SORBONNE UNIVERSITÉ

Rapport de Projet

Sorbonne Université

❧

Master 1 SFPN : Sécurité, Fiabilité et Performance du Numérique

Unité d'enseignement : Projet SFPN

Estimation de rang de clefs pour les attaques par canaux auxiliaires

---

PAR : **TALEB ABDUL RAHMAN, FERROUKHI DJAMEL**

Encadré Par : PROUFF EMMANUEL

Date : 27 Mai 2019

Table des matières

Chapitre 1

Introduction

1.1 Problématique

De nos jours, la cryptographie est omniprésente en embarquée sur de diverses composantes de sécurité telles que les cartes à puce. Les algorithmes de chiffrement sur ces composantes ont pour but d'assurer une sécurité élevée contre les actions malveillantes. Néanmoins, leur sécurité «mathématique» ne garantit pas forcément une sécurité lors de l'utilisation en «pratique». En effet, ces algorithmes sont souvent vulnérables à des attaques qui interceptent et analysent des mesures particulières, comme des mesures de consommation électrique, ou des mesures de temps d'exécution. Ces attaques sont appelées des "attaques par canaux auxiliaires (SCA pour Side-Channel Attack en anglais)".
Les attaques par canaux auxiliaires sont des attaques de type "diviser pour régner", qui visent à reconstruire le paramètre secret de l'algorithme (ou la clef) par morceaux. L'exécution de l'attaque aboutit à un ensemble de valeurs possibles pour chaque morceau, associées chacune à une probabilité. Retrouver le paramètre secret revient ainsi à recombiner la suite de morceaux la plus probable parmi les autres, et qui correspond à celle du paramètre secret pour une exécution normale de l'algorithme. Cependant, en fonction de la robustesse de l'implémentation algorithmique, l'attaque peut fournir une mauvaise clef avec la probabilité maximale, alors que la vraie clef secrète peut en avoir une inférieure. Dans ce cas, on commence à parler du "rang de la clef secrète" dans l'ensemble des clefs possibles. Le rang étant le nombre de possibilités avec une probabilité supérieure ou égale à celle de la clef secrète, et le but de ce projet est d'implanter des algorithmes qui "estiment" ce rang.

1.1.1 Défition Mathématique

En notant $\boldsymbol{S}$ la distribution de probabilités des valeurs possibles pour chaque morceau de la clef, et par $\boldsymbol{P}$ celui contenant toutes les distributions $\boldsymbol{S}$ (il y a d distributions $\boldsymbol{S}$, chacune de v valeurs), la probabilité d'une reconstruction possible de clef est notée par $p \in \prod_{\boldsymbol{S}\in\boldsymbol{P}} \boldsymbol{S} = s_1 * ... * s_d$ où $s_i \in \boldsymbol{S}$. Ce calcul vient du fait qu'une clef possible K est représentée par ses sous-clefs $K = k_1||..||k_d$ où $||$ représente une concaténation. Et puisque les morceaux sont supposés être indépendants dans l'attaque, la probabilité de la clef K est le produit des probabilités des sous-clefs.
On note aussi p^* la probabilité de la vraie clef secrète.
Par conséquent, le rang de la clef secrète K^*, notée $rang(K^*, P)$ est définie par :

$$rang(K^*, P) = \sum_{p\in\prod_{\boldsymbol{S}\in\boldsymbol{P}}\boldsymbol{S}} 1[p^* \leq \prod_{k_i\in K} s_i] \tag{1.1}$$

où 1 désigne la fonction de vérité définie comme suit :

$$1[f] = \begin{array}{l} \text{1 si la condition f renvoie VRAI,} \\ \text{0 si la condition f renvoie FAUX} \end{array} \tag{1.2}$$

En effet, pour 2 clefs K_1 de probabilité p_1 et K_2 de probabilité p_2 possibles, on dit que

$$rang(K_1, P) < rang(K_2, P) \text{ ssi } p_1 \geq p_2 \tag{1.3}$$

Intuitivement, la formule du rang signifie qu'on va effectuer le produit cartésien de tous les ensembles S pour obtenir toutes les probabilités des clefs possibles, et ensuite compter le nombre de probabilités qui sont supérieures ou égales à p^*. Dans le contexte de l'AES-128 par exemple, avec 256 possibilités pas sous-clef, et 16 sous-clefs, cela revient à calculer 256^{16} valeurs possibles, ce qui est infaisable. D'où l'intérêt d'avoir des algorithmes non naïfs qui estime le rang de la clef.
Une estimation dans ce cas sera de calculer un intervalle $[I_1, I_2]$ qui contient le rang de la clef. L'efficacité et la précision de l'algorithme dépendent ainsi de la taille de cet intervalle estimé.

1.2 Contexte Initial

De ce qui a été dit, il fallait résoudre le problème d'estimation du rang de la clef et c'est là où intervient l'article de Bernstein [1], fournit par notre encadrant Prof. PROUFF Emmanuel.
Les auteurs citent dans cet article plusieurs algorithmes d'estimation de rang de la clef et propose-en un nouveau. Le but de ce projet était d'implémenter les algorithmes se trouvant

dans l'article et de comparer leurs efficacités. Prof. PROUFF nous a aussi fournit une base de données qui sera détaillée dans la suite et qui contient des traces de mesure de consommation électromagnétique, pendant l'exécution de l'algorithme de chiffrement AES-128 masquée (où la clef est décomposée en 16 morceaux), permettant ainsi d'effectuer des attaques par canaux auxiliaires dessus, et tester les algorithmes. On a choisi le langage de programmation C pour tous nos algorithmes et nos implémentations, puisque la vitesse du calcul est l'un des buts essentiels des procédures étudiées, et puisqu'en tant que binôme qui effectue ce projet ensemble, on est le plus à l'aise avec ce langage, ce qui nous a facilité la répartition des tâches ainsi que l'implémentation des algorithmes.

1.3 La Base "ASCAD"

Pour mettre en place l'attaque, il nous fallait obtenir des traces correspondantes à l'écoute des signaux émises lors d'un algorithme de chiffrement. Pour cela notre encadrant de projet nous a introduit la base "ASCAD ANSSI SCA Databases" [5] qui, à la base, est utilisé pour des études d'apprentissage profond dans le contexte des attaques SCA, mais qui contient les informations nécessaires pour effectuer d'autres traitements dessus. Cette base consiste en 60000 exécutions différentes de l'algorithme du "AES-128" masqué utilisant la même clef secrète, composées chacune de différents éléments :

1. Le message en clair $m = (m_1, .., m_{16})$
2. Le message chiffré $c = (c_1, .., c_{16})$
3. Les masques associés au chiffrement $(r_1, .., r_{16}, r_{in}, r_{out})$
4. La clef secrète $k^* = (k_1, .., k_{16})$, qui est la même pour toutes les exécutions
5. Une trace de mesure électromagnétique, prise durant 100000 instants de l'algorithme

L'algorithme de chiffrement utilisé pour cette base est le "AES-128 masqué", une variante du "AES-128" qui ajoutent du bruit (qu'on appelle des "masques") aux entrées et sorties de l'algorithme, pour une protection contre les attaques de type SCA. Néanmoins, nous avons implanté nos attaques tout en utilisant les masques pour adapter l'attaque sur un "AES-128" basique, puisque le but de ce projet est essentiellement d'évaluer la performance des algorithmes d'estimation de rang de clef. Ainsi, nous avons utilisé les masques pour obtenir des scores représentatifs d'une attaque de type SCA qui fonctionne correctement et retourne la clef secrète.

1.4 CPA : Correlation Power Analysis

A partir de cette base, on a pu exécuter une attaque SCA nommée "CPA : Correlation Power Analysis" [4] sur les traces fournies, et obtenir des "scores" associés aux différentes clefs

possibles.

En effet, la CPA désigne une attaque qui utilise les mesures de consommation émises lors d'un chiffrement (dans notre cas l'AES-128), afin de corréler via le coefficient de correlation de "Pearson", ces consommations d'énergie à toutes les valeurs des sous-clefs possibles. Précisément, le moment où l'attaque fournit des résultats corrects est à la première itération de l'AES-128, pendant le calcul des résultats des SBOX de l'algorithme, puisqu'à cette itération, chacune des sous-clefs peut toujours être attaquée indépendamment. Pour cela, on construit un modèle de consommation basé sur le poids de hamming de la sortie des SBOX. Ce modèle de consommation, pour la bonne clef considérée à l'entrée, doit fortement corréler avec les mesures de consommation réelles. Un pseudo-algorithme sur la base "ASCAD" est résumé dans l'Algorithme 1.

Pour lancer les algorithmes d'estimation de rang de clef, on ajoute une étape de normalisation

Algorithme 1 : Correlation Power Analysis [4]

Données : L'ensemble des messages en clair. Les mesures de consommation de 100000 instants
Résultat : La clef secrète la plus probable. L'ensemble des scores.
$K \leftarrow []$;
$scores \leftarrow []$;
pour $i = 1..16$ **faire**
  $I \leftarrow []$;
  pour $k = 0..255$ **faire**
    Construire le modèle de consommation (poids de hamming) sur la sortie du SBOX, en considérant la clef k avec l'octet i de chacun des messages en clair (m_i);
    $L \leftarrow []$;
    pour $n = 1..100000$ **faire**
      $L.append$(Coefficient de Pearson entre le modèle de consommation, et la consommation à l'instant n de chacune des traces);
    fin
    $I.append$(Maximum de la liste L);
  fin
  $scores.append(I)$;
  $K.append$(Sous-clef correspondante au maximum de la liste I);
fin
retourner $(K, scores)$;

des scores, afin de s'approcher de la représentation par distributions de probabilités, sachant que cela ne change rien sur les résultats finaux des algorithmes.

Chapitre 2

Algorithmes d'Estimation de Rang de Clef

Dans cette partie, on détaillera les algorithmes présentés dans [1] par *Bernstein* (REA et EREA), ainsi que l'algorithme qu'il a proposé (PRO). Les mesures de performance ainsi que la comparaison seront abordées dans le chapitre 4. On précise que les algorithmes présentés dans ce travail sont des algorithmes de type "boîte blanche", c'est à dire en connaissant la clef secrète et sa probabilité, on veut estimer son rang par rapport aux autres possibles. D'autres algorithmes de type "boîte noire" existent, qui essaient de retrouver la clef ainsi que son rang, sans pour autant la connaître à l'avance. Néanmoins, ces algorithmes sont hors du sujet de notre projet, et ont pour nom "algorithmes d'énumération de clef".

2.1 REA : Rank Estimation Algorithm

L'algorithme du REA a été introduit en premier par les chercheurs Veyrat-Charvillon, Gerard et Standaert dans leur article [2], et a été repris par *Bernstein* dans le sien. L'idée de cet algorithme est inspiré de la représentation graphique de l'espace des clefs possibles. La Figure 2.1 montre un exemple où la clef est composée de deux sous-clefs seulement. Les deux dimensions k_1 et k_2 représentent chacune la liste des probabilités de la sous-clef correspondante, triée dans l'ordre croissant des probabilités. La troisième dimension représente ainsi la probabilité de la clef correspondante $Pr((k_1^{(i)}||k_2^{(j)})) = p_1^{(i)}p_2^{(j)}$ (on rappelle que $||$ est la concaténation). On peut remarquer que l'espace représente une structure de "montagne" où les probabilités sont en ordre croissant du bas vers le haut. Pour la probabilité de la clef secrète K^* représentée par le carré bleu, toutes les clefs avec une probabilité supérieure ou égale (avec un rang inférieur) sont représentées par la couleur verte, et les autres par la couleur rouge. Le rang de la clef est alors le nombre de carrés verts dans l'espace.
Ainsi, l'algorithme REA profite de cette strucutre en montagne de l'espace. En généralisant pour d sous-clefs possibles, et en connaissant la probabilité de clef secrète, un point $(i_1, .., i_d)$ représentant une clef K est choisi. Si $Pr(K) \geq p^*$ (carré vert), alors les points $(i'_1, .., i'_d)$ avec

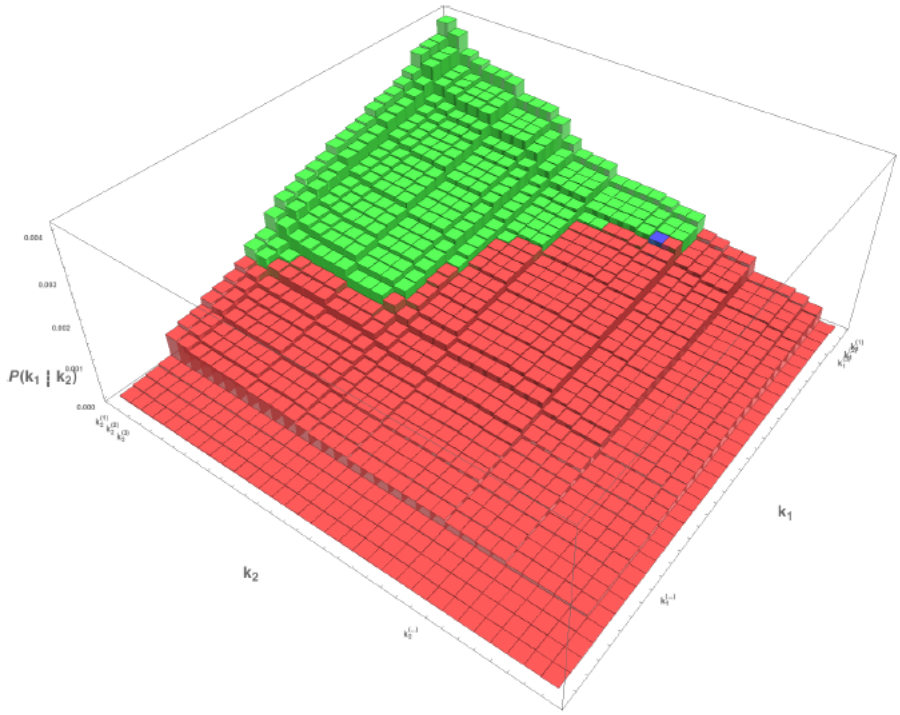

FIGURE 2.1 – Représentation Graphique de l'espace des clefs possibles

$i'_j \leq i_j$ sont tous de rang inférieur à celui de la clef, leur nombre peut être calculé et additionné à la borne inférieure I_1 de l'intervalle du rang $I = [I_1, I_2]$ (On rappelle qu'une probabilité supérieure à celle de la clef secrète, correspond à un rang inférieur à celui de la clef, une clef de rang 0 étant la plus probable parmi les autres, et de rang maximum la moins probable). Sinon, le deuxième cas consiste à soustraire le nombre de points $(i'_1, .., i'_d)$ tel que $i'_j \geq i_j$ de la borne supérieure I_2. L'ensemble des points traités forment un "volume". Chaque volume ou hyperrectangle est ainsi représenté algorithmiquement par ses deux points extrémités, et éventuellement le point de découpage choisit. Un volume est alors soit un hyperrectangle représenté par deux points, soit la différence de deux hyperrectangles représentés par trois points. Le stoquage de ces hyperrectangles devient de plus en plus complexe au cours de l'algorithme. Ainsi, lorsqu'on veut retraiter un volume qui est la différence de deux hyperrectangles, on commence par le couper en deux volumes et réitérer dessus. Cette procédure est répétée itérativement jusqu'à avoir traité tout l'espace (ou jusqu'à atteindre une capacité de calcul maximale). L'algorithme est résumé dans l'Algorithme 2. On rappelle que le nombre de sous-clefs est d, chacune ayant v possibilités.

Dans cet algorithme, on peut considérer L comme la représentation graphique de l'espace des clefs (ou l'ensemble des volumes). L contient au début un unique volume qui est l'espace en entier. Ensuite, la procédure consiste à choisir le volume maximum de cet espace. Si ce volume n'a pas encore été traité (c'est-à-dire n'est pas la différence de deux hyperrectangles), un point

sera choisit par un algorithme de "hill climbing" pour déterminer celui qui maximise la taille de l'hyperrectangle enlevé du volume. Suivant la probabilité de ce point, la taille de l'hyperrectange induit supérieur (resp. inférieur) par le point sera additionnée à la borne inférieure I_1 (resp. soustraite de la borne supérieure I_2). La procédure $Découper\_volume()$ modifie le volume V en une différence de deux hyperrectangles représentée par trois points. Quand ce volume sera choisit de nouveau par l'algorithme, la procédure $Couper()$ sera appelée afin de le couper en deux sous-volumes suivant une seule dimension d dimensions. Le choix de la dimension est suivant l'optimisation du volume restant, puisqu'un découpage naïf suivant les d dimensions résulte en $2^d - 1$ nouveaux volumes, ce qui devient non gérable dans la mémoire après quelques itérations. De plus, le choix du volume de taille maximum à traiter à chaque itération a été prouvé comme efficace empiriquement, malgré qu'il existe d'autres choix possibles. Dans ce cas, puisque le choix du maximum est fait, l'ensemble L des volumes a été implanté comme un tas, où retrouver le maximum a un coût constant $O(1)$, et les modifications de la structure sont en $O(log\ n)$.
Le plus long l'algorithme continue à tourner, le plus petit l'intervalle I devient. Par contre, le temps d'exécution est aussi limité par la capacité mémoire. Les résultats expérimentaux seront présentés dans le Chapitre 4.
Enfin, la phase de pré-traitement correspond à une étape de "fusion des dimensions". En effet, les expériences ont montré de meilleurs résultats lorsque l'on fusionne les d dimensions en un nombre plus petit à traiter. Par exemple pour une attaque sur l'AES-128, une fusion des 16 dimensions de 2^8 valeurs chacune en 8 dimensions de 2^{16} valeurs a donné un intervalle plus précis contenant le rang de la clef secrète. Pareil pour une fusion en des dimensions de 2^{24} valeurs au maximum... Ces résultats seront également présentés dans le Chapitre 4.

2.2 EREA : Extended Rank Estimation Algorithm

L'algorithme EREA est en fait une extension statistique de l'algorithme présenté dans 2.1. L'idée est de faire tourner l'algorithme REA pour un temps t avant de faire un échantillonnage sur l'espace restant des clefs et calculer un intervalle de confiance pour le rang de la clef. En effet, d'après [2], pour estimer $rang(K^*, P)$ à partir de l'espace restant dans l'intervalle retourné par l'algorithme REA de taille M, on tire un ensemble de clefs de cet espace de façon uniforme. Cela correspond à un tirage aléatoire et uniforme de différents rangs et leurs probabilités correspondantes. Donc, si on tire une clef K de probabilité p_K (et de rang noté $rang(K, P)$), on a que $Pr(p_K \geq p^*) = Pr(rang(K, P) \leq rang(K^*, P)) = \frac{rang(K^*,P)}{M}$.
Ainsi, pour estimer $rang(K^*, P)$, on doit donner un intervalle de confiance à la probabilité q qu'on tire une clef de l'espace restant, et que la probabilité de cette clef soit supérieure à p^* (donc de rang inférieur à $rang(K^*, P)$). Cette observation correspond à une distribution

Algorithme 2 : Rank Estimation Algorithm [2]

Données : L'ensemble des d distributions $\boldsymbol{P} = \{\boldsymbol{S_i}\}_{1 \leq i \leq d}$, chacune triée dans l'ordre croissant. La probabilité de la clef secrète p^*.
Résultat : L'intervalle $I = [I_1, I_2]$ contenant $rang(K^*, P)$
$L \leftarrow \{[1; v]^d\}$;
$I \leftarrow \{[1; v^d]\}$;
$Pretraitement(P)$;
tant que $L \neq \emptyset$ **faire**
  $V \leftarrow max_{V \in L}|V|$;
  $L \leftarrow L \setminus V$;
  si $Non\_traité(V)$ **alors**
    $K \leftarrow Choisir\_point(V)$; $Découper\_volume(V, K, p^*, P)$;
    $Mettre\_à\_jour(V)$; $L \leftarrow L \cup V$;
    $Mettre\_à\_jour(I)$;
  sinon
    $\{V_1, V_2\} \leftarrow Couper(V)$;
    $L \leftarrow L \cup \{V_1, V_2\}$;
  fin
fin
retourner I;

binomiale, où q peut être estimé par $\bar{q} = \frac{\sum_{i=1}^{n} X_i}{n}$, où n est le nombre de clefs tirées, et X_i valant 1 lorsqu'on tire une clef K_i telle que $rang(K_i, P) < rang(K^*, P)$ et 0 sinon. $\bar{q}$ a comme valeur espérée q et comme variance $\frac{q(q-1)}{n}$. Par le théorème central limite, on pourra obtenir un intervalle de confiance pour cette probabilité :

$$\left[\bar{q} - z_{1-\frac{1}{2}\alpha}\sqrt{\frac{1}{n}\bar{q}(1-\bar{q})}, \bar{q} + z_{1-\frac{1}{2}\alpha}\sqrt{\frac{1}{n}\bar{q}(1-\bar{q})}\right] \tag{2.1}$$

où $z_{1-\frac{1}{2}\alpha}$ représente la valeur du $1 - \frac{1}{2}\alpha$ quartile de la loi normale. Afin d'assurer une bonne estimation même quand le nombre d'échantillons tirés n'est pas très grand, une optimisation de l'intervalle de confiance proposée dans [3] est utilisé pour le cas de cet algorithme, et consiste en la formule suivante :

$$\begin{aligned}\left[\frac{1}{1+\frac{1}{n}z^2_{1-\frac{1}{2}\alpha}}\left(\bar{q} + \frac{1}{2n}z^2_{1-\frac{1}{2}\alpha} - z_{1-\frac{1}{2}\alpha}\sqrt{\frac{1}{n}\bar{q}(1-\bar{q}) + \frac{1}{4n^2}z^2_{1-\frac{1}{2}\alpha}}\right),\right. \\ \left.\frac{1}{1+\frac{1}{n}z^2_{1-\frac{1}{2}\alpha}}\left(\bar{q} + \frac{1}{2n}z^2_{1-\frac{1}{2}\alpha} + z_{1-\frac{1}{2}\alpha}\sqrt{\frac{1}{n}\bar{q}(1-\bar{q}) + \frac{1}{4n^2}z^2_{1-\frac{1}{2}\alpha}}\right)\right]\end{aligned} \tag{2.2}$$

D'après cela, l'algorithme EREA est résumé dans l'Algorithme 3.

Algorithme 3 : Extended Rank Estimation Algorithm [2]

Données : L'ensemble des d distributions triées dans l'ordre croissant $\boldsymbol{P} = \{\boldsymbol{S_i}\}_{1 \leq i \leq d}$. La probabilité de la clef secrète p^*. Un temps t. Un nombre d'échantillons n. Une valeur de confiance α.

Résultat : L'intervalle $I = [I_1, I_2]$ contenant $rang(K^*, P)$ avec 100% de confiance. L'intervalle $I_\alpha = [I_{\alpha_0}, I_{\alpha_1}]$, contenant $rang(K^*, P)$ avec α.100% de confiance.

$(L, I) \leftarrow REA(P, p^*, t)$;
$total \leftarrow 0$;
pour chaque $V \in L$ **faire**
    $n_V \leftarrow Binomial(n, \frac{|V|}{|L|})$;
    pour $i = 1 .. n_V$ **faire**
        $p \leftarrow$ Probabilité d'un point aléatoire de V;
        si $p \geq p^*$ **alors**
            $total \leftarrow total + 1$;
        fin
    fin
fin
$\bar{q} \leftarrow \frac{total}{n}$;
$Var(\bar{q}) \leftarrow \frac{q(q-1)}{n}$;
retourner I_α , calculé à partir de $I, \bar{q}, Var(\bar{q})$ et α;

2.3 PRO : Polynomial Rank Outlining Algorithm

L'algorithme PRO utilise la notion de "polynôme généralisé". D'après [1], un polynôme généralisé est une fonction qui s'écrit sous la forme $\sum_{r \in \mathbb{R}} F(r)x^r$ où $F : \mathbb{R} \to \mathbb{R}$ avec $F(r) \neq 0$ pour un nombre fini de $r \in \mathbb{R}$. Les opérations d'addition et de soustraction s'effectuent comme sur les polynômes habituels. La multiplication de deux polynômes généralisés F et G s'exprime comme $(FG)(r) = \sum_{s \in \mathbb{R}} F(s)G(r - s)$.
La distribution d'un polynôme généralisé F, noté $distr\ F$ par rapport à $h \in \mathbb{R}$ dénote la somme des coefficients des termes $F(r)x^r$ tel que $r \leq h$. En d'autres termes, $distr\ F\ (h) = \sum_{r \leq h} F(r)$. La distribution vérifie les propriétés suivantes :

$$distr(-F)(h) = -(distr\ F)(h)$$
$$distr(F + G)(h) = distr\ F(h) + distr\ G(h)$$
$$distr(FG)(h) = \sum_{s \in \mathbb{R}} F(s)(distr\ G)(h - s)$$

La distribution forme aussi un ordre partiel sur les polynômes généralisés. En effet, on dit que $F \leq G$ ssi $(distr\ F)(h) \leq (distr\ G)(h)$ pour tout $h \in \mathbb{R}$.
L'algorithme PRO est basé essentiellement sur la distribution de polynôme généralisé. L'idée

c'est de calculer la distribution de deux polynômes G_P et H_P bien choisis par rapport à la probabilité de la clef secrète p^*, afin d'obtenir les deux bornes de l'intervalle $I = [I_1, I_2]$.
Pour cela, on expliquera dans la suite comment obtenir la borne supérieure I_2. La borne inférieure I_1 sera obtenue de la même méthode. Pour cet algorithme, un paramètre α sera introduit et sera nommé le "paramètre de précision". Soit la fonction $\widetilde{.} :]0, 1] \to \mathbb{R}$ telle que :

$$\widetilde{p} = \alpha.log_2\,(1/p)$$

A partir de cette fonction, on peut définir le polynôme F_P :

$$F_P(x) = \prod_{S \in P} \sum_{s \in S} x^{\widetilde{s}}$$

où P est comme avant l'ensemble des différentes distributions de probabilités S. Ce polynôme a un terme par probabilité de clef possible. Il aura en fait $O(v^d)$ termes. Retrouver la borne supérieure revient ainsi à calculer $(distr\ F_P)(p^*)$, ce qui revient à retrouver le nombre de termes qui ont un exposant plus petit ou égale à p^*. Ce calcul revient à un comptage naïf des termes et n'est ainsi pas faisable. Pour cela, on définit la fonction $\underline{.} :]0, 1] \to \mathbb{Z}$ telle que :

$$\underline{p} = \lfloor \alpha.log_2\,(1/p) \rfloor = \lfloor \widetilde{p} \rfloor$$

et on définit le polynôme G_P :

$$G_P(x) = \prod_{S \in P} \sum_{s \in S} x^{\underline{s}}$$

En observant les définitions des fonctions $\widetilde{.}$ et $\underline{.}$, on peut remarquer que pour une probabilité de clef $p = s_1..s_d$, on a :

$$\widetilde{p} = \alpha.log_2\,(1/p) \geq \lfloor \alpha.log_2\,(1/p) \rfloor \geq \sum_{s_i} \lfloor \alpha.log_2\,(1/s_i) \rfloor$$

Donc, dans les polynômes F_P et G_P,

$$\sum_{s \in S} x^{\underline{s}} \geq \sum_{s \in S} x^{\widetilde{s}}$$

Alors,

$$G_P \geq F_P \text{ et } (distr\ G_P)(p^*) \geq (distr\ F_P)(p^*)$$

On a donc créér une borne supérieure sur $(distr\ F)(p^*)$. En effet, le polynôme G_P a moins de termes que le polynôme F_P, et plus le paramètre α augmente, plus on aura de termes distincts dans G_P, et plus la valeur calculée sera précise, d'où le fait de nommer α le "paramètre de précision". Par conséquent, pour calculer la borne supérieure I_2, on calcule $(distr\ G_P)(p^*)$. Le calcul de la borne inférieure se fait de la même manière, en remplaçant $\underline{.}$ par $\bar{.} :]0, 1] \to \mathbb{Z}$ avec $\bar{p} = \lceil \widetilde{p} \rceil$. Des preuves théoriques ont été faites dans [1] afin de borner les valeurs calculés par cet algorithme, et estimer l'erreur sur ces calculs. Ces preuves ne seront pas présentées dans ce rapport. La procédure PRO est résumée dans l'Algorithme 4.

On remarque que pour cet algorithme, il faut effectuer des multiplications de polynômes au début afin de construire les deux polynômes généralisés G_P et H_P. Pour effectuer des multiplications de polynômes, on préfère d'habitude utiliser des multiplications rapides telle que la multiplication via FFT, ou l'algorithme de Karatsuba. Néanmoins, plusieurs tests effectués ont montré que les polynômes manipulés dans notre cas ont un caractère extrêmement creux, ce qui indique qu'une simple multiplication naïve qui profite du caractère creux de polynômes est suffisante pour obtenir des bonnes performances. Ainsi, pour notre implémentation, on se contente de la version naïve où les coefficients nulls sont éliminés du calcul tout de suite. Les résultats expérimentaux qui seront présentés dans la suite du rapport, valident notre choix.

Algorithme 4 : Polynomial Rank Outlining Algorithm [1]

Données : L'ensemble des d distributions $\boldsymbol{P} = \{\boldsymbol{S_i}\}_{1 \leq i \leq d}$. La probabilité de la clef secrète p^*. Le paramètre de précision α.

Résultat : L'interval $I = [I_1, I_2]$ contenant $rang(K^*, P)$.

$G_P(x) \leftarrow \prod_{S \in P} \sum_{s \in S} x^{\underline{s}}$;

$H_P(x) \leftarrow \prod_{S \in P} \sum_{s \in S} x^{\bar{s}}$;

$I_1 \leftarrow (distr\ H)(p^*)$;

$I_2 \leftarrow (distr\ G)(p^*)$;

retourner I;

Chapitre 3

Partie Expérimentale

Dans ce chapitre, nous allons présenter les résultats obtenus avec les différents algorithmes cités, ainsi qu'analyser la performance de ces algorithmes sur la base ASCAD de la partie 1.3.

3.1 Configurations

Les tests ont été tous effectués sur les machines de la PPTI de l'université. L'attaque CPA a été effectué sur la base ASCAD (qui utilise l'algorithme AES-128) avec un nombre croissant de 10 jusqu'à 62 traces, tout en connaissant les masques utilisés. On obtient ainsi une distribution de probabilités ou "scores" avec chaque nombre de traces testé. Ayant 52 distributions, les algorithmes ont été testé sur ces 52 rangs différents de la clef secrète et des mesures de performance ont été effectué.
Pour la partie expérimentale, on utilise les masques comme nous avons expliqué dans la partie 1.3, pour faire une attaque sur un algorithme "AES-128" basique, puisque notre objectif n'est pas d'attaquer l'AES-128 masqué, mais plutôt d'évaluer les algorithmes d'estimation de rang de clef. Mais en effectuant un test supplémentaire, on a validé qu'en ne connaissant pas les masques, on ne réussit pas à retrouver la clef secrète, même avec les 60,000 traces de la base (avec 10,000 traces par exemple, le rang de la clef secrète est estimée à 2^{97}).

3.2 Retour sur la Base ASCAD

Afin d'évaluer la performance des algorithmes, il fallait choisir des clefs de différents rangs pour faire les tests. Pour cela, on a commencé par estimer le rang de la clef secrète avec de 10 à 62 traces sur la base ASCAD. La courbe 3.1 montre l'évolution du rang de cette clef (en log_2), en fonction du nombre de traces. On peut remarquer que le log_2 du rang diminue avec l'augmentation du nombre des traces. Ce rang commence avec une valeur de 2^{127} pour

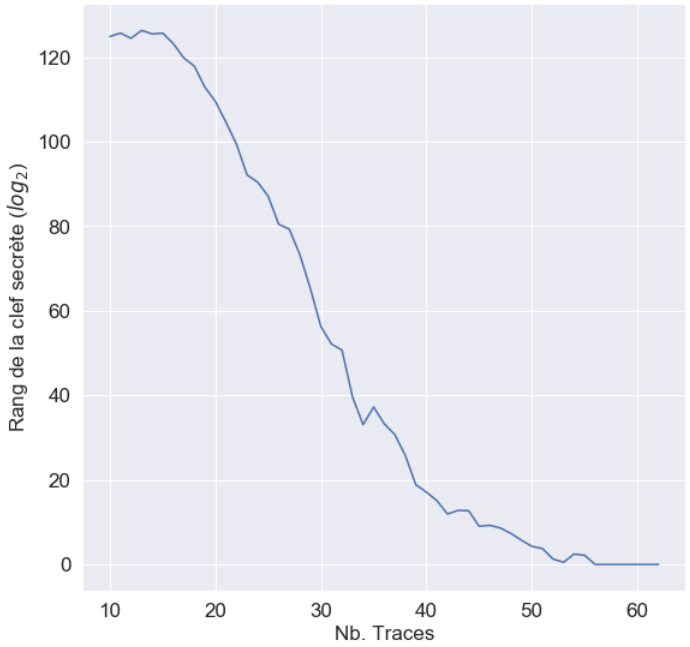

FIGURE 3.1 – Evolution du rang de la clef secrète en fonction du nombre de traces utilisés pour l'attaque sur la base ASCAD

10 traces, et arrive à la valeur minimale avec les 62 traces. Ces estimations ont été réalisées en utilisant les algorithmes implémentés, en plus d'autres mécanismes d'évaluations. La courbe indique que la connaissance d'un nombre insuffisant de traces dans une attaque CPA ne permet pas de retrouver la clef secrète. En effet, en fonction des mesures de consommation électrique faites, un nombre différent de traces pourrait être nécessaire pour la réussite de l'attaque. Ainsi, l'attaquant a intérêt à intercepter le plus de messages possibles pour arriver à casser le secret.
A l'aide de ces observations, on a pu utiliser des clefs de rangs différents afin d'évaluer les algorithmes étudiés.

3.3 REA

L'algorithme REA a été testé sur les clefs des scores de la base ASCAD, ayant des rangs allant de 1 jusqu'à 2^{127}. L'évaluation de la performance a été faite en mesurant la différence entre les valeurs en log_2 des bornes supérieure et inférieure de l'intervalle $I = [I_1, I_2]$; cette différence représente $log_2(\frac{I_2}{I_1}) = log_2(I_2) - log_2(I_1)$. Les résultats figurent sur les courbes de la Figure 3.2, en faisant tourner l'algorithme pour des durées variables. Pour ces tests, les dimensions ont été fusionnées en des dimensions d'au plus 2^{24} valeurs, et les temps d'exécution mentionnés n'incluent pas cette phase de pré-traitement des données.
On peut remarquer à partir de cette figure que faire tourner l'algorithme pour plus de temps diminue l'écart entre les borne supérieure et inférieure estimées. Ce dernier est à son maximum

quand on fait tourner l'algorithme pour 5 secondes, et atteint le minimum pour une exécution de 100 secondes. Cette comparaison sur les temps d'exécution a été faite puisque l'algorithme du REA en pratique ,pour des clefs de rangs non triviaux, continue à tourner jusqu'à l'épuisement des ressources puisqu'il n'arrive pas au cas d'arrêt où tout l'espace des clefs a été traité, ce qui revient à renvoyer la valeur exacte du rang de la clef secrète. En effet, la convergence de cet écart devient plus lente quand on laisse l'algorithme tourner pour plus de 100 secondes, puisque les volumes manipulés deviennent plus petits et plus nombreux au fur et à mesure du calcul. Ainsi, on arrive à un épuisement de ressources bien avant de retrouver le rang de la clef. Pour cela, une exécution de 100 secondes était pour nous déjà une bonne borne pour estimer l'efficacité de l'algorithme. On verra dans la suite que l'intérêt des autres algorithmes est justement de fournir des intervalles plus précis que ce que REA fournisse.
De plus, les courbes montrent que les clefs de rang 2^{90} environ présentent les cas les plus difficiles pour l'algorithme. En effet, l'écart prend son maximum pour ces rangs, et même en exécutant l'algorithme pour 100 secondes, cet écart reste à 7.5 environ, ce qui n'est pas négligeable puisqu'on est à des puissances de 2 très grandes. Les rangs de clefs compris entre 1 et 2^{40} ont été les plus faciles pour l'algorithme, et une exécution de 5 secondes était parfois suffisante pour retourner une valeur exacte (puisque la différence entre les bornes est de 2^0), ce qui est logique quand la clef a un rang de petite valeur, c'est-à-dire qu'elle se trouve au "sommet" de l'espace des clefs, et le choix de point de découpage devient plus facile éliminant beaucoup plus de volume à chaque itération. Contrairement au cas difficile d'une valeur de 2^{90}, où la clef se retrouve au "centre" de l'espace des clefs, le choix de point de découpage par l'algorithme n'étant pas facile, l'élimination des morceaux de l'espace prend plus de temps.
D'autres tests ont été lancés afin de montrer l'effet de la fusion des dimensions sur la précision de l'intervalle fourni. Le tableau 3.1 montre l'écart entre le log_2 des bornes de l'intervalle I en faisant tourner l'algorithme pour 100 secondes, sur le cas où la clef secrète est de rang approximatif 2^{90}, qui est le plus difficile à calculer pour le REA comme validé dessus. On peut observer que la différence entre les bornes diminue de façon non négligeable quand on fusionne davantage les dimensions, et que l'intervalle retourné est le plus précis pour la fusion en des tailles de 2^{24} valeurs (on peut voir que la différence commence à une valeur de 64.55 sans fusion, et atteint 7.41 en faisant cette fusion de 2^{24} valeurs).

3.4 EREA

Afin de tester l'algorithme EREA, on a décidé de le faire tourner sur le cas le plus difficile à traiter par l'algorithme REA, c'est-à-dire pour une clef de rang entre 2^{80} et 2^{90}. La Figure 3.3 montre l'évolution des bornes de l'intervalle calculées à différents arrêts durant l'exécution du REA. L'intervalle estimé par EREA est donné pour une valeur de confiance de 99.9%, avec

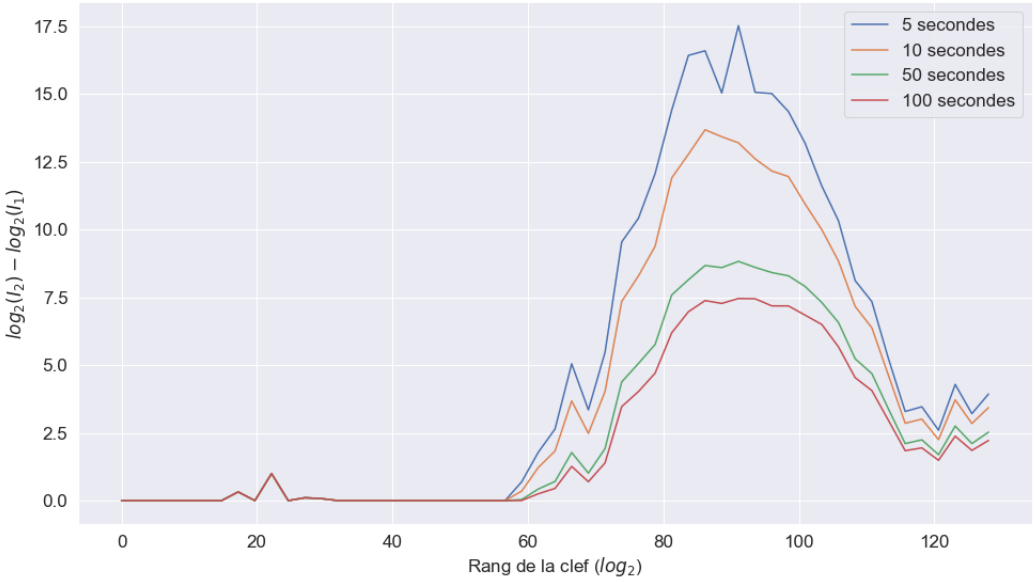

FIGURE 3.2 – Ecart entre log_2 des bornes de l'intervalle retourné par l'algorithme REA, pour différents rangs de clefs, pour différentes durées d'exécution.

Taille de Fusion	$\mathbf{log_2(I_1)}$	$\mathbf{log_2(I_2)}$	$\mathbf{log_2(I_2) - log_2(I_1)}$
Sans Fusion	53.76	118.29	64.55
2^{16}	74.08	97.96	23.88
2^{24}	83.29	90.70	7.41

TABLE 3.1 – Variation de la taille de l'intervalle $I = [I_1, I_2]$ retourné par l'algorithme REA après 100 secondes d'exécution, en fonction de la taille de la fusion effectuée sur les dimensions

un échantillonnage de 2^{24} clefs en moyenne. On précise que l'intervalle peut être calculé pour plusieurs valeurs de confiance avec le même échantillonnage effectué, donnant plus d'informations sur le vrai rang de la clef.

On peut remarquer que la taille de l'intervalle diminue plus rapidement pour l'algorithme EREA avec l'échantillonnage. En effet, cet intervalle reste de grande taille et converge très lentement au cours de l'exécution du REA. Contrairement au EREA, où l'exécution de l'échantillonnage prend quelques secondes pour arriver à estimer des bornes plus précises, qui s'approchent à une plus grande vitesse du rang exacte de la clef.

De plus, l'échantillonnage dure plus longtemps quand plus de temps est passé à exécuter le REA. Cela est dû à la croissance du nombre de volumes découpés, puisque pour échantillonner, l'algorithme itère sur tous les volumes restants dans l'espace. Néanmoins, ce temps n'est pas très pénalisant par rapport au REA.

Le nombre d'échantillons tirés de l'espace restant joue un rôle aussi sur la précision de l'inter-

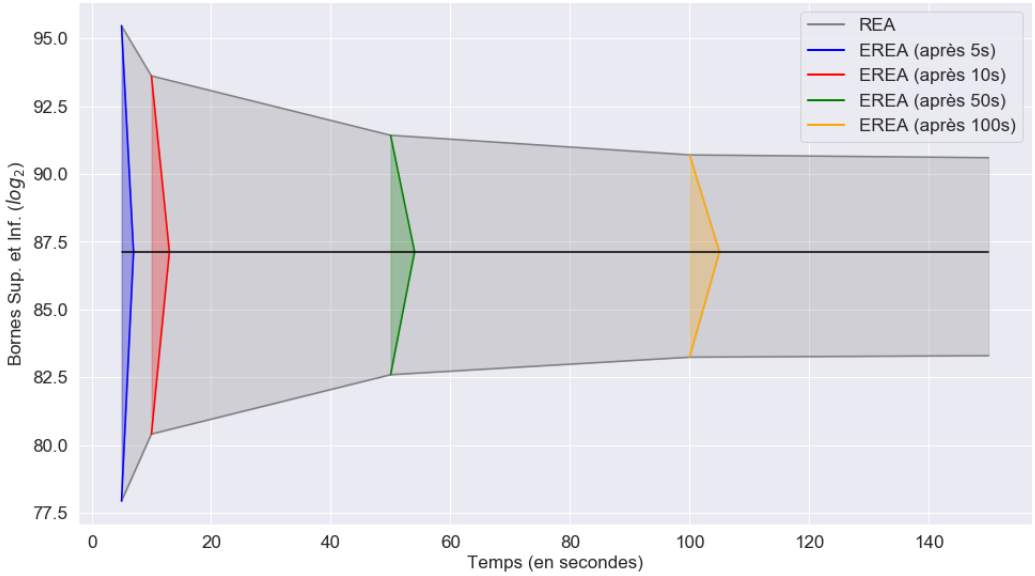

FIGURE 3.3 – Evolution des bornes de l'intervalle I retourné par l'algorithme REA, ainsi que par EREA exécuté à plusieurs moments du REA (après 5s, 10s, 50s et 100s du REA), avec un échantillonnage de 2^{24} clefs environ de l'espace restant, et un paramètre de confiance $\alpha = 0.999$

valle retourné. La Figure 3.4 montre que l'écart entre les bornes diminue considérablement avec l'augmentation du nombre d'échantillons. Cela vient de la formule de l'intervalle de confiance expliquée dans 2.2, où l'augmentation de l'échantillonnage permet d'augmenter la précision de l'intervalle.

3.5 PRO

Comme pour REA et EREA, l'évaluation de l'algorithme PRO a été faite en le faisant tourner sur les probabilités de la base ASCAD, avec le rang de la clef secrète allant de 1 à 2^{127}. Les courbes de la Figure 3.5 montrent l'écart entre le log_2 de la borne supérieure et inférieure de l'intervalle résultant pour différentes valeurs du paramètre de précision α.

Une première observation à faire sur ces courbes, c'est que l'écart entre les bornes ne dépassent jamais une valeur de 5 pour n'importe quel rang de clef, et même avec une faible valeur de paramètre de précision ($\alpha = 2^7$). L'augmentation de la valeur de α entraîne une diminution rapide dans cet écart, dû à la présence de plus de termes distincts dans les polynômes généralisés G_P et H_P et ainsi un calcul plus exacte et plus précis des bornes de l'intervalle I. Cet écart atteint vite une valeur inférieure à 0.5 pour tous les rangs de clefs possibles, avec $\alpha = 2^{15}$, pour laquelle le temps moyen d'exécution est d'environ 50 secondes. Ce dernier est le temps maximum pris parmi les différentes valeurs du paramètre de précision testées, ce qui indique que l'algorithme est "rapide" même pour des grandes valeurs de α (On rappelle que la multiplication de poly-

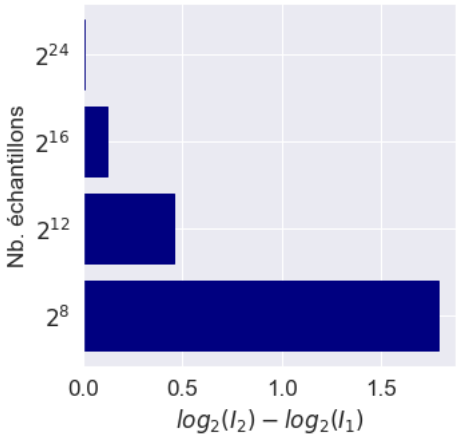

FIGURE 3.4 – Ecart entre log_2 des bornes de l'intervalle $I = [I_1, I_2]$ retourné par l'algorithme EREA, après 100s d'exécution du REA, en fonction du nombre d'echantillons tirés (avec une valeur de confiance $\alpha = 0.999$)

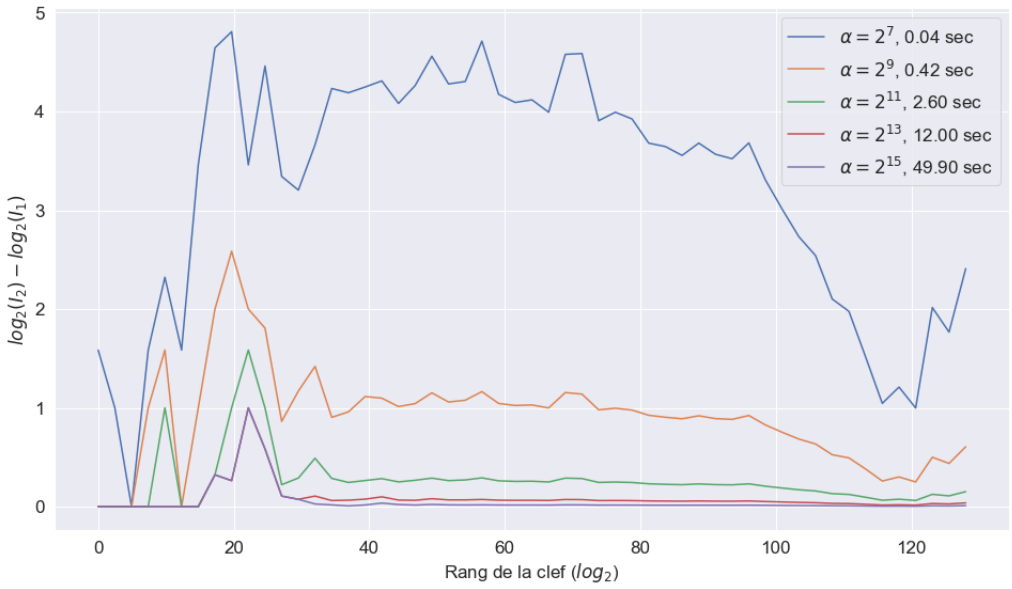

FIGURE 3.5 – Ecart entre log_2 des bornes de l'intervalle retourné par l'algorithme PRO, pour différents rangs de clefs, avec plusieurs valeurs du paramètre de précision α

nômes utilisée ici est la multiplication naïve qui profite du caractère creux des polynômes, ce qui valide notre justification faite dans la partie 2.3).
De plus, l'augmentation de la valeur de α entraîne aussi une courbe de plus en plus lisse en fonction des rangs de la clef secrète. En fait, la courbe avec $\alpha = 2^7$ présente beaucoup de fluctuations entre les différents rangs, signifiant moins de précision dans le calcul des bornes, à cause du manque de termes dans les polynômes généralisés calculés. Par contre, la courbe devient de plus en plus lisse en augmentant la valeur de α jusqu'à avoir une forme complètement lisse avec $\alpha = 2^{15}$ où la précision de calcul est la plus grande.
D'autre part, on remarque que contrairement aux REA et EREA, le rang le plus difficile à estimer pour le PRO est compris entre 2^{10} et 2^{30}. En effet, pour les clefs de petits rangs, une grande valeur de α est nécessaire pour qu'ils soient représentés par plus de termes distincts dans les polynômes généralisés calculés durant l'algorithme, afin d'obtenir des bornes plus exactes. Cela peut être observé à partir de la différence des résultats entre la plus petite et la plus grande valeur de α testé pour les rangs entre 2^{10} et 2^{30}.

3.6 Comparaison : REA, EREA & PRO

Dans le but de comparer les trois algorithmes présentés, les courbes de la Figure 3.6 montrent les résultats obtenus avec la meilleure configuration des paramètres pour chacune des procédures. On peut d'abord remarquer que les algorithmes PRO et EREA estiment des intervalles beaucoup plus précis du rang de la clef que l'algorithme REA. Les rangs les plus difficiles pour REA sont entre 2^{80} et 2^{90} à une valeur de différence de 7.5, alors que le PRO présente sa pire estimation pour les clefs de rang 2^{20}.
Les algorithmes REA et EREA fonctionnent mieux que le PRO avec les clefs de petits rangs. Néanmoins, en pratique, les clefs de petits rangs sont facilement attaquables par des procédures de types SCA et ne nécessitent pas des mesures avec des algorithmes d'estimation de rang. Le cas le plus intéressant est avec les clefs de rangs importants, où l'algorithme REA ne fournit pas des résultats précis.
Un autre avantage du PRO par rapport aux deux autres est son temps d'exécution. On voit sur la figure que la meilleure configuration du PRO ne prend que 49 secondes environ pour son exécution, alors que le REA nécessite de s'exécuter pendant 100 secondes, ainsi que le EREA qui attend 100 secondes d'exécution du REA avant de faire l'étape de l'échantillonnage. Et cela ne prend toujours pas en compte le temps de prétraitement pour fusionner les dimensions (d'environ 10 secondes), un traitement qui n'est pas nécessaire pour l'exécution du PRO. Malgré que PRO et EREA sont efficaces pour les grands rangs, le temps d'exécution du PRO reste inférieur par rapport à EREA.
Par contre, l'avantage des REA et EREA par rapport au PRO c'est que leur exécution peut

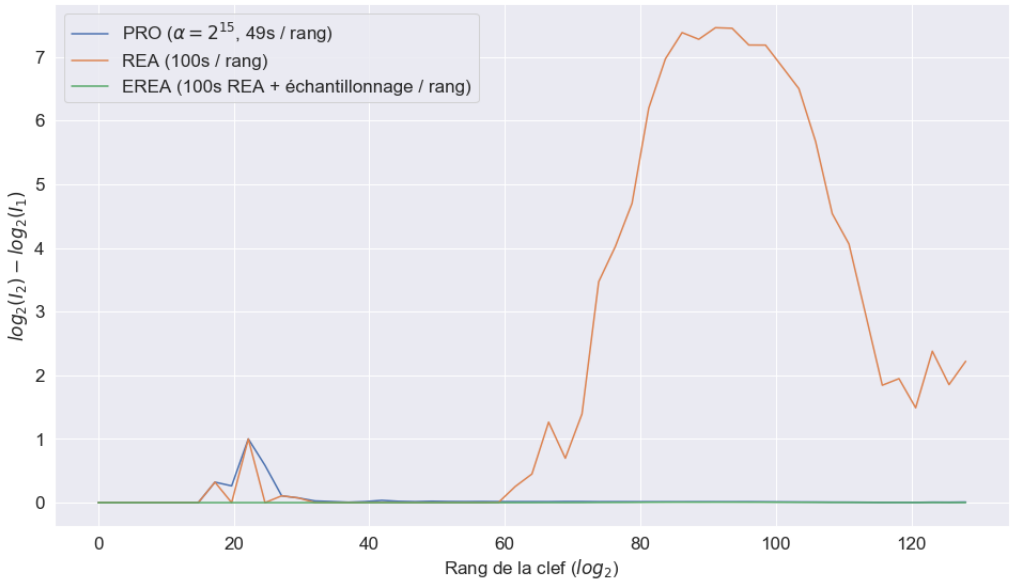

FIGURE 3.6 – Comparaison entre les trois algorithmes pour différents rangs de clefs, avec les meilleures configurations pour chacun (REA exécuté pendant 100s, EREA avec 100s et 2^{24} échantillons en moyenne, PRO avec $\alpha = 2^{15}$)

être interrompue à tout moment et repris plus tard. En effet, on peut arrêter l'exécution du REA à plusieurs moments, afin d'exécuter le EREA, et ensuite reprendre les mêmes résultats pour continuer le traitement. Cela n'est pas possible avec le PRO, où une exécution signifie la construction de polynômes et le calcul des distributions. Arrêter son exécution avant la terminaison ne fournit pas de résultats et ne permet ainsi aucune analyse. Alors que des estimations peuvent être faites sur le rang de la clef en interrompant le REA (ainsi que le EREA pendant l'échantillonnage).

D'autre part, l'algorithme du EREA présente l'avantage de pouvoir retourner plusieurs intervalles correspondant à plusieurs valeurs de confiance, afin de faire des analyses sur le rang de la clef, et ce pour une même exécution de l'échantillonnage, ce qui permet d'avoir des intervalles avec différentes précisions pour différentes valeurs de confiance.

En conclusion, les observations montrent que quand le temps d'exécution est un facteur important dans le calcul, l'algorithme PRO serait le plus efficace. Sinon, quand la précision du calcul est la plus importante, et quand on veut analyser les résultats à plusieurs moments de l'algorithme, le EREA serait le plus adapté. Puisque EREA est une extension du REA, une exécution fournit des intervalles pour ces deux algorithmes, et donc exécuter EREA peut s'avérer plus intéressant puisque ça inclut déjà REA.

Chapitre 4

Conclusion

Pendant ce projet, on a été introduit au principe des attaques par canaux auxiliaires. On a pu étudier un exemple de ces attaques qui est l'attaque CPA, en l'implantant et en étudiant ses résultats sur la base ASCAD. Ensuite, on a pu voir le principe et la définition de "rang de clef secrète" grâce à l'article [1] et ses algorithmes. On a eu l'occasion de les implanter par nous même et de reproduire les résultats de performance présentés dans l'article, ce qui nous a permis de mieux comprendre leur fonctionnement.

A l'issue du travail effectué, on va terminer par documenter le code des algorithmes, ainsi que de les mettre sur Github, avec une interface utilisateur Python pour faciliter leur utilisation par les autres, tout en les exportant comme une librairie "open source".

On tient à remercier Prof. PROUFF pour son encadrement et sa présence en continue pour toutes nos questions, et pour avoir toujours organisé des réunions afin de nous aider à surmonter les difficultés qu'on rencontrait parfois dans la compréhension du sujet.

Bibliographie

[1] Bernstein, Daniel J., Tanja Lange, and Christine van Vredendaal. "Tighter, faster, simpler side-channel security evaluations beyond computing power." IACR Cryptology ePrint Archive 2015 (2015) : 221.

[2] Veyrat-Charvillon, Nicolas, Benoît Gérard, and François-Xavier Standaert. "Security evaluations beyond computing power." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2013.

[3] Sean Wallis. Binomial confidence intervals and contingency tests : Mathematical fundamentals and the evaluation of alternative methods. Journal of Quantitative Linguistics, 20(3) :178–208, 2013.

[4] Brier, Eric, Christophe Clavier, and Francis Olivier. "Correlation power analysis with a leakage model." International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2004.

[5] Benadjila, Ryad, et al. "Study of deep learning techniques for side-channel analysis and introduction to ASCAD database." ANSSI, France & CEA, LETI, MINATEC Campus, France, 2018.