grâce au fichier code8.R se trouvant dans le DVD joint à ce mémoire.

30. Le PCA fut construit grâce à la méthode dudi.pca() de la librairie ade4[P1].

31. Toutes les annexes qui sont symbolisées par une lettre se trouvent dans le DVD joint à ce mémoire.

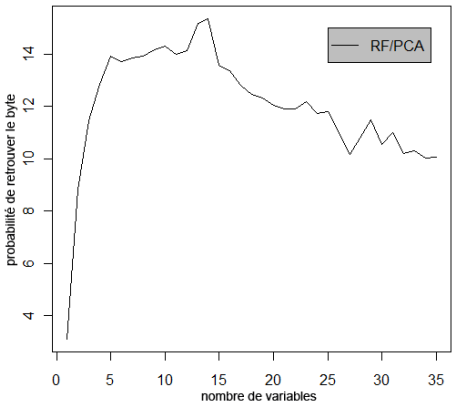


Figure 5.8 : L'abscisse indique le nombre de variables pris en compte et l'ordonné, la probabilité de retrouver le byte en entier.

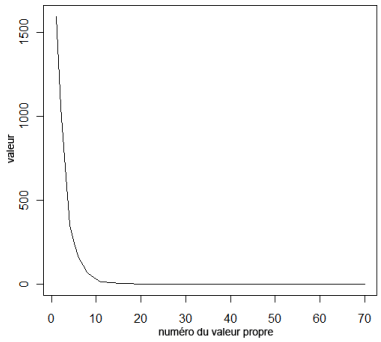
La première remarque est que dès que nous prenons au moins 3 composantes principales, nous obtenons de meilleurs résultats que si nous n'avions pas réalisé de sélection de variables [32]. La raison pourrait s'expliquer par le fait qu'en diminuant la taille du problème, nous réduisons le bruit représenté par un surplus d'informations. Cette diminution permet de réaliser un meilleur apprentissage et donc une meilleure prédiction. D'un autre côté, en élevant le nombre de composantes principales, nous augmentons la taille du problème. C'est une des raisons qui pourrait expliquer la diminution du pourcentage de bonnes réponses après 14 composantes prises.

Ces résultats pourraient aussi se voir en regardant uniquement les valeurs des valeurs propres (Figure 5.9.a) et la quantité de variances initiales gardées (Figure 5.9.b). Celles-ci montrent qu'il suffisait de s'arrêter à une dimension 14 pour avoir 98.08 % de la variance initiale.

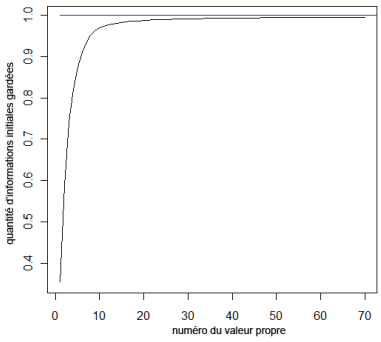
De plus, une autre constatation en pratique est la rapidité d'exécution. En effet, il n'a suffi que d'environ 5 heures pour construire et valider un modèle en entier alors que les précédents modèles en demandaient entre 24 et 48 heures. Ceci pourrait s'expliquer par une technique de sélection rapide de variables combinée avec un temps d'apprentissage et de validation sur un petit ensemble de variables. Ainsi, ça en fait un modèle de choix pour des exécutions rapides.

En conclusion, pour notre deuxième modèle utilisant une sélection de variables et un modèle d'apprentissage, pour le meilleur nombre de composantes principales, à savoir 14, nous pouvons dire que nous avons 15.33 % de chances d'avoir l'ensemble du byte alors qu'il n'est que de 0.78 % dans le cas où nous tirons au hasard un byte. De plus, nous réduisons considérablement le temps de calcul tout en augmentant la précision du modèle. Ce dernier modèle sera donc celui utilisé pour la suite du mémoire.

32. Le modèle RF/Nosel permet de prédire un byte à une probabilité de 11.03 %.



(a) Le graphique représente le numéro de chaque valeur propre en fonction de leur valeur.



(b) L'abscisse représente le numéro de chaque valeur propre et l'ordonnée, la variance initiale gardée. La courbe noire représente le lien entre ces deux données. La ligne bleue représente la limite atteignable par la courbe noire.

.

Figure 5.9 : Ces figures montrent qu'il suffisait de s'arrêter à une dimension 14 pour avoir 98.08% de l'information initiale.

5.7 Attaque des autres bytes

L'étape suivante fut d'obtenir de nouvelles données sur le matériel dans le but de s'attaquer aux autres bytes de la clé. Ainsi, un ordinateur de production envoyait 400 fois des demandes de chiffrement d'une donnée constante en utilisant la même clé. Ceci fut réalisé pour tout un byte de la clé, soit 128 fois. Parallèlement à ces envois de messages, un oscilloscope mesurait la consommation d'énergie du FPGA. Après cela, un prétraitement fut réalisé pour n'avoir que les données nettes à traiter. Les résultats, où nous attaquons chaque byte individuellement, sont disponibles ci-dessous.

Attaque de la première clé – 1^{er} byte

Les résultats figurent sur le graphique 5.10. Les données détaillées se trouvent en Annexe H.

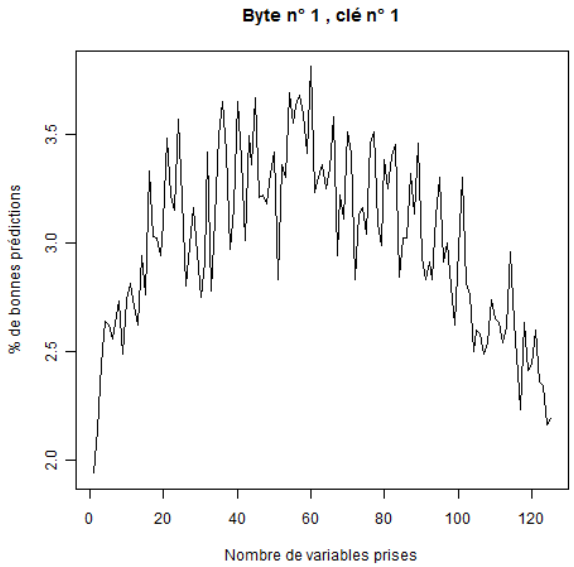


Figure 5.10 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le 1^{er} byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 61 pour avoir 3.81 % de chances d'avoir l'ensemble du byte. En effet, nous voulons maximiser le taux de bonnes prédictions. Ainsi, en prenant 61 dimensions, nous remarquons que nous arrivons à un maximum de 3.81 %.

Attaque de la première clé – $2^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.11. Les données détaillées se trouvent en Annexe K.

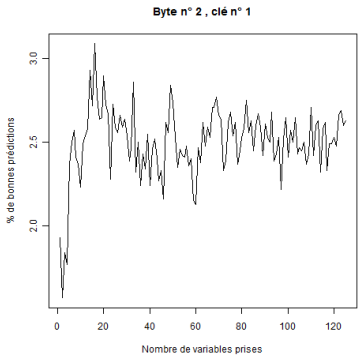


Figure 5.11 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $2^{ème}$ byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 17 pour avoir 3.09% de chances d'avoir l'ensemble du byte.

Attaque de la première clé – $3^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.12. Les données détaillées se trouvent en Annexe L.

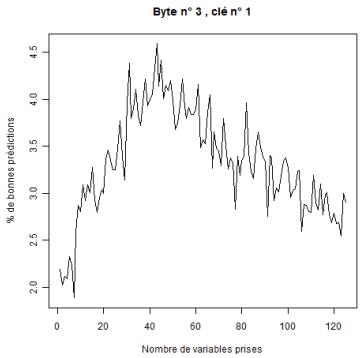


Figure 5.12 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $3^{ème}$ byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 44 pour avoir 4.59% de chances d'avoir l'ensemble du byte.

Attaque de la première clé – $4^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.13. Les données détaillées se trouvent en Annexe M.

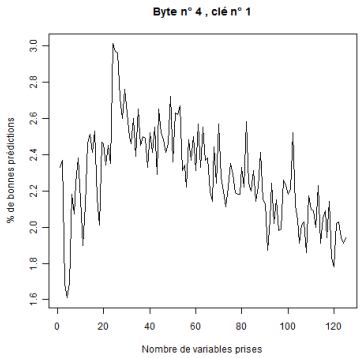


Figure 5.13 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $4^{ème}$ byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 25 pour avoir 3.01% de chances d'avoir l'ensemble du byte.

Attaque de la première clé – $5^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.14. Les données détaillées se trouvent en Annexe N.

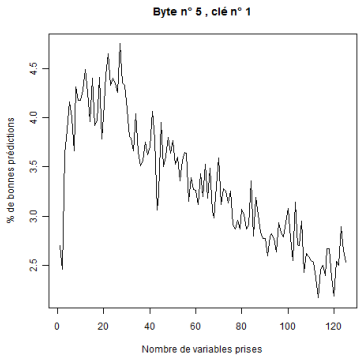


Figure 5.14 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $5^{ème}$ byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 28 pour avoir 4.75% de chances d'avoir l'ensemble du byte.

Attaque de la première clé – $6^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.15. Les données détaillées se trouvent en Annexe O.

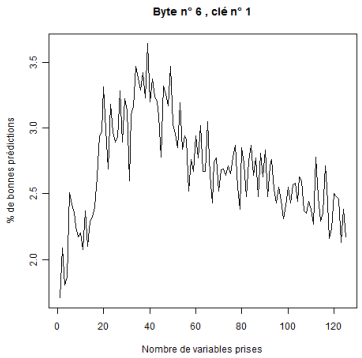


Figure 5.15 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $6^{ème}$ byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 40 pour avoir 3.64% de chances d'avoir l'ensemble du byte.

Attaque de la première clé – $7^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.16. Les données détaillées se trouvent en Annexe P.

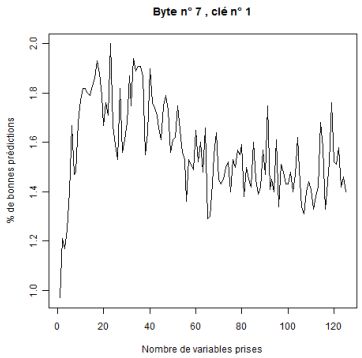


Figure 5.16 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $7^{ème}$ byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 24 pour avoir 2.00% de chances d'avoir l'ensemble du byte.

Attaque de la première clé – $8^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.17. Les données détaillées se trouvent en Annexe Q.

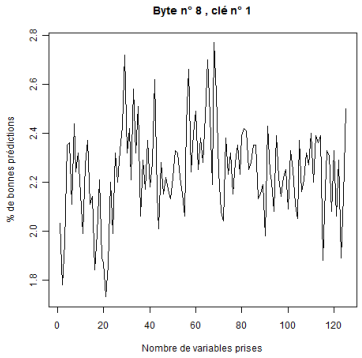


Figure 5.17 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $8^{ème}$ byte de la $1^{ère}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 39 pour avoir 2.17% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – 1^{er} byte

Les résultats figurent sur le graphique 5.18. Les données détaillées se trouvent en Annexe I.

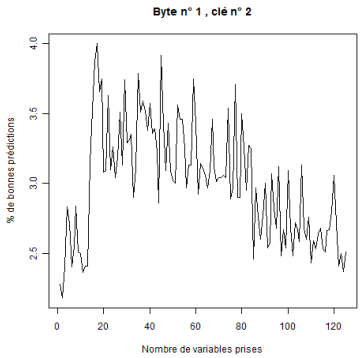


Figure 5.18 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le 1^{er} byte de la $2^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 18 pour avoir 4.00% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – $2^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.19. Les données détaillées se trouvent en Annexe R.

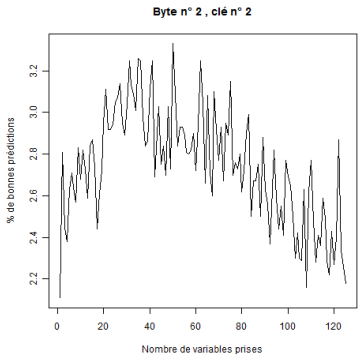


Figure 5.19 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $2^{ème}$ byte de la $2^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 51 pour avoir 3.33% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – $3^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.20. Les données détaillées se trouvent en Annexe S.

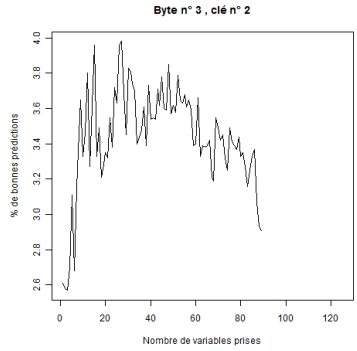


Figure 5.20 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $3^{ème}$ byte de la $2^{ème}$ clé. Nous nous sommes arrêtés à une dimension 90 car au-delà les autres variables n'étaient qu'une combinaison linéaire d'au moins une des 90 variables.

Elle montre qu'il suffisait de s'arrêter à une dimension 28 pour avoir 3.98% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – $4^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.21. Les données détaillées se trouvent en Annexe T.

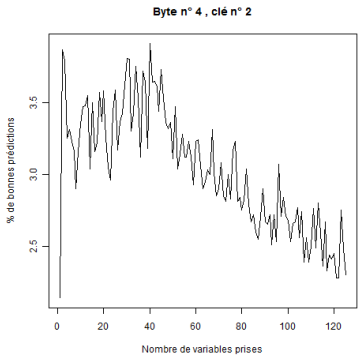


Figure 5.21 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $4^{ème}$ byte de la $2^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 41 pour avoir 3.91% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – $5^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.22. Les données détaillées se trouvent en Annexe G.

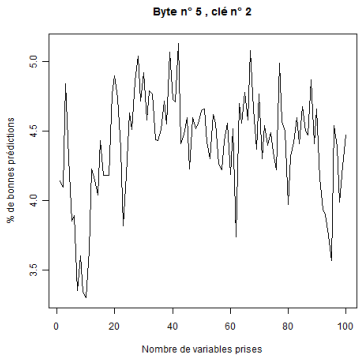


Figure 5.22 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $5^{ème}$ byte de la $2^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 43 pour avoir 5.13% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – $6^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.23. Les données détaillées se trouvent en Annexe V.

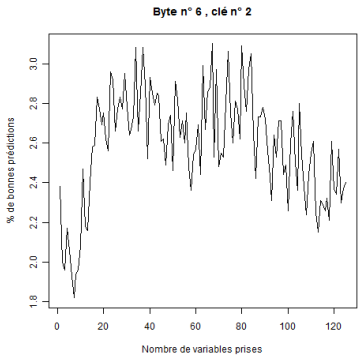


Figure 5.23 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $6^{ème}$ byte de la $2^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 68 pour avoir 3.10% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – $7^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.24. Les données détaillées se trouvent en Annexe U.

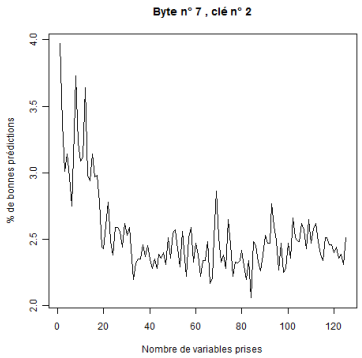


Figure 5.24 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $7^{ème}$ byte de la $2^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 2 pour avoir 3.97% de chances d'avoir l'ensemble du byte.

Attaque de la deuxième clé – $8^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.25. Les données détaillées se trouvent en Annexe X.

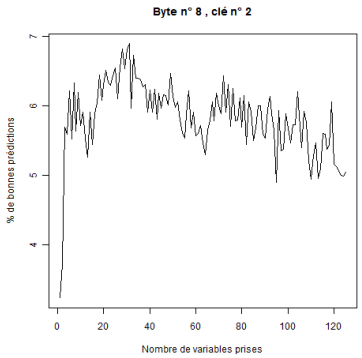


Figure 5.25 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $8^{ème}$ byte de la $2^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 32 pour avoir 6.89% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – 1^{er} byte

Les résultats figurent sur le graphique 5.26. Les données détaillées se trouvent en Annexe J.

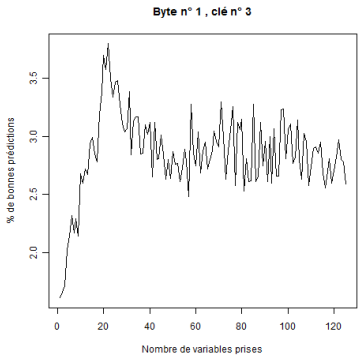


Figure 5.26 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le 1^{er} byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 23 pour avoir 3.80% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – $2^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.27. Les données détaillées se trouvent en Annexe W.

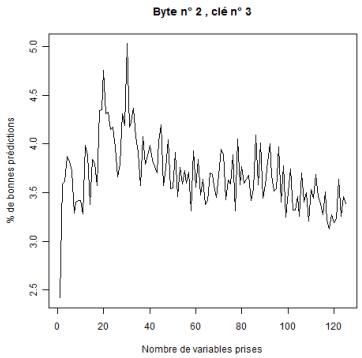


Figure 5.27 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $2^{ème}$ byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 31 pour avoir 5.03% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – $3^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.28. Les données détaillées se trouvent en Annexe Y.

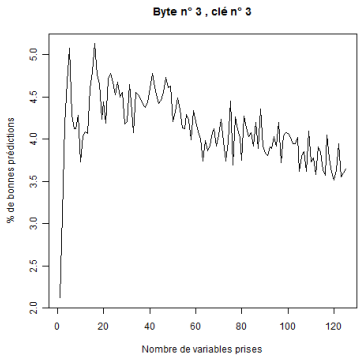


Figure 5.28 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $3^{ème}$ byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 17 pour avoir 5.13% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – $4^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.29. Les données détaillées se trouvent en Annexe ZB.

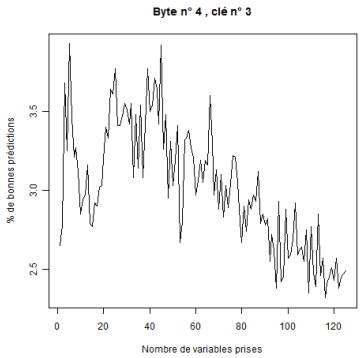


Figure 5.29 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $4^{ème}$ byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 6 pour avoir 3.93% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – $5^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.30. Les données détaillées se trouvent en Annexe Z.

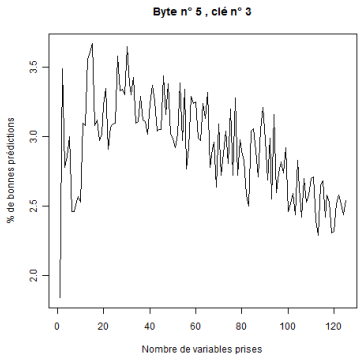


Figure 5.30 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $5^{ème}$ byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 16 pour avoir 3.67% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – $6^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.31. Les données détaillées se trouvent en Annexe ZA.

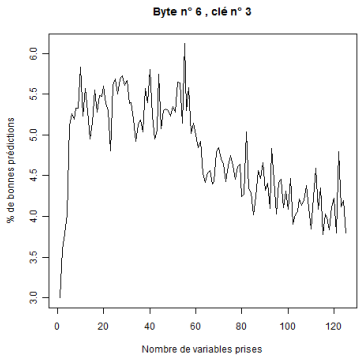


Figure 5.31 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $6^{ème}$ byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 56 pour avoir 6.12% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – $7^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.32. Les données détaillées se trouvent en Annexe ZC.

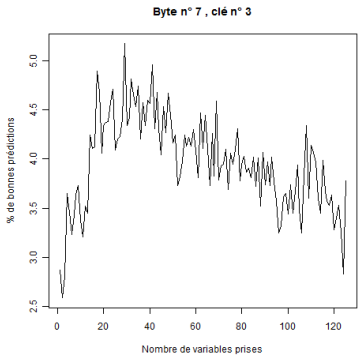


Figure 5.32 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $7^{ème}$ byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 30 pour avoir 5.17% de chances d'avoir l'ensemble du byte.

Attaque de la troisième clé – $8^{\text{ème}}$ byte

Les résultats figurent sur le graphique 5.33. Les données détaillées se trouvent en Annexe F.

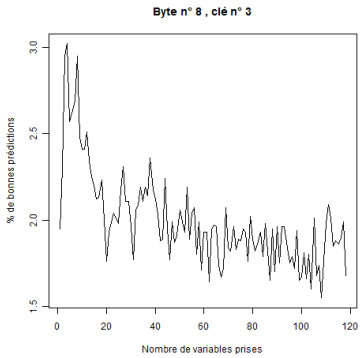


Figure 5.33 : Cette figure montre le pourcentage de bonnes prédictions en fonction du nombre de variables prises pour le $8^{ème}$ byte de la $3^{ème}$ clé.

Elle montre qu'il suffisait de s'arrêter à une dimension 5 pour avoir 3.02% de chances d'avoir l'ensemble du byte.

Résumé

Le tableau suivant résume le taux de prédiction ainsi que du meilleur nombre de variables à prendre pour chaque byte attaqué.

	1^{er} bit	$2^{ème}$bit	$3^{ème}$bit	$4^{ème}$bit	$5^{ème}$bit	$6^{ème}$bit	$7^{ème}$bit	Dim.	%
$1^{ère}$ clé									
1^{er}byte	*78.13*	*65.63*	*77.34*	*60.16*	*60.16*	*53.13*	*50.00*	61	3.81
$2^{ème}$ byte	*85.16*	*75.00*	*67.97*	*50.00*	*57.03*	*50.00*	*50.00*	17	3.09
$3^{ème}$ byte	*78.91*	*67.97*	*70.31*	*69.53*	*67.97*	*50.00*	*51.56*	44	4.59
$4^{ème}$ byte	*85.16*	*73.44*	*60.94*	*57.81*	*50.00*	*50.00*	*54.69*	25	3.01
$5^{ème}$ byte	*89.84*	*78.91*	*65.63*	*60.16*	*64.84*	*52.34*	*50.00*	28	4.75
$6^{ème}$ byte	*82.03*	*73.44*	*60.16*	*59.38*	*50.78*	*54.69*	*60.94*	40	3.64
$7^{ème}$ byte	*69.53*	*67.19*	*61.72*	*50.78*	*54.69*	*50.00*	*50.00*	24	2.00
$8^{ème}$ byte	*78.91*	*72.66*	*56.25*	*50.00*	*53.91*	*50.00*	*50.00*	39	2.17
$2^{ème}$ clé									
1^{er}byte	95.31	67.19	70.31	59.38	53.91	55.47	50.00	18	4.00
$2^{ème}$ byte	78.13	75.00	67.19	59.38	50.00	50.00	57.03	51	3.33
$3^{ème}$ byte	97.66	85.94	65.63	57.81	50.00	50.00	50.00	28	3.98
$4^{ème}$ byte	93.75	84.38	63.28	52.34	57.03	52.34	50.00	41	3.91
$5^{ème}$ byte	92.19	82.81	67.97	63.28	50.00	62.50	50.00	43	5.13
$6^{ème}$ byte	75.00	71.88	64.06	65.63	50.00	50.00	54.69	68	3.10
$7^{ème}$ byte	90.63	69.53	70.31	61.72	56.25	51.56	50.00	2	3.97
$8^{ème}$ byte	91.41	83.59	82.81	67.19	64.84	50.00	50.00	32	6.89
$3^{ème}$ clé									
1^{er}byte	89.84	74.22	62.50	54.69	60.94	50.00	54.69	23	3.80
$2^{ème}$ byte	96.09	82.81	64.06	60.16	65.63	50.00	50.00	31	5.03
$3^{ème}$ byte	95.31	84.38	76.56	54.69	60.94	50.00	50.00	17	5.13
$4^{ème}$ byte	84.38	74.22	68.75	64.06	57.03	50.00	50.00	6	3.93
$5^{ème}$ byte	93.75	81.25	60.94	54.69	57.81	50.00	50.00	16	3.67
$6^{ème}$ byte	90.63	89.84	72.66	68.75	60.16	50.00	50.00	56	6.12
$7^{ème}$ byte	96.88	87.50	64.06	61.72	61.72	50.00	50.00	30	5.17
$8^{ème}$ byte	71.09	66.41	64.06	65.63	50.00	60.94	50.00	5	3.02
moyenne	86.66	76.47	66.89	59.54	56.90	51.79	51.40	31.04	4.05
écart type	8.44	7.39	6.09	5.71	5.71	3.45	2.88	17.38	1.15

Nous constatons qu'en moyenne les premiers bits sont mieux prévisibles que les autres. De plus, en moyenne, le nombre de variables à prendre en compte est d'environ 31 avec un écart type de 17.38. Finalement, en moyenne, le taux de bonnes prédictions est de 4.05

avec un écart type de 1.15.

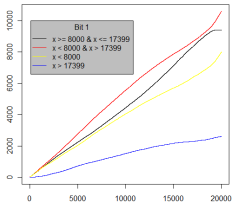
La section suivante traite d'une question que nous nous sommes posés lors du stage.

5.8 Etude des bornes de la trace

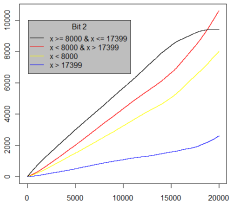
Une question, qui nous intriguait depuis le début, se portait sur les variables utiles prises pour l'apprentissage du modèle. En effet, depuis le début, nous nous étions fixés comme hypothèse que l'information utile se trouvait entre les instants 8000 et 17399. Or, aucune preuve n'a été effectuée. La seule explication serait que le matériel de chiffrement commençait et terminait son travail de chiffrement entre ces deux bornes. Néanmoins, l'envoi d'informations vers et par le matériel de chiffrement n'a jamais été supposé comme pouvant amener une donnée à la cryptanalyse du matériel. Ainsi, pour en être sûr, une technique de sélection de variables a été mise en place pour évaluer la quantité d'informations se trouvant en dehors des bornes 8000 à 17399.

La technique utilisée fut le mRMR qui trie les variables de celles ayant le plus d'informations à celles en ayant le moins. Ainsi, les k premières variables sont celles ayant le minimum de corrélation entre elles, permettant d'avoir le moins de redondance. Autrement dit, ce sont celles ayant le moins d'informations mutuelles entre elles. De plus, ces variables sont aussi celles ayant le maximum d'informations mutuelles avec le bit à prédire. Cette technique a été détaillée dans le chapitre 4, section 3. Le temps de calcul moyen pour chaque bit attaqué fut de quatre jours.

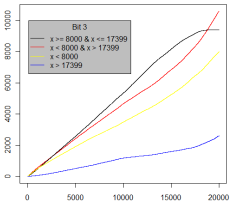
L'analyse de cette nouvelle donnée a été résumée en un graphique qui, chacun, représente un bit à attaquer (Figure 5.34.a, 5.34.b, ..., 5.34.g). Chaque graphique contient quatre courbes. La première courbe en noir montre en abscisse le nombre de variables prises en considération alors que l'ordonnée montre le nombre de variables se trouvant dans la borne 8000 à 17399 parmi celles prises en compte. La deuxième courbe en rouge montre en abscisse le nombre de variables prises en considération alors que l'ordonnée montre le nombre de variables se trouvant en dehors de la borne 8000 à 17399 parmi celles prises en compte. La troisième et quatrième courbe en jaune et en bleu montre en abscisse le nombre de variables prises en considération alors que l'ordonnée montre le nombre de variables se trouvant en dessous de la borne 8000 et au-dessus de la borne 17399 respectivement parmi celles prises en compte. La courbe rouge est donc la somme de la courbe jaune et de la bleue. Ainsi, lorsqu'on a un point en (5000,2238) pour la courbe noire, cela signifie que parmi les 5000 premières variables contenant le plus d'informations, il y a 2238 d'entre elles qui sont dans la borne alors que le reste est en dehors de la borne.



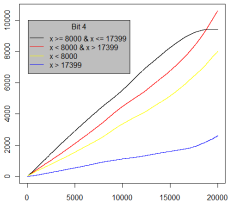
(a) 1^{er} bit



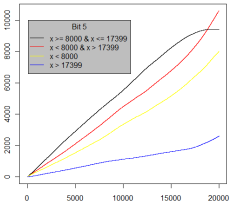
(b) $2^{ème}$ bit



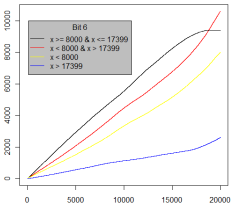
(c) $3^{ème}$ bit



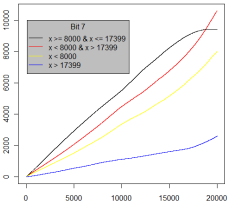
(d) $4^{ème}$ bit



(e) $5^{ème}$ bit



(f) $6^{ème}$ bit



(g) $7^{ème}$ bit

.

Figure 5.34 : La première courbe en noir montre en abscisse le nombre de variables prises en considération alors que l'ordonnée montre le nombre de variables se trouvant dans la borne 8000 à 17399 parmi celles prises en compte. La deuxième courbe en rouge montre en abscisse le nombre de variables prises en considération alors que l'ordonnée montre le nombre de variables se trouvant en dehors de la borne 8000 à 17399 parmi celles prises en compte. La troisième et quatrième courbe en jaune et en bleu montre en abscisse le nombre de variables prises en considération alors que l'ordonnée montre le nombre de variables se trouvant en dessous de la borne 8000 et au-dessus de la borne 17399 respectivement parmi celles prises en compte.

En conclusion, nous voyons que la quantité d'informations se trouvant en dehors et celle se trouvant dans la borne sont très proches. Ainsi, il semblerait intéressant de prendre des données se trouvant en dehors de la borne puisque ceux-ci contiennent de l'information utile pour la cryptanalyse. Une raison à cette découverte viendrait du fait que le FPGA a besoin de la clé de chiffrement. Ainsi, préalablement au chiffrement représenté dans la borne, il est nécessaire d'envoyer la clé au FPGA, qui est visible sur toutes les traces, avant la borne 8000. L'algorithme 5.1 illustre cela. Ceci expliquerait le rapprochement entre la courbe jaune et la courbe noir.

Algorithme 5.1 : Le chiffrement d'une donnée sur le matériel de production.
Le chiffrement d'une donnée est orchestré par un processeur 386. Celui-ci réalise 4 appels au FPGA qui chiffre les données, et qui est par conséquent un coprocesseur :

1. 386→RAM : 386 lit dans la RAM la clé
2. 386→FPGA : 386 écrit la clé dans la mémoire interne du FPGA
3. 386→RAM : 386 lit dans la RAM la donnée à chiffrer
4. 386→FPGA : 386 écrit la donnée à chiffrer dans la mémoire interne du FPGA
5. 386→FPGA : 386 indique au FPGA de commencer le chiffrement

Désormais que nous connaissons mieux le matériel que nous avons attaqué, nous allons nous focaliser sur la mise en place de l'attaque dans un cadre général dans la section suivante.

5.9 Comment mettre la solution en pratique ?

Cette section, que nous pourrions qualifier de tutorial, détaille une proposition de méthode mettant en pratique l'attaque de manière efficace contre un matériel de chiffrement. Il se divise en 7 points représentant les 7 étapes de l'attaque.

La première étape est le choix d'un modèle d'apprentissage automatique. Nous pouvons, par exemple, choisir le RF/PCA pour chaque bit de la clé à attaquer.

Durant la deuxième étape, nous allons prendre l'ensemble des comportements qui caractérise le matériel de chiffrement. Les données collectées peuvent être aussi bien la consommation d'énergie grâce à un oscilloscope que d'autres informations telles que le message chiffré, le message en clair, le temps des opérations, l'émanation électromagnétique, la température du matériel, le son, ... Toutefois, l'ensemble des variables prises n'est probablement pas nécessaire pour caractériser le matériel. C'est pourquoi il est nécessaire de réduire la taille du problème en réduisant la taille des données. Pour cela, nous pouvons utiliser plusieurs techniques de sélection de variables telles que PCA ou mRMR. Notons que cette partie de l'attaque est celle qui prend le plus de temps.

Durant la troisième phase, nous donnons l'ensemble des données collectées à notre modèle d'apprentissage. Après la collecte de données, cette partie est celle qui prend le plus de temps.

Après avoir créé notre modèle, nous le mettons dans un cas réel où le but est de retrouver la clé de la machine attaquée qui est supposée avoir des caractéristiques hardware et software proches de la machine ayant été utilisée durant la phase d'apprentissage. Nous utilisons donc le modèle pour prédire la clé utilisée selon la trace que nous avons collecté sur la machine attaquée. Puisqu'il existe une probabilité non nulle d'avoir une mauvaise

Algorithme 5.2 : Le force brute amélioré.
Soit $\{B_1, B_2, ..., B_{56\times3}\}$, les numéros de bits du plus simple au plus complexe à prédire.
$i = 1$
Temps que nous n'avons pas la clé, faire :
Tester toutes les valeurs possibles pour les i derniers $B_{i...56\times3}$:

$$B_{(56\times3-i+1)}, B_{(56\times3-(i+1)+1)}, ..., B_{(56\times3)}$$

Vérifier si nous n'avons pas trouvé la clé
$i = i + 1$

prédiction, il est nécessaire de vérifier si la clé est correcte. Dans le cas contraire, une méthode par force brute devra être réalisée. Toutefois, cette méthode par force brute se fera en utilisant l'ensemble des informations que nous avons trouvé durant le mémoire, à savoir que certains bits ont plus de probabilités d'être correctes que d'autres. L'algorithme 5.2 résume le principe du force brute amélioré. En implémentant cet algorithme, nous aurons un vecteur de clés potentielles dont les premières sont celles les plus probables. L'attaque est considérée comme terminée si soit la clé de chiffrement est retrouvée, soit la clé de chiffrement a été modifiée par le matériel.

Notons finalement que notre modèle peut s'attaquer à chaque byte de la clé parallèlement ce qui permettrait de diminuer considérablement le temps d'attaque.

Exemple

Prenons un exemple simple pour illustrer l'algorithme. Supposons que nous devons prédire une clé de 8 bits et que notre modèle ait prédit la valeur 0011 1101. Supposons que les derniers bits soient plus complexes à prédire que les autres. Le tableau suivant résume les probabilités de prédire chaque bit :

numéro du bit	1	2	3	4	5	6	7	8
% de bonnes réponses	0.9	0.85	0.80	0.79	0.65	0.60	0.55	0.5

De ce fait, si le modèle n'a pas correctement prédit la clé, nous devrons d'abord modifier les derniers bits. Ainsi, nous testerons d'abord 0011 1101, puis 0011 110**0**, puis 0011 111**1**, puis 0011 11**10**, puis 0011 **10**01, ...

5.10 Conclusion

Nous venons de voir la mise en application de la nouvelle technique d'attaque. Nous avons ainsi trouvé que certains bits sont plus prévisibles que d'autres. De plus, nous avons trouvé une faille dans le matériel attaqué, à savoir que la clé pouvait être vue en clair avant le chiffrement. Enfin, nous avons élaboré une stratégie d'attaque. Le prochaine chapitre fera la comparaison entre la nouvelle technique et les autres.

Chapitre 6

Comparaison RF/PCA versus template Based DPA

Ce chapitre a comme but d'estimer la qualité de notre modèle RF/PCA par rapport à l'ensemble des attaques contre un matériel de chiffrement. Ainsi, nous commencerons par définir la manière dont l'évaluation se fera. Ensuite, nous nous focaliserons sur les résultats.

6.1 Contexte d'évaluation

Basée sur le principe de ne pas refaire la roue, la première idée pour évaluer notre méthode devrait être réalisée en regardant les articles qui ont déjà été publiés dans le même domaine. Toutefois, est-ce vraiment réaliste, utile, efficace ?

En effet, les articles ne s'attaquent pas au même algorithme de chiffrement. Certains chercheurs s'attaquent à DES, d'autres à RSA, à AES ou à des modèles réduits [1] et utilisent des attaques par DPA, SPA ou *template Based DPA*. De plus, les moyens utilisés ne sont pas toujours les mêmes : il existe des matériaux de chiffrement qui sont protégés contre certaines attaques et d'autres non. Enfin, il est raisonnable de considérer le fait que peu de chercheurs travaillent avec les mêmes oscilloscopes, les mêmes ordinateurs, le même nombre de traces collectées, les mêmes matériaux de chiffrement.

L'ensemble de ces variations font que l'efficacité d'une attaque est variable selon la qualité du matériel attaqué ou de celui de l'attaquant. En d'autres termes, nous sommes face à un problème où l'attaque est fortement liée à l'implémentation.

Nous sommes donc confrontés à un réel obstacle : comment comparer des techniques différentes ? Notre choix se base sur deux idées. La première est de choisir la meilleure

1. Un modèle réduit signifie un algorithme non complet tel qu'un DES avec 4 tours au lieu des 16 habituel, ou un RSA avec une clé de 8 bits.

attaque connue à ce jour contre un matériel de chiffrement, à savoir le *template Based DPA*. La deuxième idée est liée à la dépendance que possède l'attaquant vis-à-vis de l'attaqué.

Nous avons réalisé les techniques d'attaque dans les mêmes conditions, impliquant l'utilisation du même oscilloscope, du même appareil de chiffrement, des mêmes sondes, du même nombre de traces, des mêmes traces ainsi que du même ordinateur qui réalise les calculs d'efficacité de chaque attaque. Pour cela, nous avons utilisé les données récoltées par [21].

6.2 Exercice de style

Pour tester notre modèle, nous avons d'abord réalisé un exercice de style en effectuant plusieurs variantes du *template Based DPA*. Toutefois, ce n'est qu'à la prochaine section que nous évaluerons à proprement parlé notre nouveau modèle en mettant le *template based DPA* dans les mêmes conditions que RF/PCA.

Template Based DPA / mRMR - En s'attaquant à tout le byte

Cette première variante[2] du *template based DPA* s'attaque à tout le byte. La sélection de variable s'est faite grâce à mRMR. Chaque template représente une clé parmi les 128 possibles dans un byte où nous ne prenons pas en compte le bit de parité. Nous avons utilisé les 400×128 traces où chaque clé est liée à 400 traces. Pour valider le modèle, nous avons utilisé la validation croisée où 20 traces était utilisées pour la partie validation.

Les résultats détaillés se trouvent dans le fichier AnnexeCodeC.xlsx. Le résumé de ces résultats se trouve sur la figure 6.1.

Nous constatons qu'avec 27 variables, nous obtenons un taux de bonnes prédictions de l'ordre de 10.8 %, qui est toutefois inférieur à celui du RF/PCA. La prise en considération d'un plus grand nombre de variables n'était pas possible pour une raison de temps de calcul.

Template DPA / mRMR - En s'attaquant à un bit à la fois avec les 400×128 traces

Nous avons implémenté la technique où nous attaquons un bit à la fois avec le t*emplate Based DPA* et mRMR. Pour cela, nous avons utilisé uniquement les 400×128 traces où 400 traces étaient liées à une clé. Nous avons validé l'ensemble avec la validation croisée où 1280[3] traces étaient utilisées pour la validation.

2. Le code source de l'attaque se trouve dans le fichier code19c.R.

3. Nous utilisions 10 traces par clé pour la validation. Puisque nous avions 128 clés au total, nous avions 10×128 traces (1280 traces) pour la partie de la validation.

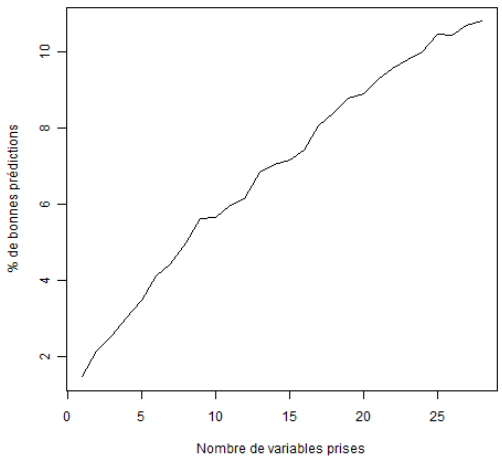


Figure 6.1 : Ce graphique montre la probabilité de prédire le byte en fonction de la quantité de dimensions prises en considération. Cette prévision est faite en s'attaquant à tout le byte. Pour cela, nous avons utilisé les 400×128 traces où chaque clé est liée à 400 traces.

Les résultats sont disponibles dans le fichier AnnexeCodeCA1.xlsx et sont condensés sur la figure 6.2.

Nous constatons qu'avec 4 variables, nous obtenons un taux de bonnes prédictions de 1.44 %. Pour des raisons pratiques, un nombre plus grand de variables prises en considération n'était pas possible.

6.3 Evaluation du modèle

Cette section traite sur les résultats de la comparaison entre le modèle proposé par ce mémoire et le *template Based DPA*.

La première étape est l'implémentation [4] de la technique où nous attaquons un bit à la fois avec le *template Based DPA* / mRMR. Pour cela, nous avons utilisé uniquement les 128 traces représentant la moyenne des traces de chaque byte. Nous avons validé l'ensemble avec le *leave-one-out*. Nous nous trouvons donc dans le même contexte que du modèle RF/PCA.

Les résultats sont disponibles dans le fichier AnnexeCodeCB1.xlsx et sont condensés sur la figure 6.3.

4. Le code source de l'attaque se trouve dans le fichier code19cb.R.

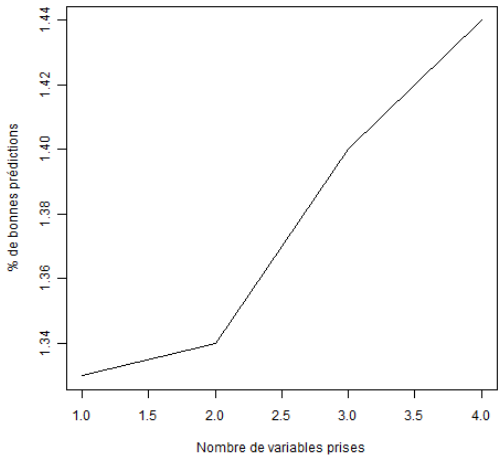


Figure 6.2 : Ce graphique montre la probabilité de prédire le byte en fonction de la quantité de dimensions prises en considération. Cette prévision est faite en s'attaquant à un bit à la fois. Pour cela, nous avons utilisé les 400×128 traces où chaque clé est liée à 400 traces.

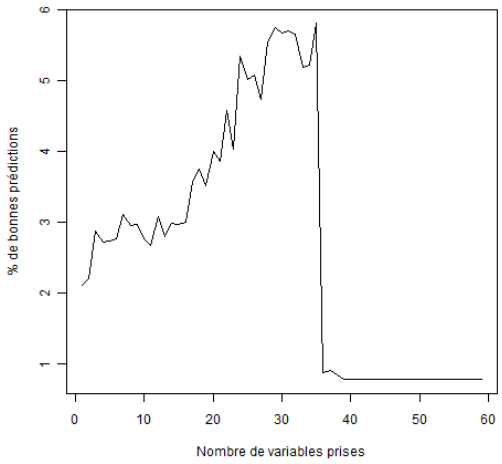


Figure 6.3 : Ce graphique montre la probabilité de prédire le byte en fonction de la quantité de dimensions prises en considération.

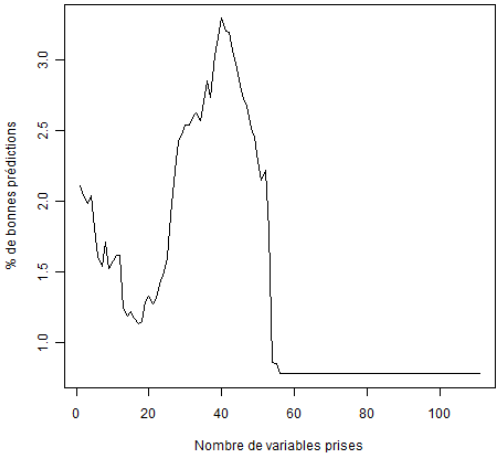


Figure 6.4 : Ce graphique montre la probabilité de prédire le byte en fonction de la quantité de dimensions prises en considération.

Ainsi, nous constatons qu'avec 35 dimensions, nous avons 5.80 % de chance de retrouver le byte. Le taux de bonnes prédictions est donc en dessous du modèle RF/PCA. Le tableau ci-dessous montre les probabilités de retrouver chaque bit en considérant 35 variables.

template Based DPA/mRMR(35)	1	2	3	4	5	6	7
% de bonnes réponses	94.53	78.13	78.13	67.19	53.13	55.47	50.78

Suivant les résultats, nous remarquons qu'au-delà de 35 dimensions, nous ne sommes plus capable de construire un modèle fiable. Ainsi, dans le cas où le modèle devrait prendre en considération plus de variables, il serait donc impossible de le faire. La raison de cette chute vient du fait que nous avons que 64 observations pour chaque valeur du bit attaqué. Le faible nombre d'observations ne permet pas de déterminer avec fiabilité la matrice variance-covariance. Pire encore, au-delà de 59 variables prises en considération, nous obtenons un déterminant nul de la matrice variance-covariance. Cette valeur en induit une infinie dans le calcul des probabilités, induisant une impossibilité de calcul. Pour résoudre le problème d'estimation de la matrice variance-covariance, nous avons utilisé une technique de régulation qui s'appelle le "*Shrinkage estimation cov*"[5]. Les résultats sont détaillés dans AnnexeCodeCB2 et sont résumés sur la figure 6.4.

5. Le « *Shrinkage estimation cov* » est utilisable grâce à la fonction cov.shrink de la librairie corpcor[P3] du langage R. Cette fonction prend en paramètre, entre autres, une valeur λ. Toutefois, nous avons laissé la fonction cov.shrink calculer la meilleure valeur du λ en ne mettant pas de valeurs au λ.

Mise à part le fait que nous n'avons plus de problèmes pour l'estimation des probabilités, nous n'améliorons pas le résultat.

6.4 Estimation du temps d'attaque

En s'inspirant de [49], nous définissons la réussite d'ordre i de l'attaque et le taux de réussite de l'attaque d'ordre i. La première est exacte si parmi les i premières clés du vecteur, la bonne clé s'y trouve. Le second est la probabilité que parmi les i premières clés du vecteur, la bonne clé s'y trouve. Cette probabilité aura une conséquence directe sur le travail qu'il restera à réaliser après la phase de collecte de la trace qui va nous permettre d'attaquer le matériel. Cette probabilité permet donc de décider si le problème à résoudre est raisonnable en temps. Pour calculer le nombre moyen de clés à tester après que nos modèles aient renvoyés une clé potentielle, nous utilisons la formule suivante, nommée « *guessing entropy* » [49] :

$$\sum_{j=1}^{2^N} P(\text{avoir la clé à la } j^{ème} \text{ étape}) \times j$$

Dans cette formule, 2^N est le nombre maximum de permutations à réaliser sur la clé potentielle de N bits pour trouver la clé finale. Notons que l'algorithme 5.2, représentant la stratégie d'attaque, est fortement liée à cette formule. Ainsi, à chaque étape, nous allons permuter une valeur d'un bit. Ce bit sera celui qui est le plus dur à prédire, en d'autres termes, celui dont la probabilité que le modèle réussisse à le prédire est le plus faible.

Prenons un exemple. Soit une clé de 3 bits prédite par nos modèles : 101. Supposons que nos modèles réussissent à prédire chaque bit de la clé avec les probabilités suivantes :

- $P(\text{prédire le } 1^{er} \text{ bit}) = 0.9$
- $P(\text{prédire le } 2^{ème} \text{ bit}) = 0.7$
- $P(\text{prédire le } 3^{ème} \text{ bit}) = 0.5$

Notons que nous avons 3 bits dans la clé, donc 2^3 (=8) clés possibles. Nous permuterons la clé prédite en changeant d'abord les derniers bits puisque ceux-ci semblent être les plus durs à prédire.

Le nombre d'étapes qu'il faudra réaliser en moyenne pour retrouver la clé est de :

$$P(\text{prédire la clé à la } 1^{ère} \text{ fois}) \times 1 + ... + P(\text{prédire la clé à la } 8^{ème} \text{ fois}) \times 8$$

$$= (0.9 \times 0.7 \times 0.5) \times 1 + (0.9 \times 0.7 \times 0.5) \times 2 + (0.9 \times 0.3 \times 0.5) \times 3 + (0.9 \times 0.3 \times 0.5) \times 4$$

$$+(0.1 \times 0.7 \times 0.5) \times 5 + (0.1 \times 0.7 \times 0.5) \times 6 + (0.1 \times 0.3 \times 0.5) \times 7 + (0.1 \times 0.3 \times 0.5) \times 8$$

$$= 2.5$$

Ainsi, il faudrait en moyenne entre 2 et 3 essais de clés pour retrouver la bonne. En d'autres termes, nous avons une clé de 1 à 2 bits ($2.5 \approx 2^1$, $2.5 \approx 2^2$) en réalité et non de 3. Ainsi, il faudrait tester les clés suivantes : 101, 100 et 111.

En supposant que la taille de la clé est de 7 bits, nous pouvons déduire grâce aux données de [21] qu'il est nécessaire d'avoir en moyenne $10.8368(\approx \mathbf{11})$ clés pour retrouver la bonne grâce au modèle RF/PCA alors qu'il en faut $20.8808(\approx \mathbf{21})$ avec le *template Based DPA*.

6.5 Conclusion

Nous constatons que la combinaison entre apprentissage automatique et cryptanalyse donne des résultats surprenants. En effet, nous constatons une amélioration de la meilleure technique d'attaque connue à ce jour. Cette amélioration pourrait venir du fait que les données ne suivent pas une distribution normale.

Pour tester la normalité des données, aucune méthode efficace n'existe pour des problèmes impliquant beaucoup de variables et peu de données. Ainsi, la seule méthode que nous avons utilisée pour évaluer cela est de tester une attaque qui fait l'hypothèse de normalité et une autre qui ne la fait pas. Ces résultats montreraient que les données ne suivent pas totalement une distribution gaussienne ou qu'il n'est pas possible de déterminer avec une précision suffisante les paramètres de la distribution avec les données mises à disposition.

Chapitre 7

Conclusion

Parallèlement à la réalisation d'un modeste état de l'art sur la cryptologie et l'apprentissage automatique, ce mémoire a permis d'avoir une approche pratique de la combinaison entre ces deux domaines.

La première période de ce mémoire fut la plus longue, à savoir la période de collecte brute de données d'un matériel de chiffrement. Elle s'est faite grâce à un stage de 6 mois chez Atos Worldline. La prise a été réalisée de manière manuelle avant un processus totalement automatique mélangeant une synchronisation entre un oscilloscope, un ordinateur pour gérer et collecter les informations et un FPGA pour le chiffrement de données.

Puis, un temps non négligeable a demandé l'exécution des algorithmes d'apprentissage automatique. Ces derniers ont permis de se rendre compte qu'un algorithme de chiffrement, qui est sécurisé en théorie, ne l'est pas forcément en pratique. En effet, nous avons détecté que le matériel de chiffrement recevait la clé en clair avant le chiffrement, permettant une prévision plus simple de la clé grâce aux traces collectées. La faille de sécurité réside donc dans la « non coopération » entre différents domaines tels que l'électronique et la cryptographie. Ainsi, ceux qui construisent un algorithme cryptographique, devraient travailler avec ceux le mettant en pratique.

Toutefois, la plus grande contribution de ce mémoire est d'avoir montré qu'il existe une nouvelle méthode très prometteuse, à savoir la combinaison entre l'apprentissage automatique et une technique d'attaque venant de la cryptanalyse. Après une recherche d'un an, elle semble être plus dangereuse que la plus puissante technique d'attaque contre un matériel de chiffrement, à savoir le *template Based DPA*. En effet, dans le contexte d'un FPGA implémentant un 3DES constitué de trois clés différentes, le modèle final RF/PCA a montré que nous pouvions retrouver un certain byte de la clé dans 15.33% des cas alors qu'il est de 5.80% avec la meilleure technique d'attaque. En utilisant la notion de *guessing entropy*, et en ne s'attaquant qu'à un byte de la clé, nous pouvons en conclure qu'il faut tester en moyenne **11** clés, après que le modèle RF/PCA l'ait prédite, alors qu'il en faut **21** pour le *template Based DPA*. Outre le nombre accru de clés à tester

en moyenne, nous avons constaté qu'au-delà de 35 dimensions, le template Based DPA n'était plus capable de construire un modèle fiable. Pire encore, aux delà de 59 variables, nous avions un problème pour calculer les prédictions du modèle. La raison venait d'une introduction d'une valeur infinie dans le calcul de la prévision. Nous avons essayé de résoudre ce problème en utilisant le "*Shrinkage estimation cov*". Bien que nous n'avions plus de problèmes pour les calculs grâce à ce dernier, les résultats n'étaient toutefois pas meilleurs.

Outre la comparaison de deux attaques différentes, nous avons attaqué chaque byte de la clé individuellement grâce au nouveau modèle. Cette partie nous a permis de se rendre compte que certains bits - en majorité les plus éloignés du bit de parité - sont plus prévisibles que les autres. Cette constatation nous a amenés à créer un algorithme, un dérivé de la force brute, qui cherche la clé, après que le modèle RF/PCA l'ait prédite, en prenant en compte le taux de bonnes prédictions du modèle.

Notons que l'ensemble de la clé peut être attaqué avec un système distribué où chaque modèle attaque un bit de la clé. De plus, il n'est pas indispensable de connaitre l'algorithme implémenté dans le FPGA ni d'avoir une connaissance approfondie de ce dernier pour réaliser l'attaque [1], telle que le moment de chiffrement ou les opérations possibles. En effet, RF/PCA ne fait que le lien entre la consommation d'énergie et la clé utilisée. Ainsi, ce type d'attaque peut être généralisé sur d'autres algorithmes de chiffrement. Enfin, nous pourrions généraliser pour passer à un « *Multi-Channel Attacks* », en mettant en entrée non seulement des traces, mais aussi le temps de chiffrement, le message chiffré, le message clair, le son,... Notons que dû au fait que le *template Based DPA* n'a pas pu dépasser les 35 variables d'entrée, nous sommes très dubitatifs sur ses capacités à réaliser du *multi-channel attack*.

En ce qui concerne le long terme, ce mémoire n'est pas une fin en soi. Grâce au côté novateur de la découverte, plusieurs pistes restent encore à explorer telles que :

- Regarder si la fréquence d'utilisation d'un bit influence la précision de sa prédiction.
- De manière générale, comprendre pourquoi certains bits sont plus prévisibles que d'autres.
- Travailler avec d'autres modèles du domaine de l'apprentissage automatique.
- Travailler avec d'autres techniques de sélection de variables.
- Travailler sur un autre algorithme de chiffrement (RSA, RC4,...).
- Travailler sur un autre matériel de chiffrement.
- Travailler avec le temps de chiffrement, le son du matériel de chiffrement, le message chiffré,.... .
- Chercher le meilleur nombre de traces à prendre pour réduire le bruit.
- Chercher le meilleur nombre d'instants à prendre par trace.

1. Ceci va dans l'hypothèse de base, bien connue, de Kerckhoff qui est que seule la clé est le secret réel.

- Voir s'il y a des protections contre la nouvelle attaque.
- Réaliser un programme automatisant l'attaque.
- Essayer de s'attaquer non pas à un byte à la fois mais à tous.

Pour conclure ce mémoire, je partagerai une constatation que j'ai pu remarqué tout au long de mes études, que ce soit dans des cours de Management Humain, de Philosophie ou de Cryptanalyse : "*Il faut essayer de comprendre ce que veut dire une personne ou un matériel par ses gestes, sa posture, le contexte, son comportement ou son passé et non pas uniquement par les mots qu'ils prononcent.*".

Bibliographie

[1] D. Agrawal & J.R. Rao & P. Rohatgi, (2003), "*Multi-Channel Attacks*", in the proceedings of Cryptographic Hardware and Embedded Systems (CHES) 2003, LNCS, vol 2779, pp 2-16, Cologne, Germany.

[2] D. Agrawal & J. R. Rao & P. Rohatgi & K. Schramm, (2005),"*Templates as Master Keys*", Cryptographic Hardware and Embedded Systems (CHES) 2005, pp. 15-29, Edinburgh, UK.

[3] C. Archambeau & E. Peeters & F.-X. Standaert & J.-J. Quisquater, (2006) ,"*Template Attacks in Principal Subspaces*", 8th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Yokohama, Japan, 10-13 October, Lecture Notes in Computer Science vol. 4249, pp. 1-14. Springer.

[4] J. Asselin de Beauville & F. Kettaf, (29/04/2005) ,"Bases théoriques pour l'apprentissage et la décision en reconnaissance des formes", ISBN. : 2.85428.668.5, Cépaduès.

[5] _E.Biham & A. Shamir, (1993), "*Differential cryptanalysis of the data encryption standard*", Springe-Verlag, London, UK, ISBN-13 : 9780387979304.

[6] _D. Boneh & D. Lie & P. Lincoln & J. Mitchell & M. Mitchell, (2000), "*Hardware Support for Tamper-Resistant and Copy-Resistant Software*", Stanford University, Stanford, CA, USA.

[7] G. Bontempi, (2006), "*Statistical foundations of machine learning*", Computer Science Department, Université Libre de Bruxelles, Belgium.

[8] L. Breiman. (1994), "*Heuristics of instability in model selection*", Technical Report, Statistics Department, University of California at Berkeley.

[9] E. Brier & C. Clavier & F. Olivier,(1999),"*Correlation Power Analysis with a Leakage Model*", In proceedings of CHES 2004, LNCS 3156, pp. 16-29, Springer.

[10] C. Canovas & J. Clédière, (2005), "*What do S-boxes say in differential side channel attacks ?*", Cryptology ePrint Archive, Report 2005/311.

[11] S. Chari & J. R. Rao & P. Rohatgi, (2002), "*Template Attacks* ", in CHES, volume 2523 of LNCS, pages 13–28. Springer.

[12] C. Clavier, (2007),"*De la sécurité physique des crypto-systèmes embarqués*", Laboratoire de recherche en informatique, Université de Versailles Saint-Quentin.

[13] N. Cristianini & J. Shawe-Taylor, (2000), "Support Vectors Machines and other kernel-based learning methods", Cambridge University Press.

[14] J. F. Dhem & F. Koeune & P. A. Leroux & P. Mestré & J. J. Quisquater & J. L. Willems, (1998), "*A Practical Implementation of the Timing Attack*" in Proceedings of CARDIS 1998, pp. 167–182.

[15] W. Diffie & M. Hellman, (June 1977), "*Exhaustive Cryptanalysis of the NBS Data Encryption Standard*", Computer, vol. 10, no. 6, pp. 74-84.

[16] W. Diffie & M. E. Hellman. (1976), "*New directions in cryptography*", IEEE Transactions on Information Theory, IT-22(6) : 644–654.

[17] C. Ding & H.C. Peng, (2003), "*Minimum Redundancy Feature Selection from Microarray Gene Expression Data*", Proc. Second IEEE Computational Systems Bioinformatics Conf., pp. 523-528.

[18] M. Dworkin, (2001), "*Recommendation for Block Cipher Modes of Operation*", U.S. Department of Commerce, National Institute of Standards and Technology.

[19] T. Eisenbarth & T. Kasper & A. Moradi & C. Paar & M. Salmasizadeh & M. T. Manzuri Shalmani, (2008), "*On the Power of Power Analysis in the Real World : A Complete Break of the KeeLoq Code Hopping Scheme*", pp. 203-220, vol. 5157, CRYPTO.

[20] M.A. El Aabid & S. Guilley & P. Hoogvorst, (2007), "*Template Attacks with a Power Model*", Cryptology ePrint Archive, Report.

[21] S. Fernandes Medeiros, (2009), "*Power Analysis Attacks*", Computer Science Department, Université Libre de Bruxelles, Belgium.

[22] K. Gandolfi & C. Mourtel & F. Olivier. (2001), "*Electromagnetic analysis : Concrete results*", CHES 2001, C¸ . K. Koc¸, D. Naccache, and C. Paar, Eds., vol. 2162 of LNCS, pp. 255–265, Springer-Verlag.

[23] L. Goubin & J. Patarin, (1999), "*DES and Differential Power Analysis (The "Duplication" Method)*", CHES 1999, pp. 158-172

[24] Y. Han & X. Zou & Z. Liu & Y. Chen, (2008),"*Efficient DPA Attacks on AES Hardware Implementations*", I. J. Communications, Network and System Sciences, Scientific Research Publishing.

[25] P. Karsmakers & B. Gierlichs & K. Pelckmans & K.D. Cock & J. Suykens & B. Preneel & B.D. Moor, (2007), "Si*de channel attacks on cryptographic devices as a classification problem*", Associatie K.U.Leuven, Leuven, Belgium.

[26] F. Koeune & F.-X. Standaert., (2005), "*A Tutorial on Physical Security and Side-Channel Attacks*", 5th International School on Foundations of Security Analysis and Design FOSAD, pp 78-108, LNCS vol 3655.

[27] B. Kopf & D. Basin, (2007), "*An Information Theoretic Model for Adaptive Side-Channel Attacks*", in the proceedings of ACMCCS 2007, Alexandria, VA, USA.

[28] T. S. Messerges & E. A. Dabish & R. H. Sloan, (1999), "*Investigations of Power Analysis Attacks on Smartcards*", In Proc. of the usenix Workshop on Smart-card Technology (Smartcard'99). usenix Association.

[29] P. C. Kocher. (1996), "*Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*", Neal Koblitz, Advances in Cryptology – CRYPTO'96, volume 1109 de Lecture Notes in Computer Science, pages 104–113. Springer-Verlag.

[30] P. C. Kocher & J. Jaffe & B. Jun. (1999), "*Differential Power Analysis : Leaking Secrets*", In Proc. Crypto '99, Springer-Verlag, LNCS 1666, pages 388–397.

[31] P. C. Kocher & J. Jaffe & B. Jun. (1998), "*Introduction to differential power analysis and related attacks*", Rapport Technique, Cryptography Research.

[32] L. Lerman, (2008), "*Cryptographie quantique*", Computer Science Department, Université Libre de Bruxelles, Belgium.

[33] L. Lerman, (2009), "*Les systèmes de detections d'intrusion basés sur du machine learning*", Computer Science Department, Université Libre de Bruxelles, Belgium.

[34] S. Mangard & E. Oswald & T. Popp, (2007), "*Power Analysis Attacks*",Springer US.

[35] S. Mangard & E. Oswald & T. Popp, (2007), "*Power Analysis Attacks : Revealing the Secrets of Smart Cards*", Springer, Cambridge, Massachusetts.

[36] O. Markowitch, (2000-2001), "*Les protocoles de non-répudiation*", Faculté des Sciences, ULB.

[37] O. Markowitch, (2009), "*Cours de Cryptologie distribuée et protocoles*", Computer Science Department, Université Libre de Bruxelles, Belgium.

[38] R. Mayer-Sommer. (2000), "*Smartly analyzing the simplicity and the power of simple power analysis on smartcards*", Proc. Second Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES 2000), pp. 78-92, 2000.

[39] R.C. Merkle & M. Hellman, (1981), "*On the Security of Multiple Encryption*", Communications of the ACM, vol.24, pp 465-467.

[40] K. Pearson, (1901), "*On Lines and Planes of Closest Fit to Systems of Points in Space*", Philosophical Magazine 2 (6), 559–572.

[41] H. Peng & F. Long & C. Ding, (2005), "*Feature Selection based on Mutual Information : Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy*", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 27, No 8, pp 1226-1238.

[42] L. Personnaz & I. Rivals, (2003), "Réseaux de neurones formels pour la modélisation, la commande et la classification", CNRS Editions.

[43] J-J. Quisquater & D. Samyde, (2002), "*Cryptanalyse par side channel*", Groupe Crypto, Université Catholique de Louvain.

[44] R. L. Rivest, (1993), "*Cryptography and Machine learning*", Laboratory for Computer Science, Massachusetts Insitute of Technology, Cambridge.

[45] S.S. Shapiro & M.B. Wilk, (1965), "An analysis of variance test for normality (complete samples)", Biometrika, 52, 591-611.

[46] P. W. Shor, (1997), "*Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*", SIAM Journal on Computing, vol. 26, pages 1484 – 1509.

[47] F.X. Standaert & B. Gierlichs & I. Verbauwhede, (2008), "*Partition vs. Comparison ,Side-Channel Distinguishers*", In Information Security and Cryptology - ICISC 2008 : 11th International Conference,Springer-Verlag, pp. 253-267.

[48] F.-X. Standaert & C. Archambeau, (2008), "*Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages*", in the proceedings of CHES 2008, Lecture Notes in Computer Science, vol 5154, pp 411-425, Washington DC, USA, Springer.

[49] F.-X. Standaert & T. G. Malkin & M. Yung, (2009), "*A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks*", in International Conference on the Theory and Applications of Cryptographic Techniques - Eurocrypt 2009, ser. LNCS, vol. 5479. Springer, pp. 443–461.

[50] D. R. Stinson, (2006), "*Cryptography, Theory and Practice (Third Edition)*", Chapman & Hall/CRC, Boca Raton.

[51] Y. Swan & G. Latouche, (2008-2009), "*Modèles stochastiques des systèmes informatiques*", ULB, Belgique.

[52] G. Van Assche, (2006), "*Quantum cryptography and secret-key distillation*", Cambridge University Press, ISBN-13 : 978-0521864855.

[53] S. Vaudenay, (2006), "*A classical Introduction to Cryptograph*", Applications for Communications Security, Springer, ISBN-10 : 0-387-25464-1.

[54] Vladimir Vapnik & A. Lerner, (1963), "Pattern Recognition using Generalized Portrait Method", Automation and Remote Control, vol. 24.

[P1] S. Dray & A.B. Dufour, (2007), *"The ade4 package : implementing the duality diagram for ecologists*", Journal of Statistical Software, vol. 22, numéo 4, pp. 1-20.

[P2] A. Liaw & M. Wiener, (2002), "*Classification and Regression by randomForest*", R News, vol. 2, numéro 3, pp. 18-22, http ://CRAN.R-project.org/doc/Rnews/.

[P3] J. Schaefer & R. Opgen-Rhein & K. Strimmer, (2010), "*corpcor : Efficient Estimation of Covariance and (Partial) Correlation*", http ://CRAN.R-project.org/package=corpcor.

[P4] R. Wehrens & L.M.C. Buydens, (2007), "*Self- and Super-organising Maps in R : the kohonen package",* J. Stat. Softw., vol. 5, numéro 5, http ://www.jstatsoft.org/v21/i05.

Chapter 8

Annexe

Annexe 1

Ci dessous se trouve les différentes S_i, appelée fonction de sélection. Le regroupement de ces fonctions S_i constituent la S-Boxes du DES.

S_1 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8 :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Annexe 2

Les slides de la présentation sont disponible dès la page suivante.

Side Channel Attacks & Machine Learning

Présentateur : Liran LERMAN
Promoteurs : Gianluca BONTEMPI & Olivier MARKOWITCH

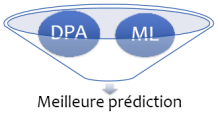
ULB
Faculté des Sciences
Département des Sciences Informatiques
24 mars 2010

Plan

- Introduction
  Y a-t-il a lien entre clé et consommation d'énergie ?
- Recherche du meilleur modèle
- Recherche du meilleur feature selection
- Attaque des bytes des clés
- Stratégie d'attaque
- Découverte
- Future ?

Introduction

- Présentateur
  - Liran LERMAN
  - 2ème année de master en Sciences Informatique à l'ULB
  - Mémoire : Side Channel Attacks & Machine learning
- Stage
  - Suite de la cryptanalyse DPA effectuée par Stéphane Fernandes Medeiros
  - Avec des techniques de machine learning
  - Calcul dans Hydra
    - En cours
  - Code source en R
    - Open source
    - Utilisé dans le monde scientifique

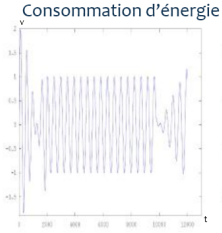


Meilleure prédiction

DPA = Differential Power Analysis

Introduction

Consommation d'énergie par unité de temps

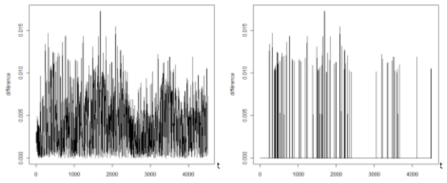

Clé 1


Clé 2

$Z_1 \dots Z_{20000}$
-> 20000 variables

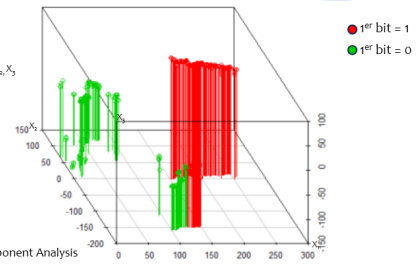
Introduction

Y a-t-il a un lien entre consommation d'énergie et clé utilisée ?


Différence entre clé FF et 00

Introduction

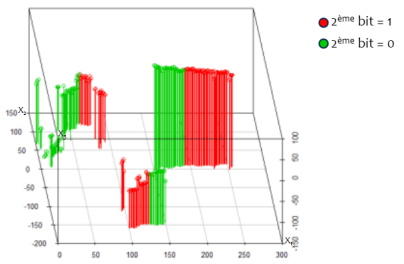
Bit 1
PCA :
- $Z_1 \dots Z_{20000} \rightarrow X_1, X_2, X_3$



PCA = Principal Component Analysis

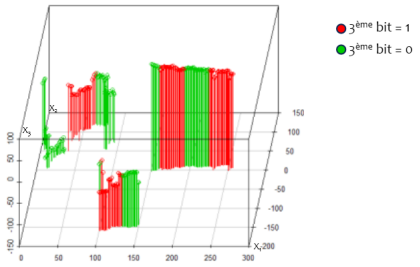
Introduction

Bit 2



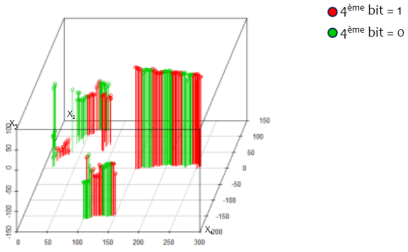
Introduction

Bit 3



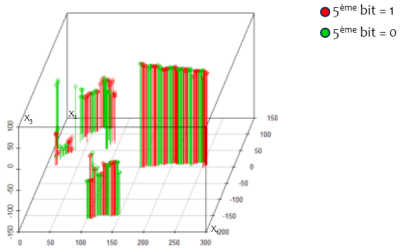
Introduction

Bit 4



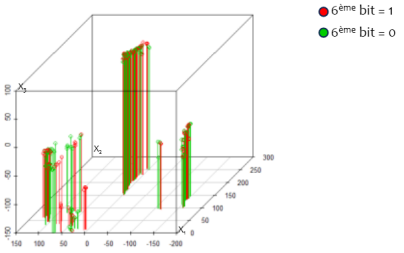
Introduction

Bit 5



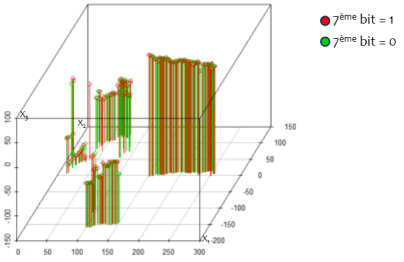
Introduction

Bit 6

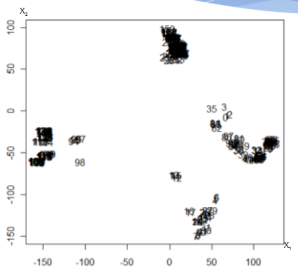


Introduction

Bit 7

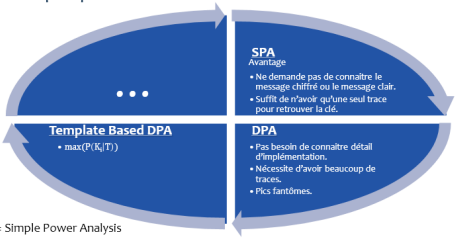


Introduction



Introduction

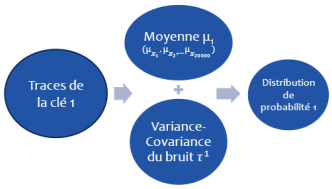
Techniques possibles actuelles



SPA = Simple Power Analysis

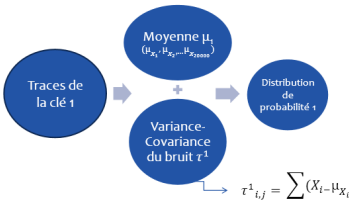
Introduction

* Template Based DPA – Phase d'apprentissage



Introduction

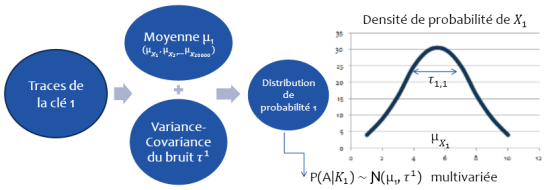
* Template Based DPA – Phase d'apprentissage



$$\tau^{1}_{i,j} = \sum (X_i - \mu_{X_i})(X_j - \mu_{X_j})$$

Introduction

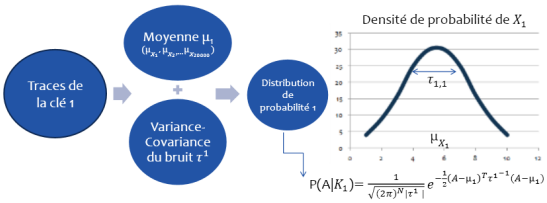
* Template Based DPA – Phase d'apprentissage



$P(A|K_1) \sim \mathcal{N}(\mu, \tau^1)$ multivariée

Introduction

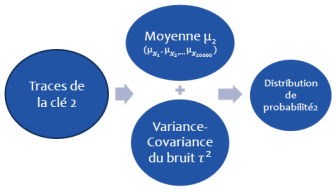
* Template Based DPA – Phase d'apprentissage



$$P(A|K_1) = \frac{1}{\sqrt{(2\pi)^N |\tau^1|}} e^{-\frac{1}{2}(A-\mu_1)^T \tau^{1-1}(A-\mu_1)}$$

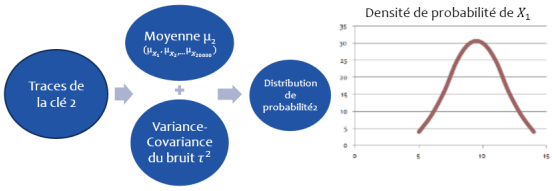
Introduction

* Template Based DPA – Phase d'apprentissage



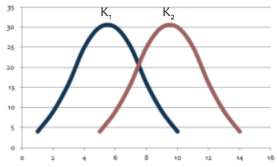
Introduction

* Template Based DPA – Phase d'apprentissage



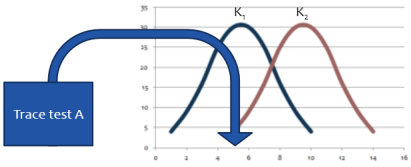
Introduction

❖ Template Based DPA – Phase d'apprentissage



Introduction

❖ Template Based DPA – Phase d'attaque



$$\max_i (P(A|K_i)) = \max_i \left(\frac{1}{\sqrt{(2\pi)^N |\tau^i|}} e^{-\frac{1}{2}(A-\mu_i)^T \tau^{i-1} (A-\mu_i)}\right)$$

Introduction

Template Based DPA connu comme étant le plus puissant

Ne nécessite qu'une seul trace lors de l'attaque

Ne nécessite pas d'avoir les messages chiffrés ou les messages en clair

Prends en compte l'ensemble des informations disponible dans une trace

➢ Pourquoi essayer avec du machine learning?

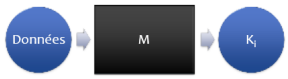
Introduction

❖ Avantage

- ❖ Template Based DPA & Machine learning plus puissant que Template Based DPA
  - ❖ Template Based DPA fait une hypothèse de normalité des données
    - ❖ Or rien n'est totalement gaussien
  - ❖ Modèle non paramétrique -> bon avec peu de données
  - ❖ Machine learning permet de prendre en compte en même temps :
    - ❖ Courant
    - ❖ Temps de chiffrement
    - ❖ Le message chiffré
    - ❖ …

Recherche du meilleur modèle

Machine learning a 3 étapes
- Récolter des données
- Pour chaque structure de modèle
  - Apprentissage des données
  - Validation du modèle
- Test Final

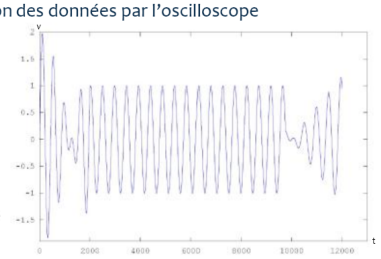


But
- trouver le meilleur modèle
  - RF ?
  - SVM ?
  - … ?

Recherche du meilleur modèle
Récolte de données

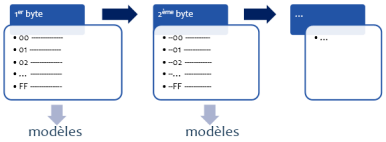
- Récupération des données par l'oscilloscope

Consommation d'énergie par unité de temps



Recherche du meilleur modèle
Récolte de données

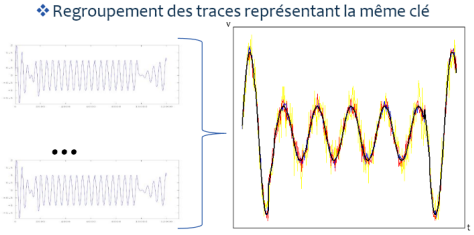
Ces données représentent
- Consommation d'énergie par unité de temps
- Seul 1 byte de la clé variait



- Le message chiffré ne variait pas
  - Chiffrement par 3DES (3 clés différentes)
    - 56*3 bits à prédire

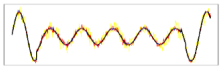
Recherche du meilleur modèle
Récolte de données

Pour diminuer le bruit :
- Regroupement des traces représentant la même clé



Recherche du meilleur modèle
Récolte de données

Conséquence
- Obtention de 256 traces différentes
- Une trace liée à une clé

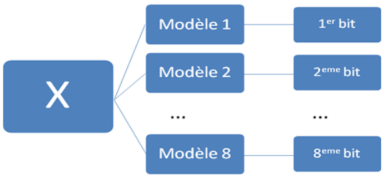

= 00

•••


= FF

Recherche du meilleur modèle
Modèle 1

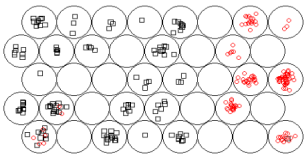
Créer un modèle supervisé pour chacun des 8 bits à attaquer



Recherche du meilleur modèle
Modèle 1

But du modèle i

- Couper les traces en deux ensembles
  - Ceux ayant le bit i à 0
  - Ceux ayant le bit i à 1



Recherche du meilleur modèle
Modèle 1 - Validation

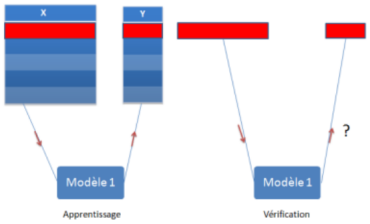
But

- Savoir si le modèle a compris notre problème

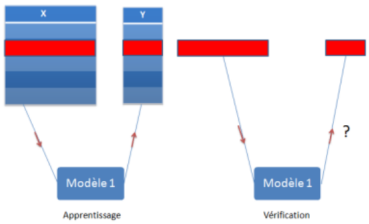
A cause du manque de données (256 traces au total) :

- Utilisation de la technique leave-one-out
- 256 étapes

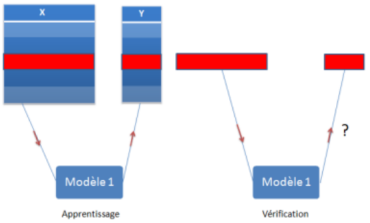
Recherche du meilleur modèle
Modèle 1 - Validation



Recherche du meilleur modèle
Modèle 1 - Validation



Recherche du meilleur modèle
Modèle 1 - Validation



Recherche du meilleur modèle
Modèle 1 - Validation

Avantage

- Voir si le modèle sait prédire un bit de la clé d'une trace inconnue lors de l'apprentissage
- Ne pas être biaisé par les données d'apprentissage lors de la validation du modèle

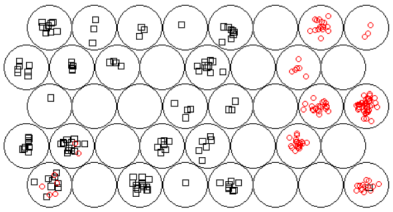
Recherche du meilleur modèle
Modèle 1 - Validation

Résumé

- 256 validations par modèle
- Chaque validation permet de savoir :
  - TP = *le modèle a prédit correctement 0*
  - TN = *le modèle a prédit correctement 1*
  - FN = *le modèle a prédit 0 alors qu'il aurait dû prédire 1*
  - FP = *le modèle a prédit 1 alors qu'il aurait dû prédire 0*
- Utilisation de 7 modèles
- Temps d'apprentissage et de validation :
  - 24h à 48h

Recherche du meilleur modèle
Modèle 1 - SOM(8x5)

SOM de taille 8x5



SOM = Self Organizing Map

Recherche du meilleur modèle
Modèle 1 - SOM(8x5)

SOM(8x5)	1	2	3	4	5	6	7
TP	125	115	115	101	69	58	55
TN	121	116	109	89	68	65	59
FN	7	12	19	39	60	63	69
FP	3	13	13	27	59	70	73
% de bonnes réponses	96.09	90.23	87.50	74.22	53.52	48.05	44.53

Résultats semblables à ceux de Stéphane
Avec une meilleure prédiction

Stéphane	1	2	3	4	5	6	7
% de bonnes réponses	95.70	63.67	61.72	54.29	55.86	51.95	53.51

Recherche du meilleur modèle
Modèle 1 - SOM(8x5)

Conclusion

- On est passé de 0.78% de chance de trouver le byte à 7,53%

Recherche du meilleur modèle
Modèle 1 – Som()

Augmenter la complexité du modèle
But

- Augmenter la précision sur sa prédiction

Recherche du meilleur modèle
Modèle 1 - SOM(9x5)

SOM(9x5)	1	2	3	4	5	6	7
TP	126	116	113	95	79	66	60
TN	123	120	102	83	69	65	48
FN	5	8	26	45	59	63	80
FP	2	12	15	33	49	62	68
% de bonnes réponses	97.27	92.19	83.98	69.53	57.81	51.17	42.19

Conclusion

- On est passé de 0.78% de chance de trouver le byte à 7,74%

Recherche du meilleur modèle
Modèle 1 - SOM(8x6)

SOM(8x6)	1	2	3	4	5	6	7
TP	127	112	106	94	76	58	52
TN	120	117	108	94	71	59	49
FN	8	11	20	34	57	69	79
FP	1	16	22	34	52	70	76
% de bonnes réponses	96.48	89.45	83.59	73.44	57.42	45.70	39.45

Conclusion

- On est passé de 0.78% de chance de trouver le byte à 7,61%

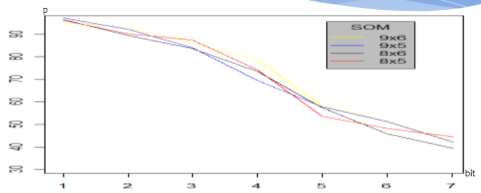
Recherche du meilleur modèle
Modèle 1 - SOM(9x6)

SOM(9x6)	1	2	3	4	5	6	7
TP	125	118	110	104	76	70	56
TN	120	120	110	97	73	62	53
FN	8	8	18	31	55	66	75
FP	3	10	18	24	52	58	72
% de bonnes réponses	95.70	92.97	85.94	78.52	58.20	51.56	42.58

Conclusion

- On est passé de 0.78% de chance de trouver le byte à 9,01%

Recherche du meilleur modèle
Modèle 1 - Conclusion



- Certains bits sont mieux prédits avec un modèle « simple » (ex : 3ème bit)
- Certains bits sont mieux prédits avec un modèle « complexe » (ex : 5ème bit)

Recherche du meilleur modèle

Utilisation d'autres modèles
- SVM
- RF

Utilisation de « techniques » de feature selection
- Nosel
- Rank

But des techniques de feature selection :
- Diminuer taille des données
- Diminuer temps d'apprentissage
- Diminuer temps de prédiction
- Diminuer bruit (données inutiles enlevées)
- …

SVM = Support Vector Machine
RF = Random Forest

Recherche du meilleur modèle

Validation réalisée par le leave-one-out vue précédemment

Recherche du meilleur modèle
Modèle 2 - SVM

But du modèle i
- couper l'ensemble des traces en deux ensembles
  - Ceux ayant le bit i à 0
  - Ceux ayant le bit i à 1

Différences par rapport au SOM
- Division réalisée avec un hyperplan
  - → algorithme d'apprentissage différent

Recherche du meilleur modèle
Modèle 2 - SVM/Rank

SVM/Rank	1	2	3	4	5	6	7
TP	119	102	94	79	40	60	56
TN	123	104	92	81	78	70	33
FN	9	26	34	49	88	68	72
FP	5	24	36	47	50	58	95
% de bonnes réponses	94.53	80.47	72.66	62.5	46.09	50.78	34.77

Conclusion

- On est passé de 0.78% de chance de trouver le byte à **4,39%**

Recherche du meilleur modèle
Modèle 2 - SVM/Nosel

SVM/Nosel	1	2	3	4	5	6	7
TP	125	116	107	93	79	70	56
TN	122	115	105	94	85	67	54
FN	3	12	21	35	49	58	72
FP	6	13	23	34	43	61	74
% de bonnes réponses	96.48	90.23	82.81	73.05	64.06	53.52	42.97

Conclusion

- On est passé de 0.78% de chance de trouver le byte à **9,03%**

Recherche du meilleur modèle
Modèle 3 - RF

But du modèle i

- couper l'ensemble des traces en deux ensembles
  - Ceux ayant le bit i à 0
  - Ceux ayant le bit i à un

Utilisation d'une forêt d'arbres de décision

Recherche du meilleur modèle
Modèle 3 – RF/Rank

RF/Rank	1	2	3	4	5	6	7
TP	126	106	104	99	77	74	45
TN	124	109	105	99	80	73	44
FN	2	22	24	29	51	54	83
FP	4	19	23	29	48	55	84
% de bonnes réponses	97.66	83.98	81.64	77.34	61.33	57.42	34.77

Conclusion

- On est passé de 0.78% de chance de trouver le byte à **9,12%**

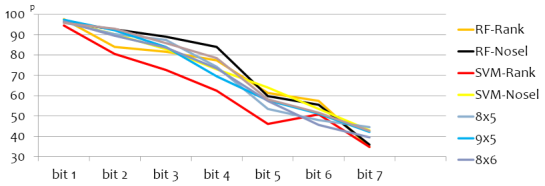
Recherche du meilleur modèle
Modèle 3 – RF/Nosel

RF/Nosel	1	2	3	4	5	6	7
TP	124	115	119	114	79	69	51
TN	122	122	109	101	74	73	41
FN	4	13	19	14	49	59	77
FP	6	6	9	27	54	55	87
% de bonnes réponses	96.09	92.58	89.06	83.98	59.77	55.47	35.94

Conclusion

- On est passé de 0.78% de chance de trouver le byte à **11,03%**
- ➡ 1 fois sur 10, on trouvera le byte

Recherche du meilleur modèle
Conclusion



Conclusion

- RF/Nosel est le meilleur modèle

Recherche du meilleur feature selection

But

- Diminution du temps de calcul total puisqu'on réduit la taille du problème.
- Augmentation de la précision de la prédiction
  - Réduction de la taille du problème amène à une diminution du bruit causé par un surplus d'informations.

Recherche du meilleur feature selection - RF/SOM

RF/SOM

RF/SOM	1	2	3	4	5	6	7
TP	127	109	109	102	81	53	62
TN	120	119	103	93	75	62	55
FN	8	9	25	34	53	66	73
FP	1	19	19	26	47	75	66
% de bonnes réponses	96.48	89.06	82.81	76.17	60.94	44.92	45.70

Conclusion

- On est passé de 0.78% de chance de trouver le byte à 8,26%

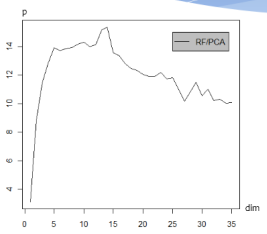
Recherche du meilleur feature selection - RF/SOM

Problème

- Augmentation drastique du temps d'apprentissage
- Dégradation des résultats par rapport à RF/Nosel

Recherche du meilleur feature selection – RF/PCA

RF/PCA



Recherche du meilleur feature selection – RF/PCA

Conclusion

- On est passé de 0.78% de chance de trouver le byte à 15,33% (avec seulement 14 dimensions !)

RF/PCA (14dimensions)	1	2	3	4	5	6	7
TP	123	116	118	113	93	74	56
TN	123	121	114	106	100	77	53
FN	5	7	14	22	28	51	75
FP	5	12	10	15	35	54	72
% de bonnes réponses	96.09	92.58	90.63	85.55	75.39	58.98	42.58

- Avantage
  - Réduction drastique du temps d'apprentissage (5 heures pour valider)
  - Amélioration des résultats par rapport à RF/Nosel
- Meilleur modèle pour l'attaque

Attaque des autres bytes

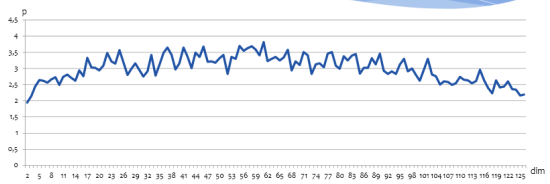
1ère clé

2ème clé

3ème clé

1ère clé

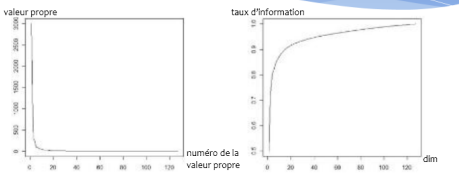
1er byte de la 1ère clé



61 Dimensions	1	2	3	4	5	6	7
TP	56	57	51	47	44	31	29
TN	44	27	48	30	33	37	30
FN	20	37	16	34	31	27	34
FP	8	7	13	17	20	33	35
% de bonnes réponses	78,13	65,63	77,34	60,16	60,16	53,13	46,09

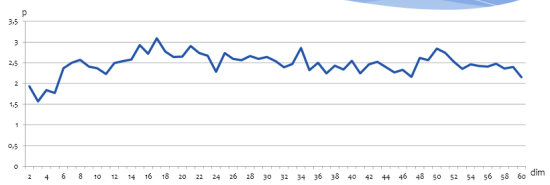
3,81%

1er byte de la 1ère clé



61 Dimensions	1	2	3	4	5	6	7
TP	56	57	51	47	44	31	29
TN	44	27	48	30	33	37	30
FN	20	37	16	34	31	27	34
FP	8	7	13	17	20	33	35
% de bonnes réponses	78,13	65,63	77,34	60,16	60,16	53,13	46,09

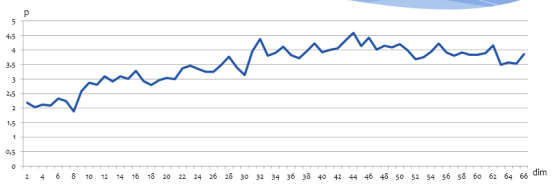
2ème byte de la 1ère clé



17 Dimensions	1	2	3	4	5	6	7
TP	55	52	45	23	38	24	27
TN	54	44	42	30	35	28	29
FN	10	20	22	34	29	36	35
FP	9	12	19	41	26	40	37
% de bonnes réponses	85,16	75,00	67,97	41,41	57,03	40,63	43,75

3,09%

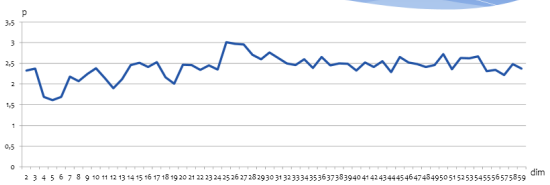
3ème byte de la 1ère clé



44 Dimensions	1	2	3	4	5	6	7
TP	59	56	43	42	45	27	36
TN	42	31	47	47	42	35	30
FN	22	33	17	17	22	29	34
FP	5	8	21	22	19	37	28
% de bonnes réponses	78,91	67,97	70,31	69,53	67,97	48,44	51,56

4,59%

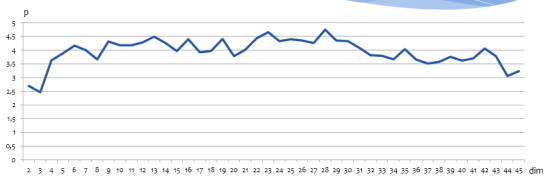
4ème byte de la 1ère clé



25 Dimensions	1	2	3	4	5	6	7
TP	61	51	45	46	30	29	30
TN	48	43	33	28	29	31	40
FN	16	21	31	36	35	33	24
FP	3	13	19	18	34	35	34
% de bonnes réponses	85,16	73,44	60,94	57,81	46,09	46,88	54,69

3,01%

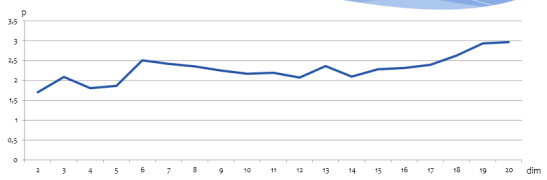
5ème byte de la 1ère clé



28 Dimensions	1	2	3	4	5	6	7
TP	59	51	44	37	39	27	15
TN	56	50	40	40	44	40	23
FN	8	14	24	24	20	24	41
FP	5	13	20	27	25	37	49
% de bonnes réponses	89,84	78,91	65,63	60,16	64,84	52,34	29,69

4,75%

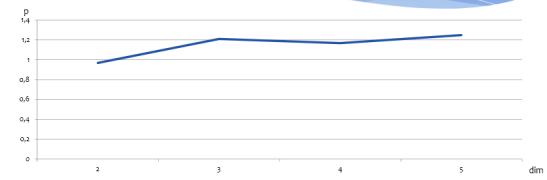
6ème byte de la 1ère clé



29 Dimensions	1	2	3	4	5	6	7
TP	54	51	35	32	36	35	34
TN	42	43	39	39	36	34	37
FN	22	21	25	25	28	30	27
FP	10	13	29	32	28	29	30
% de bonnes réponses	75,00	73,44	57,81	55,47	56,25	53,91	55,47

2,97%

7ème byte de la 1ère clé



5 Dimensions	1	2	3	4	5	6	7
TP	45	39	26	35	26	29	29
TN	37	41	27	28	38	29	34
FN	27	23	37	36	26	35	30
FP	19	25	38	29	38	35	35
% de bonnes réponses	64,06	62,50	41,41	49,22	50,00	45,31	49,22

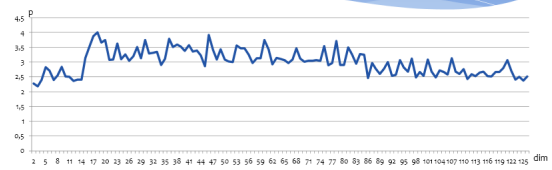
1,25%

2ème clé

1er byte de la 2ème clé

5ème byte de la 2ème clé

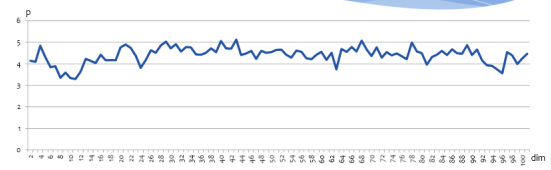
1er byte de la 2ème clé



62 Dimensions	1	2	3	4	5	6	7
TP	60	46	42	42	38	38	32
TN	62	40	48	34	31	33	30
FN	2	24	16	30	33	31	34
FP	4	18	22	22	26	26	32
% de bonnes réponses	95,31	67,19	70,31	59,38	53,91	55,47	48,44

4,00%

5ème byte de la 2ème clé



41 Dimensions	1	2	3	4	5	6	7
TP	58	47	42	37	37	38	33
TN	60	59	45	44	24	42	26
FN	4	5	19	20	40	22	38
FP	6	17	22	27	27	26	31
% de bonnes réponses	92,19	82,81	67,97	63,28	47,66	62,50	46,09

5,13%

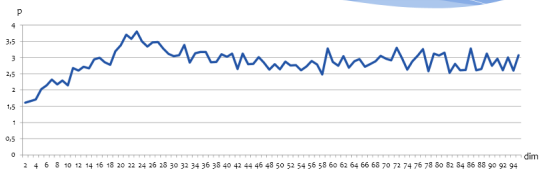
3ème clé

1er byte de la 3ème clé

↓

8ème byte de la 3ème clé

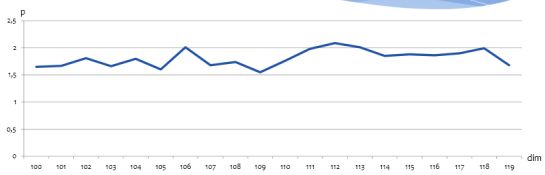
1er byte de la 3ème clé



23 Dimensions	1	2	3	4	5	6	7
TP	60	52	39	37	38	23	34
TN	55	43	41	33	40	33	36
FN	9	21	23	31	24	31	28
FP	4	12	25	27	26	41	30
% de bonnes réponses	89,84	74,22	62,50	54,69	60,94	43,75	54,69

3,8%

8ème byte de la 3ème clé



112 Dimensions	1	2	3	4	5	6	7
TP	45	35	30	34	26	25	37
TN	56	50	38	43	28	28	14
FN	8	14	26	21	36	36	50
FP	19	29	34	30	38	39	27
% de bonnes réponses	78,91	66,41	53,13	60,16	42,19	41,41	39,84

2,09%

Stratégie d'attaque

- Achat du matériel de chiffrement
- Phase d'apprentissage des 7*8*3 modèles (=7bits*8bytes * 3 clés)
- Phase d'attaque d'une clé sur un autre matériel de chiffrement
- Tant qu'on n'a pas trouvé la bonne clé
  - Permuter la valeur des bits les plus durs à prédire (les derniers de chaque byte)

Stratégie d'attaque

- Exemple
  - Clé prédite : 0011 1101
  - Clé possible
    - 0011 1101
    - 0011 1100
    - 0011 1111
    - 0011 1110
    - 0011 1001
    - 0011 1000
    - ...

Brute force

Découverte

Informations en dehors du temps de chiffrement ?

Hypothèse à vérifier

L'information utile pour le décryptage uniquement pendant la période de chiffrement ?

- Vérifions avec MRMR !

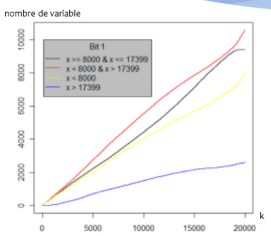
MRMR = Minimum Redundancy Maximum Relevancy

Hypothèse à vérifier

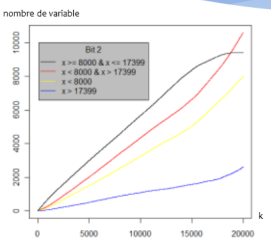
MRMR

- Renvoie les k variables qui ont :
  - le minimum de corrélation entres elles
    -> moins de redondance.
  - le maximum d'informations mutuelles avec le bit à prédire

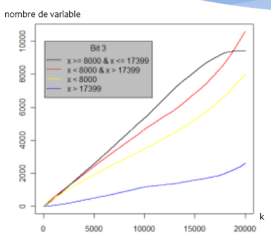
Hypothèse à vérifier



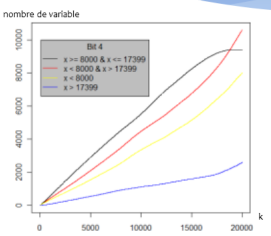
Hypothèse à vérifier



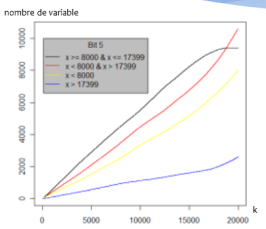
Hypothèse à vérifier



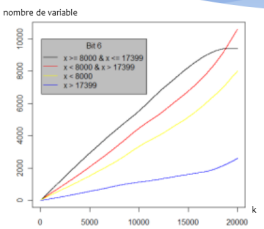
Hypothèse à vérifier



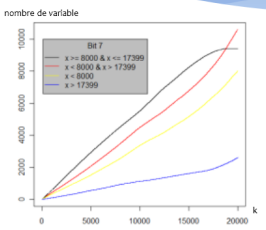
Hypothèse à vérifier



Hypothèse à vérifier



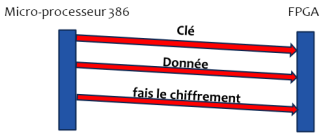
Hypothèse à vérifier



Hypothèse à vérifier

Conclusion

- Information intéressante en dehors de la période de chiffrement.
- Pourquoi ?

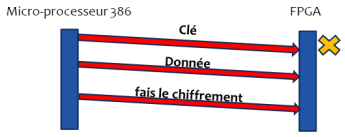


FPGA = Field-Programmable Gate Array

Hypothèse à vérifier

Conclusion

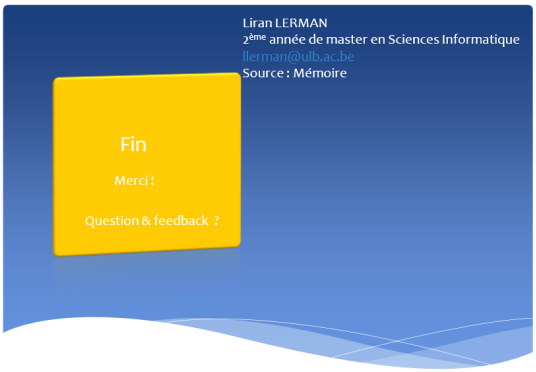
- Information intéressante en dehors de la période de chiffrement.
- Pourquoi ?



FPGA = Field-Programmable Gate Array

Futur

- Comparer notre modèle au Template Based DPA
- Regarder si la fréquence d'utilisation du bit influence la précision de sa prédiction (comparaison de deux distributions)
- Travailler avec d'autres modèles
- Travailler avec d'autres feature selection
- Travailler sur un autre algorithme de chiffrement (RSA, RC4,…)
- Travailler sur un autre matériel de chiffrement
- Travailler avec le temps de chiffrement, le son du matériel de chiffrement, le message chiffré, la météo (on peut tout mettre !),… (Multi-Channel Attacks)
- Chercher le meilleur nombre de traces à prendre
- Chercher le meilleur nombre de points à prendre
- Voir si il y a des protections contre l'attaque
- …


Liran LERMAN
2ème année de master en Sciences Informatique
llerman@ulb.ac.be
Source : Mémoire
Fin
Merci !
Question & feedback ?

Annexe 3
Le rapport de stage est disponible dès la page suivante.

Rapport de stage

16 août 2010

Résumé

Depuis l'Antiquité, l'homme a cherché des techniques pour cacher des informations. Pour cela, des méthodes de cryptographie ont vu le jour. Pour les tester, une école a vu le jour : la cryptanalyse. Celle-ci attaque les systèmes de cryptographie dans le but de retrouver la clé de chiffrement, le message échangé,...

Ce rapport de stage est un résumé d'une mise en application d'un mélange prometteur, « la cryptanalyse et l'apprentissage automatique », sur un matériel de chiffrement dans le milieu bancaire. Ceci a permis de voir que nous n'avions nul besoin de connaitre l'algorithme utilisé ni le moment où nous chiffrons réellement les données pour attaquer le matériel.

Introduction

Suite aux différentes rencontres réalisées chez Atos Worldline, il a été décidé que nous allions continuer la cryptanalyse effectuée par Stéphane Fernandes Medeiros[21] [1] l'an dernier sur un matériel de chiffrement. Toutefois, contrairement au stage qu'il avait effectué, nous avons utilisé l'apprentissage automatique pour déduire la clé de chiffrement. Ce stage a duré 6 mois, du 1^{er} octobre 2009 jusqu'au 30 avril 2010 avec un mois de congé où j'ai eu ma période d'examens en janvier.

Ce travail s'organisera ainsi : nous commencerons par un survol du contexte de travail. Par la suite, nous rechercherons le meilleur modèle d'apprentissage obtenu avec les données de [21], et ce, avec et sans sélection de variables. Nous observerons qu'un problème persistait sur certains bits où le choix du bagging fut utilisé pour essayer de le résoudre. De plus, une hypothèse réalisée par [21] fut vérifiée. Pour aller plus loin, nous avons récolté, avec ou sans cage de Faraday, et testé de nouvelles données avec le modèle choisi dans le but de trouver quels sont les bits de la clé les plus complexes à prédire. De plus, nous avons essayé de mélanger la technique de la densité spectrale avec l'apprentissage automatique pour améliorer les résultats. Nous finirons, enfin, par la conclusion.

Contexte

Le contexte du travail se trouve dans le chapitre 5, section 1 du mémoire.

Recherche du modèle optimal

Nous avons d'abord cherché le meilleur modèle. La recherche du meilleur modèle est visible dans le chapitre 5, section 5 du mémoire.

Lors de la mise en place du premier modèle, nous avions vu de très mauvaises prédictions pour les derniers bits. Vous trouverez ci-dessous l'ensemble des tests qui ont été réalisés pour comprendre ce phénomène.

1. La première idée fut de vérifier si le dernier bit est effectivement générer aléatoirement. Pour cela, nous avons lié des valeurs de bits aléatoires à l'ensemble des traces et ce uniquement pour les $7^{ème}$ et $8^{ème}$ bits. Ainsi, nous sommes certains qu'il n'est pas possible de pouvoir prédire ces bits.
   Ce test fut réalisé dans le but de prouver que le pourcentage de bonnes prédictions que nous obtiendrions pour les données de validation, se rapprocherait de celui obtenu précédemment. Les résultats sont de 49.22 % pour les deux derniers bits. Ainsi, le premier test nous fait penser que nous avons suffisamment de données. En effet, puisque dans le cas où le dernier bit est réellement aléatoire, comme c'est le cas dans

1. La bibliographie se trouve dans le mémoire.

ce test, nous devrions voir des résultats proches de ceux obtenus pour le $7^{ème}$ et le $8^{ème}$ bit pour notre premier modèle, ce qui n'est pas le cas.

2. La deuxième idée a été de regarder le pourcentage de bonnes réponses parmi l'ensemble des données d'apprentissage, ce que nous appelons l'erreur empirique. Nous avons obtenu 59.53 % et 57.88 % de bonnes réponses pour les données d'apprentissage pour les $7^{ème}$ et $8^{ème}$ bits. En voyant le taux d'erreur pour les données d'apprentissage, nous pouvons comprendre la raison du taux d'échec pour les données de validation. En effet, avec cette grandeur aux alentours de 50%, le taux de prédiction ne peut pas être bon.
3. La dernière idée fut de générer des valeurs de bits aléatoires pour l'ensemble des traces, pareillement au premier point mais en regardant l'ensemble d'entrainement. Ainsi, le but était de prouver que le pourcentage des bonnes prédictions que nous obtiendrions, pour les données d'entrainement, se rapprocherait de celui obtenu dans le point deux. Les résultats sont de 60,71 % et de 61.86 % de bonnes réponses ce qui nous conforte dans l'idée initiale du manque de données.

La conclusion de ces tests, détaillés dans la section 5.5 du mémoire, est que nous n'avons pas suffisemment de données pour prédire les derniers bits. De plus, nous devrions ramener le taux de bonnes réponses à 50 % en augmentant le nombre de données collectées.

En ce qui concerne le meilleur modèle, c'était le random forest (RF) qui est sorti vainqueur.

Attaque du $6^{ème}$ bit

Un problème -la faible précision de la prédiction de certains bits- semblait persister dans tous les modèles. C'est pour cette raison qu'il a été décidé d'implémenter une méthode pour essayer d'améliorer la prédiction d'un des bits complexes à prédire, à savoir le $6^{ème}$ bit. Pour cela, nous avons choisi le *bagging* (ou *Bootstrap aggregating*) avec des modèles SOM. La raison de ce choix a été motivée par le fait que le *bagging* améliore les performances de structures instables. Les SOM sont similaires aux réseaux neuronaux et puisque ceux-ci sont connus comme étant instables [8], le *bagging* semble être applicable aux SOM. Les détails d'implémentation d'une telle technique sont décrits dans mon mémoire au chapitre 4, section 6, sous-section 2. Après l'apprentissage, nous donnons à tous les modèles, une trace inconnue de la phase d'apprentissage, et nous réalisons un vote entre tous pour connaître le bit lié à la trace, et nous le réitérons 256 fois. Ainsi, nous avons réalisé un *leave-one-out*.

La figure A3.1 montre les résultats obtenus [2].

2. Les résultats peuvent être refaits en exécutant le fichier code5.R se trouvant dans le DVD remis avec le mémoire.

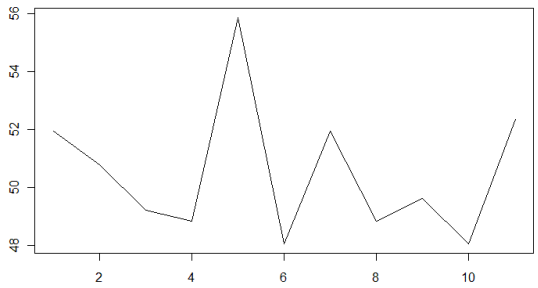


Figure A3.1 : En abscisse se trouve le nombre de SOM dans le modèle bagging et l'ordonnée représente le pourcentage de bonnes réponses.

Nous pouvons remarquer qu'il n'y a pas une forte amélioration de la prédiction lorsque nous augmentons le nombre de modèles. Nous pourrions penser qu'il suffirait alors d'augmenter encore plus le nombre de modèles pour, peut-être, voir une amélioration. Néanmoins l'élévation du nombre de modèles augmente drastiquement le temps d'apprentissage ce qui en fait un modèle non applicable en pratique.

A ce stade du stage, une hypothèse restait à confirmer.

Etude des bornes de la trace

Une hypothèse devait être vérifiée. Elle concerne l'utilité de ne pas prendre en compte les données qui se trouvent hors de la période de chiffrement. Elle est visible dans le chapitre 5, section 8. Nous y remarquons que ces données sont intéressantes pour prédire la clé de chiffrement.

Recherche de la sélection de variables optimales

Après la vérification de l'hypothèse, nous avons cherché le meilleur modèle utilisant une technique de sélection de variables. L'ensemble est visible dans le chapitre 5, section 6. Nous y constatons que le meilleur modèle est le random forest combiné avec le PCA.

Nouvelles traces

L'étape suivante fut d'obtenir de nouvelles données sur le matériel dans le but de s'attaquer aux autres bytes de la clé. Néanmoins, sachant que l'obtention de telles données est longue, la première idée était de voir si nous pouvions n'en récolter qu'une petite partie pour chaque trace. Ainsi, au lieu de prendre 20000 points, nous n'en récolterions que 752 en réduisant la fréquence d'acquisition de l'oscilloscope. Ainsi, un ordinateur de production

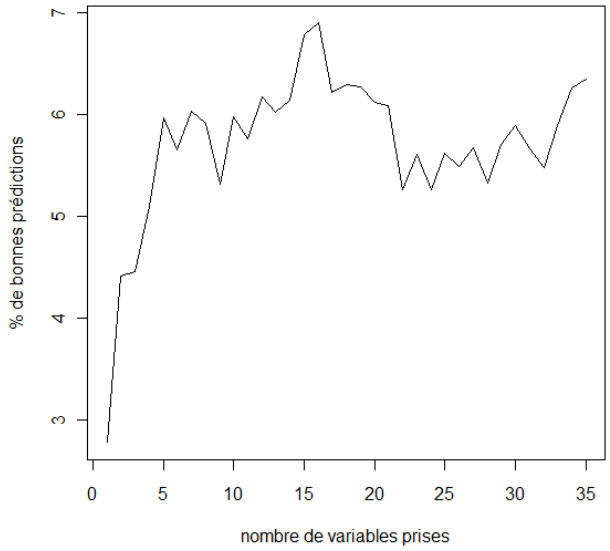


Figure A3.2 : Résultat du modèle RF/PCA avec 752 instants par trace. Nous y voyons apparaître le pourcentage de bonnes prédictions en fonction du nombre de variables choisies.

envoyait 400 fois des demandes de chiffrement d'une donnée constante en utilisant la même clé.

Ceci fut réalisé pour tout un byte de la clé, soit 256 fois. Parallèlement à ces envois de messages, un oscilloscope mesurait la consommation d'énergie du FPGA. Après cela, un prétraitement fut réalisé pour n'avoir que les données nettes à traiter par le modèle de prédiction.

La période de chiffrement fut visible grâce aux triggers mis en place pour le stage de [21], ce qui nous a permis de constater que le FPGA chiffrait entre les points 79 et 647 parmi les 752 points récoltés. Ayant déjà constaté une bonne prédiction avec le modèle RF/PCA, ce dernier fut utilisé avec la technique du *leave-one-out*. Les résultats peuvent être vus en Annexe B. La figure A3.2 résume les résultats pour les pourcentages de bonnes réponses.

En conclusion pour la première prise de données, nous constatons une très forte diminution du pourcentage de bonnes réponses. Ceci viendrait du fait que nous prenons moins de données pour représenter une trace et aussi qu'aucun blindage n'a été mis en place pour réduire le bruit causé par les matériaux électriques se trouvant aux alentours. Ainsi, pour attaquer les autres bytes de la clé, il était important de protéger le matériel de chiffrement avec une cage de Faraday et d'essayer de prendre plus de données pour chaque trace.

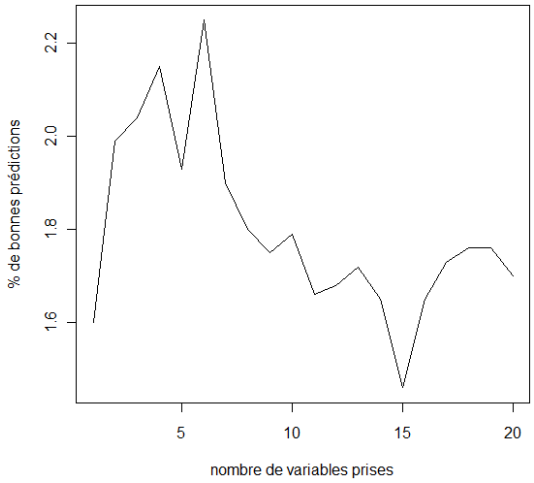


Figure A3.3 : Résultat du modèle RF/PCA sur des données où la cage de Faraday a été mise. Nous y voyons apparaître le pourcentage de bonnes prédictions en fonction du nombre de variables choisies.

Mise en place de la cage de Faraday

Pour vérifier notre protection contre le bruit, nous avons testé notre attaque sur un FPGA protégé par une cage de Faraday [3]. Nous avons pris de nouvelles traces qui ont été ensuite apprises par notre modèle RF/PCA. Comme précédemment, l'ensemble a été validé par la technique du *leave-one-out*. L'ensemble des tableaux représentant les résultats détaillés se trouve en annexe C. La figure A3.3 les résume.

Nous ne remarquons pas une forte amélioration des performances avec l'ajout d'une cage de Faraday. Toutefois, cette dernière a été utilisée pour s'attaquer à l'ensemble des bytes de la clé. Mais avant cela, une autre idée nous est venue.

Densité spectrale

Une autre manière de voir les traces est de regarder leur densité spectrale. Elle regroupe l'ensemble des quantités de chaque fréquence présente dans une trace. Une manière simple de se la représenter est de penser à l'exemple de la lumière du soleil, aussi appelée la lumière blanche. Cette dernière est composée d'une infinité de couleurs représentées chacune par une sinusoïde. En effet, il suffit de prendre un prisme triangulaire pour décomposer la lumière et ainsi voir les différentes couleurs qui la composent.

Cette expérience est exactement ce que nous comptons faire avec la densité spectrale. Cette densité va décomposer chaque trace pour permettre de voir de quoi elle se compose.

3. Une manière élégante de vérifier que la cage de Faraday est un isolant contre le bruit est de mettre un portable à l'intérieur de la boite de Faraday et de regarder si celui-ci capte une antenne portable.

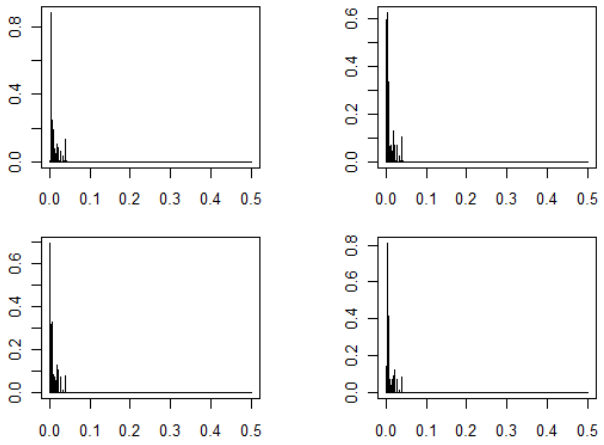


Figure A3.4 : Illustration d'une densité spectrale appliquée sur quatre traces différentes où chacune d'entre elles représente la consommation d'énergie du matériel de chiffrement sur des clés différentes. Nous y voyons en abscisse les fréquences et en ordonnée la quantité de chaque fréquence dans la trace.

La figure A3.4 illustre une densité spectrale [4] appliquée sur quatre traces [5] différentes où chacune d'entre elles représente la consommation d'énergie du matériel de chiffrement sur des clés différentes. Pour chaque graphique, aussi appelé périodigramme, nous y voyons en abscisse les fréquences et en ordonnée la quantité de chaque fréquence dans la trace.

Une remarque intéressante est que nous avons réussi à trouver, avec la densité spectrale, l'hyper-période de chaque trace. Pour cela, nous regardons pour chaque trace la fréquence F la plus significative. Ceci se fait en cherchant le maximum dans le graphique de la densité spectrale. Par la suite, pour obtenir l'hyper-période P, il suffit de prendre l'inverse de la fréquence : $P = \frac{1}{F}$. Cela nous a permis de trouver que l'hyper-période est d'environ 153.6 pour toutes les traces.

Ainsi, approximativement, et comme le montre la figure A3.5, aux environs de tous les 154 points parmi les 4500, un pattern se répète durant la phase de chiffrement. Ceci pourrait s'expliquer par le fait que l'algorithme 3DES est un algorithme répétant 16×3 fois une même étape. Un fait intriguant est que nous avons trouvé une répétition du pattern de 29 fois. Or, sachant que l'algorithme réalise 16×3 fois une même étape, il devrait y avoir 48 répétitions du pattern. Cette intrigue reste inexpliquée.

La densité spectrale fut utilisée comme entrée à notre modèle d'apprentissage. Nous avons réalisé deux types de modèles, l'un en utilisant RF et l'autre RF/PCA. Tous les modèles travaillaient avec la base de données établie par [21].

Pour le premier type de modèle, à savoir RF, nous avons comparé les résultats en

4. La densité spectrale fut construit grâce à la méthode spectrum() de la librairie standard.

5. Ces traces sont des moyennes de consommations d'énergie qui viennent de l'ensemble de données collectées par [21].

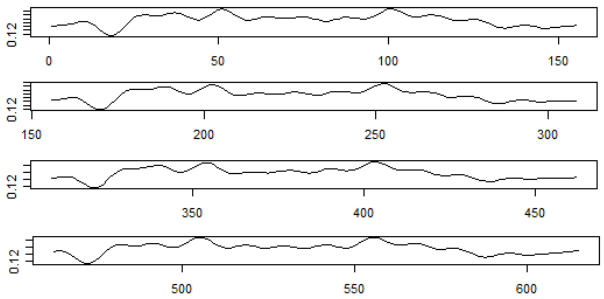


Figure A3.5 : Cette figure montre quatre instants différents d'une même trace dans la partie où le FPGA chiffre.

prenant et en ne prenant pas en compte que le chiffrement se trouvait entre deux bornes (respectivement tableau 1 et tableau 2) [6].

Tableau 1 :

RF	1	2	3	4	5	6	7
TP	126	106	102	85	64	52	57
TN	120	121	113	82	57	55	65
FN	8	7	15	46	71	73	63
FP	2	22	26	43	64	76	71
% de bonnes réponses	96.09	88.67	83.98	65.23	47.27	41.80	47.66

En conclusion avec ce modèle, nous pouvons dire qu'on a 5.83% de chances d'avoir l'ensemble du byte.

Tableau 2 :

RF	1	2	3	4	5	6	7
TP	125	115	115	104	68	52	45
TN	122	122	109	108	78	52	38
FN	6	6	19	20	50	76	90
FP	3	13	13	24	60	76	83
% de bonnes réponses	96.48	92.58	87.50	82.81	57.03	40.63	32.42

En conclusion avec ce modèle, nous pouvons dire qu'on a 9.23% de chances d'avoir l'ensemble du byte.

Ces deux modèles montrent qu'il est intéressant de prendre en compte les valeurs en dehors des bornes de chiffrement.

Pour le deuxième type de modèle, à savoir RF/PCA, nous avons, à nouveau, comparé les résultats en prenant et en ne prenant pas en compte que le chiffrement se trouvait

6. Les données peuvent être reproduites grâce au fichier code15.R.

entre deux bornes (respectivement tableau 3 et figure A3.6) [7]. Néanmoins, lorsque nous prenons uniquement les points se trouvant entre les deux bornes de chiffrement, nous avons constaté que la quantité d'informations initiales gardées n'est pas suffisante tant que nous utilisons le PCA. Ainsi, il n'est pas nécessaire d'utiliser PCA pour ce modèle.

Tableau 3 :

RF	1	2	3	4	5	6	7
TP	125	68	69	84	68	90	79
TN	116	116	101	56	67	51	34
FN	12	12	27	72	61	77	94
FP	3	60	59	44	60	38	49
% de bonnes réponses	94.14	71.88	66.41	54.69	52.73	55.08	44.14

En conclusion avec ce modèle, nous pouvons dire que nous avons 3.15% de chances d'avoir l'ensemble du byte.

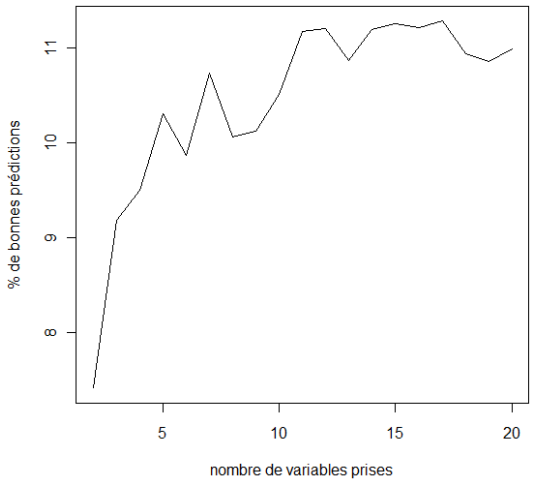


Figure A3.6 : Résultat du modèle RF/PCA basé sur la densité spectrale en ne prenant pas en compte que le chiffrement se faisait entre deux bornes. Nous y voyons apparaître le pourcentage de bonnes prédictions en fonction du nombre de variables choisies.

La figure A.3.6 montre qu'on pouvait s'arrêter à une dimension 17 pour avoir 11.28% de chances d'avoir l'ensemble du byte.

Ces deux modèles montrent qu'il est intéressant de prendre en compte les valeurs en dehors des bornes de chiffrement.

Finalement, nous ne constatons pas une amélioration des résultats en utilisant la densité spectrale. C'est pourquoi la suite de cet article ne se focalisera plus sur la densité spectrale. Toutefois, pour être certain que les modèle RF et RF/PCA sont ceux qu'il

7. Les données peuvent être reproduites grâce au fichier code14.R.

fallait utiliser, nous aurions dû les comparer à d'autres modèles tels que SOM, SVM,... Néanmoins, à cause d'un manque de temps, cette étape ne fut pas réalisée.

Attaque de tous les bytes

L'attaque sur les autres bytes de la clé est visible au chapitre 5, section 7 du mémoire. Cette partie a pris 80 % de l'ensemble du temps du stage. Nous y constatons que certains bits sont plus prévisibles que d'autres.

Conclusion

Ce stage m'a permis d'avoir une première approche pratique du mélange entre la cryptanalyse et l'apprentissage automatique et de voir les limites de l'informatique au niveau de la puissance de l'ordinateur. En effet, la plupart des algorithmes nécessitaient plusieurs jours d'exécution. De plus, au début du stage, la prise de donnée était particulièrement lente et fortement manuelle dû essentiellement à des contraintes de reconfiguration de manière périodique. Et bien même lorsque nous avons réussi à automatiser ce mécanisme, à la moitié du stage, il persistait des phénomènes aléatoires [8] qui ont ralenti cette automatisation. Certains diront que c'est dû à la malchance. Néanmoins, cela restait un des plus forts ralentissements du déroulement du stage.

Pour ce qui est des étapes prises tout au long du stage, il pourrait sembler à première vue que le début du stage était plutôt chaotique voyant les différents chemins pris. Néanmoins, cela est dû essentiellement à un manque de maturité dans le domaine. Cette maturité est venue petit à petit, de jour en jour. De plus, cet aspect chaotique, qui pourrait représenter une mauvaise organisation, a permis de travailler avec des méthodes d'apprentissage automatique totalement différentes et ainsi d'augmenter la connaissance du domaine, sans se focaliser sur une seule solution possible à notre problème initial.

Le stage a aussi permis de se rendre compte qu'un algorithme, qui est fort en théorie, ne l'est pas forcement en pratique. La faille de sécurité réside donc dans la non coopération entre différents domaines tels que l'électronique et la cryptographie. Ainsi, ceux qui construisent un algorithme cryptographique, devraient travailler avec ceux le mettant en pratique.

Notons que l'ensemble de la clé peut être attaqué avec un système distribué où chaque modèle attaque un bit de la clé. De plus, il n'est pas indispensable de connaitre l'algorithme [9] implémenté dans le FPGA ni d'avoir une connaissance approfondie de ce der-

8. Certains phénomènes sont compréhensibles, tels que l'oubli d'une procédure dans l'automatisation de la tache globale, alors que d'autres restent inexpliqués, tels que la réalisation partielle d'une tache qui en théorie devrait être réalisée totalement.

9. L'algorithme dans le FPGA peut implémenter des méthodes de chiffrement, de signature ou tout autre modèle demandant l'utilisation d'une clé.

nier [10] pour réaliser l'attaque [11]. En effet, RF/PCA ne fait que le lien entre la consommation d'énergie et la clé utilisée. Ainsi, ce type d'attaque peut être généralisé sur d'autres algorithmes de chiffrement.

De plus, puisque le modèle peut être résumé en une fonction qui renvoie la clé selon la trace qu'on lui donne en paramètre, on pourrait généraliser ce concept, pour passer à un « *Multi-Channel Attacks* », en mettant en entrée non seulement des traces, mais aussi le temps de chiffrement, le message chiffré, le message claire, le son,... Ainsi, la technique PCA ne ferait que supprimer les dimensions inutiles à l'attaque du matériel.

Finalement, il est nécessaire de préciser que les résultats obtenus ne peuvent être facilement comparés aux autres résultats se trouvant dans la littérature contemporaine. En effet, l'attaque réalisée est dépendante de plusieurs caractéristiques telles que le type de FPGA et le contexte de l'attaque, qui ne nous permet pas de réaliser une comparaison triviale.

10. En utilisant des techniques telles que SPA, on aurait dû approfondir nos connaissances sur le jeu d'instructions du matériel.

11. Ceci va dans l'hypothèse de base, bien connue, de Kerckhoff qui est : seule la clé est le secret réel.