Compte-rendu du projet Sécurité des composants Projet : attaque EMA AES-128

Auteur : Antoine Breesé - Mars 2023
Langages utilisés : MATLAB, Python
Mots clefs : AES 128 bits, Chiffrement, Sécurité des composants

Introduction au projet :

Le but de ce projet est d'implémenter une méthode d'attaque par canaux auxiliaires sur un FPGA. L'objectif est de retrouver la clef de chiffrement utilisée en analysant les traces de consommations déduite du rayonnement électromagnétique de l'objet.

Le schéma de la figure suivante rappelle la structure de l'algorithme de chiffrement AES.

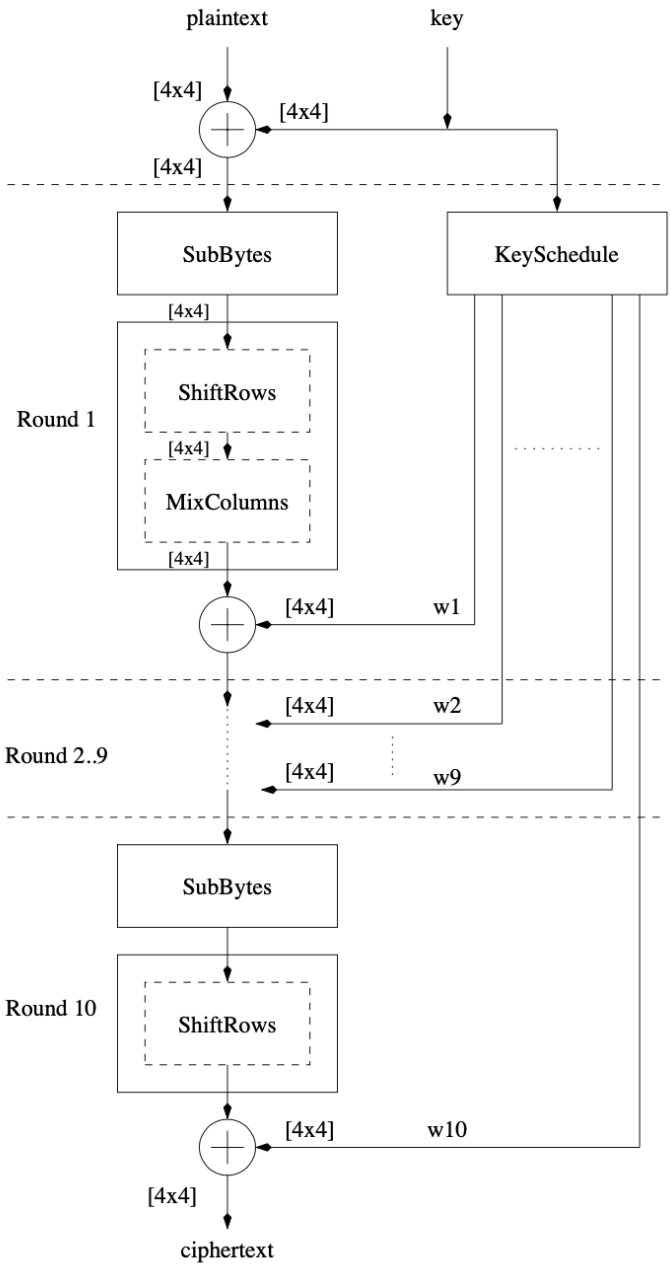

Figure 1 : Schéma de l'algorithme AES

Q0 - Parsing

Comme évoqué dans l'introduction, l'attaque portera sur l'analyse des traces de consommation de la carte. Celles-ci ont été récupérées et sauvegardées dans le dossier `SECU8917`. Celui-ci contient 20 000 fichiers des traces. Pour chacun d'entre eux, le titre du fichier suit la nomenclature suivante :
trace_AES_`indice`_key=`key`_pti=`pti`_cto=`cto`.

La première étape de ce projet consiste donc à lire chacune des fichiers pour en extraire les informations (méthode de parsing) et les sauvegarder dans les matrices suivantes :

- `key` : 20 000 x 16 (matrice des clefs de chiffrement utilisée)
- `pti` : 20 000 x 16 (matrice des *plain texte input)*
- `cto` : 20 000 x 16 (matrice des *cypher text output)*
- `traces` : 20 000 x 4 000 (matrice des traces de consommation (4 000 points mesurées pour chaque clef)

À noter que, pour une attaque réelle, la clef de chiffrement utilisée ne serait pas connue par l'attaquant. Pour ce projet pédagogique, celle-ci est connue afin de pouvoir comparer la clef déduite par l'attaque à la clef réellement utilisée. En outre, cette clef est la même pour les 20 000 fichiers.

Cette étape a été développée en Python pour des soucis d'efficacité. Le parsing s'exécute en environ 1min 30 secondes et fait appel à des expressions régulières.

La suite du projet a, quant-à-elle, été programmée en Matlab. On enregistre donc les fichiers csv dans des matrices matlab que l'on stocke dans un fichier .mat, plus rapidement chargeable par la suite.

```
% Enregistrement des fichiers CSV
data_cto = csvread('cto.csv');
data_key = csvread('key.csv');
data_pti = csvread('pti.csv');
data_traces = csvread('traces.csv');

% Enregistrement des matrices dans des fichiers MAT
save('cto.mat', 'data_cto');
save('key.mat', 'data_key');
save('pti.mat', 'data_pti');
save('traces.mat', 'data_traces');

%%
Nt = 20000;
%Chargement des fichiers.mat
load('cto.mat');
load('key.mat');
load('pti.mat');
load('traces.mat');

key = data_key(1,:); %la clef est la même pour les 20 000 fichiers
```

Q1 - Tracé d’une courbe de courant et détermination de la plage du chiffrement

```
%Q1 - Tracé d'une courbe de consommation

L1 = data_traces(1,:);
figure
plot(L1)
title("Affichage de la première trace de coura
xlabel('premier échantillon')
ylabel('courant')
```

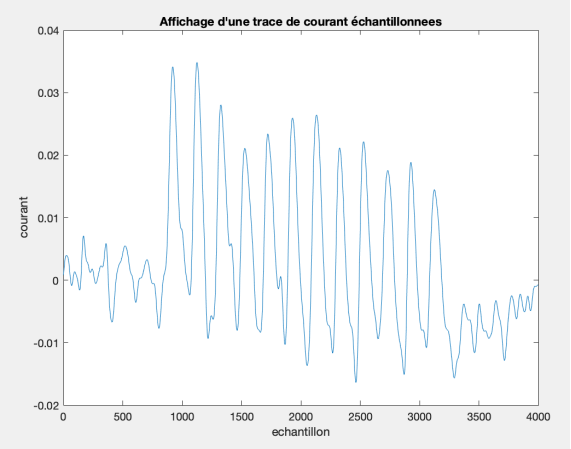

Figure 4 : Courbe de la question 1

Sur cette courbe, on observe les différents rounds de l’algorithme AES, marqués par un pic de consommation. 12 pics sont visibles sur la figure ci-dessus :

- 1 pic correspondant à l’initialisation du code VHDL (que l’on ignorera)
- 11 pics pour les différents rounds de l’algorithme AES (1 initial + 9 rounds + final)

On peut alors déduire le début et la fin du chiffrement [797, 3285]. L’attaque pourra alors être restreinte à cette intervalle des points de consommation.

Toutefois, pour être dans le cas général, on va déterminer ces valeurs sur la courbe moyenne de courant.

Q2 - Tracé de la courbe moyenne de courant et détermination de la plage du dernier round

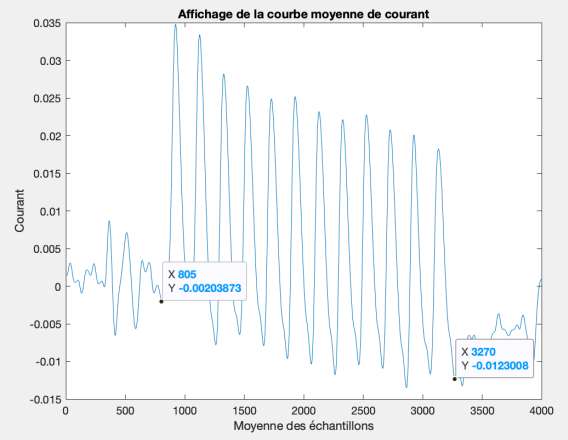

Figure 5 : Courbe moyenne de courant

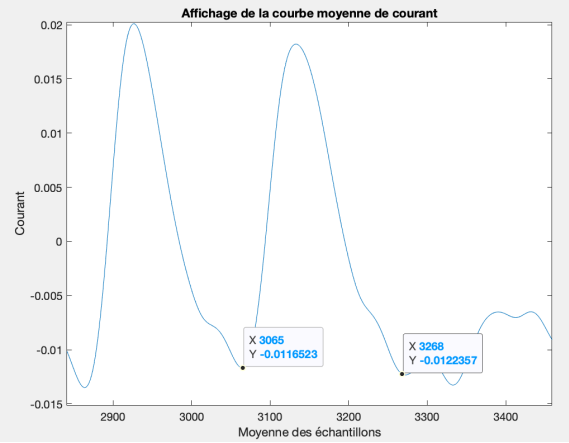

Figure 6 : Zoom sur la courbe moyenne des courants

- Sur cette courbe moyenne des courants, on détermine :
  - la plage moyenne du chiffrement : [805, 3270] (cf. courbe de gauche ci-dessus)
  - la plage du dernier round : [3065, 3268] (cf. courbe de droite ci-dessus). **Cette plage sera utilisée à la question 5 pour optimiser l'attaque sur ce dernier round.**

```
%Q2 - Tracé de la courbe moyenne de consommation

L1_mean = mean(data_traces,1);
figure
plot(L1_mean)
title("Affichage de la courbe moyenne de courant")
xlabel('Moyenne des échantillons')
ylabel('Courant')
```

Q3 - Justification de l'applicabilité du modèle du pods de Hamming

Pour modéliser la fuite, on peut utiliser plusieurs méthodes :

- Distance de Hamming qui mesure la similitude entre deux chaînes de bits ou vecteurs

- Poids de Hamming qui mesure combien de bits sont à l'état 1 dans une chaîne de bits ou un vecteur binaire.
- Hypothèses des clefs

Dans ce projet, **aucune information sur le circuit n'est fournie** on choisira donc d'adopter le modèle de la distance de Hamming.

Or, l'attaque est ciblée sur le dernier round, le schéma général (Figure 1) se résume donc à celui présenté figure 8.

Ce schéma présente plusieurs points Z0, Z1, Z2 et Z3 auxquels peut être appliquée la distance de Hamming. Il y a donc 6 possibilités de points d'attaques :

$$\binom{4}{2} = 6$$

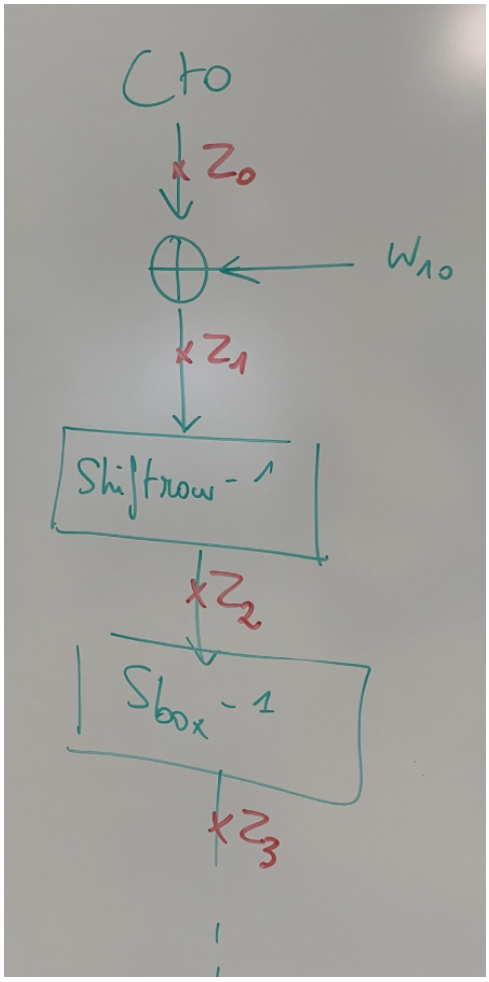

Figure 8 : Inversion du dernier round de l'algorithme AES et mise en évidence des points d'attaques

Un calcul théorique permet d'ores et déjà d'éliminer certaines possibilités entre Z0 et Z1: la clef n'apparaissant plus dans le calcul de la distance de hamming. Or, notre but étant de chercher une corrélation entre la distance de hamming et la clef w10, si cette dernière n'apparaît pas, l'attaque à ce point n'a aucun intérêt.

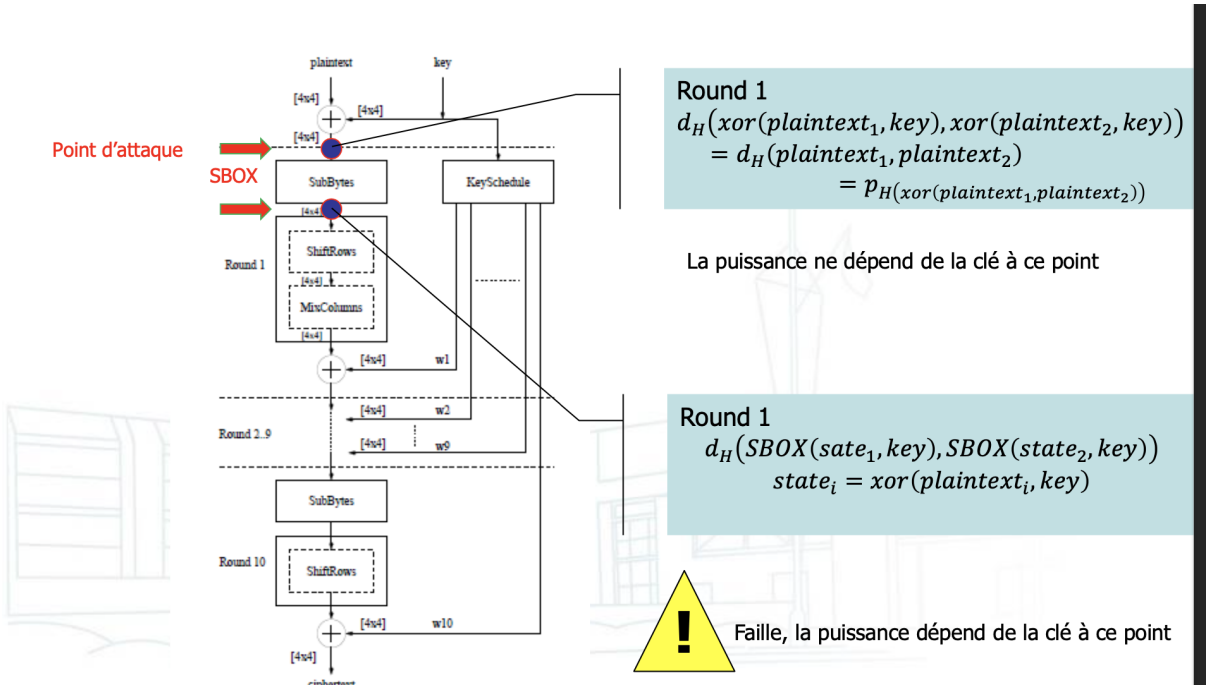

Figure 9 : Extrait du cours de Monsieur LE ROY - Illustration de la nécessité de bien choisir le point d'attaque

Je choisis donc arbitrairement d'appliquer le modèle du poids de Hamming entre Z0 et Z3.

L'influence du choix du point d'attaque sera étudiée dans la partie relative aux résultats de l'attaque.

Q4 - Détermination de la clé au round 10

Notre attaque va permettre d'approcher la valeur de la clef de chiffrement au dernier round. Afin de mesurer la précision de notre méthode, il sera utile de la comparer à la clef de chiffrement réellement utilisée. Cette dernière est générée à partir de la clef de chiffrement fourni dans le nom des fichiers et de la fonction `keysched2`.

```
%Q4

S_box = gen_s_box;

all_w = keysched2(uint32(reshape(key,[],4)));

w10_attendue = all_w(:,:,11);

disp('w10 attendue : ')
```

```
w10 attendue :
    60    64    77    37
    71   214    56   251
    64   128     0   120
     1   146   185   176
```

```
disp(w10_attendue);
```

Q6-7-8 - Implémentation de l’attaque

Méthodologie de l’attaque

L’objectif est d’aboutir à une version estimée de la clef `W_10` (clef de chiffrement au round 10 de l’algorithme AES).

Il s’agit d’une matrice de 4x4 uint8 (entre 0 et 255).

Chacune des 4x4 = 16 entiers de W_10 est appelé : `sous-clef`.

Pour chaque sous clef `i`, les 256 possibilités (0 à 255) sont `des hypothèses de clef de la sous clef i.`

Pour cela, nous allons :

- itérer sur chacune des 16 sous clefs :
  - tester chacune des 2^8 = 256 hypothèses de clef
  - Appliquer le modèle d’estimation de courant
  - Corréler l’hypothèse de clef au modèle
  - L’hypothèse de clef pour laquelle la correlation est maximale est gardée
  - w10_estimée[i] = hypothèse de clef gardée
- comparer la clef déterminée par l’attaque à celle connue (cf. introduction sur l’aspect pédagogique du projet).

Résultats de l’attaque entre Z0 et Z3

En attaquant entre Z0 et Z3, on obtient les courbes de corrélation maximales suivantes pour chacune des 16 sous clefs :

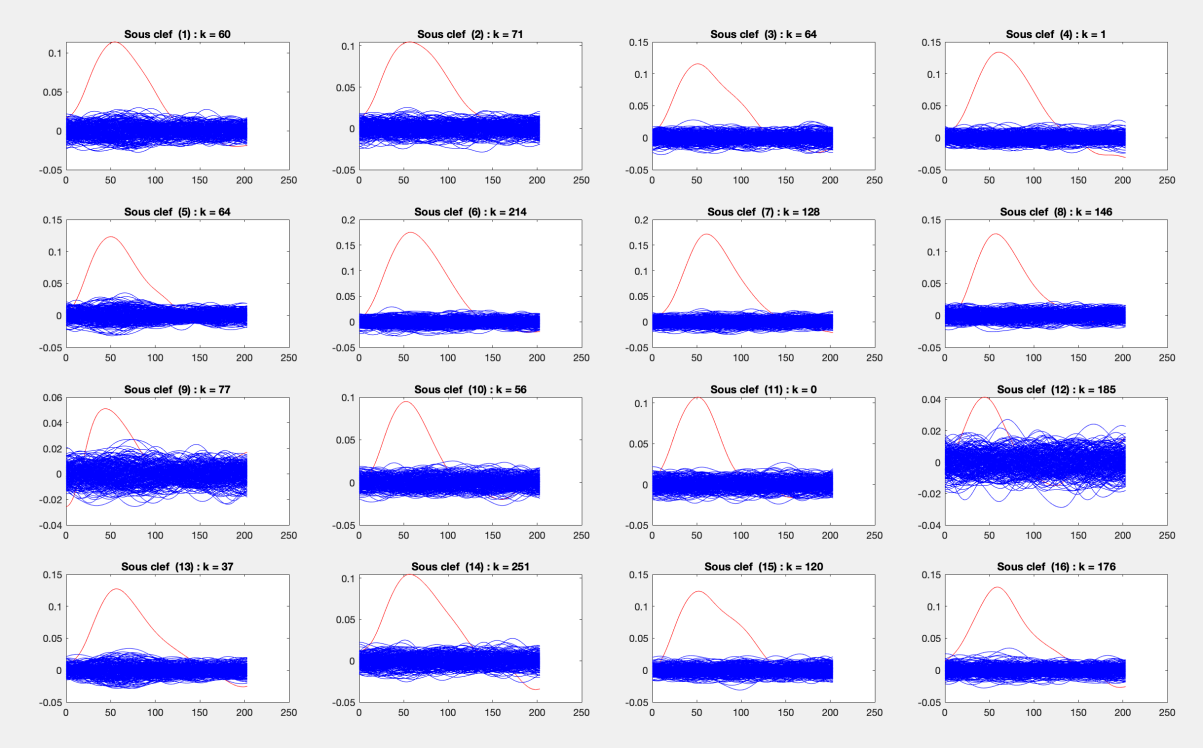

Figure 12 : Courbes de corrélation des 16 hypothèses de clef pour une attaque entre Z0 et Z3.

On remarque qu'en grande majorité, les courbes de corrélation maximales (mises en évidence en rouge) ressortent nettement du reste des hypothèses de sous clef à l'exception des sous clef n°9 (k = 77) et n°12 (k = 185).

Avec ces points d'attaques, notre méthode a permis de déterminer la clef correspondante à celle utilisée pour le chiffrement.

```
W10 calculée :
    60    64    77    37
    71   214    56   251
    64   128     0   120
     1   146   185   176
```

Étude de l'influence du choix du point d'attaque

Dans cette partie nous allons étudier l'influence du choix du point d'attaque sur la détermination de la clef et sur la précision de notre méthode.

Entre Z0 et Z1 :

Les figures suivantes montrent que ce point d'attaque n'est pas satisfaisant, aucune des sous-clefs n'ayant été trouvée correctement. La figure 16 confirme le calcul théorique présenté à la question 3 : ce point d'attaque ne permet pas de corréler la clef avec la distance de Haming, ce qui cohérent avec l'absence de corrélation maximale visible.

```
W10 calculée :
     0     0     0     0
    72   130    23   235
   152    87   152    87
   109   238    14    60
```

```
W10 attendue :
    60    64    77    37
    71   214    56   251
    64   128     0   120
     1   146   185   176
```

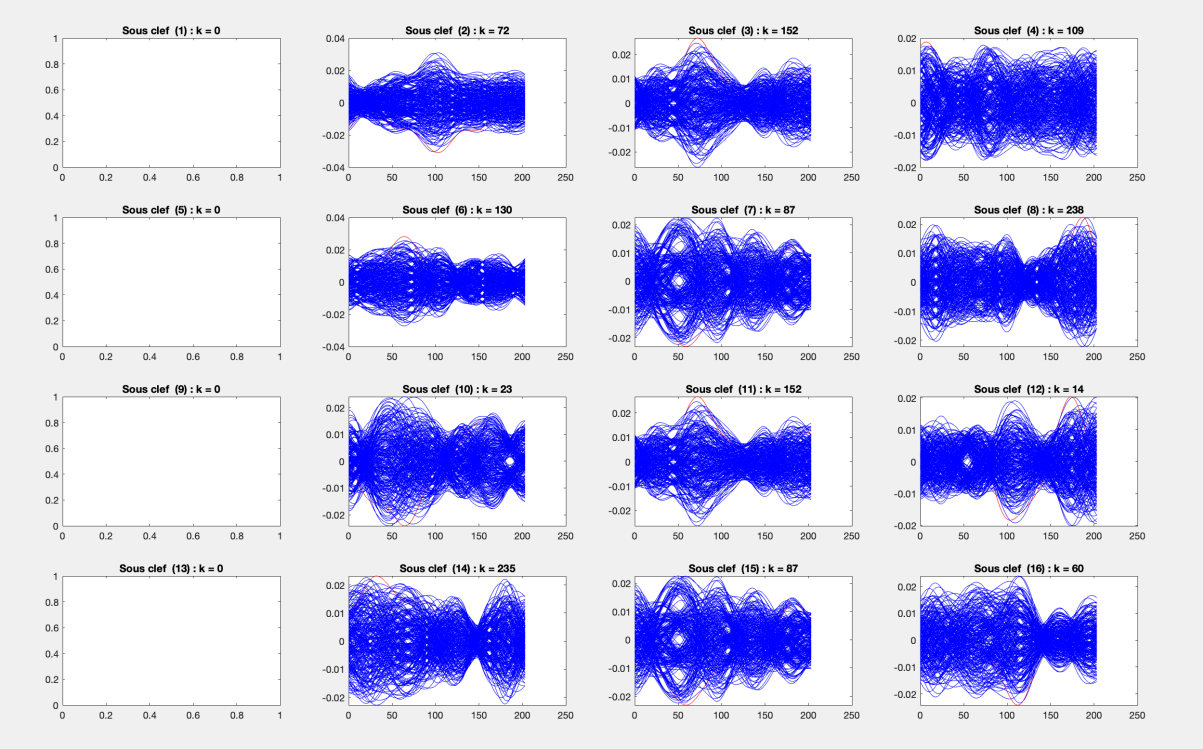

Figure 16 : Courbes de corrélation des 16 hypothèses de clef pour une attaque entre Z0 et Z1.

Entre Z1 et Z3

Une attaque entre les points Z1 et Z3 permet d'améliorer nettement l'attaque précédent. 12 sous-clefs sur 16 sont alors correctes. La figure 19 illustre le fait que les 4 sous-clefs incorrectes ne présentent pas de corrélation maximale suffisament visible : les courbes rouges sont similaires à celles des autres hypothèses de sous-c. En outre, bien que certaines des sous-clefs aient correctement été identifiées, la courbe de correlation correspondante n'est pas toujours très satisfaisante (exemple : k = 67)

```
W10 calculée :
    67    64   197    37
    10   214    56   251
    64   128     0   175
     1   146    84    17
```

```
W10 attendue :
    60    64    77    37
    71   214    56   251
    64   128     0   120
     1   146   185   176
```

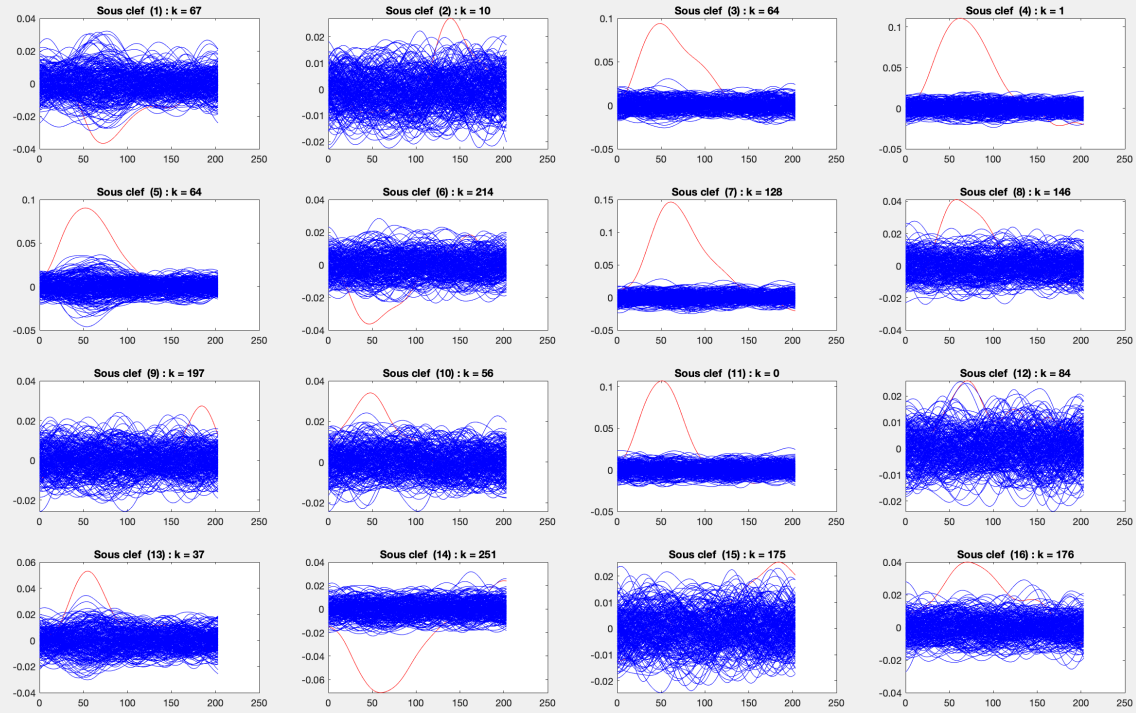

Figure 19 : Courbes de corrélation des 16 hypothèses de clef pour une attaque entre Z1 et Z3.

ℹ️ **<u>En comparaison avec la figure 12, il apparaît évident que le point d'attaque dont les résultats sont les plus satisfaisants est entre Z0 et Z3.</u>**

Q8

Enoncé non compris ⇒ question non réalisée.

Q9 - Guessing Entropy

Cas de l'algorithme AES

Pour l’algorithme AES, la guessing entropy (Entropie de devinette) mesure la quantité d'entropie (ou d'incertitude) que l'attaquant doit vaincre pour deviner la clé de chiffrement correcte. L'entropie est mesurée en bits, et elle représente le nombre de devinettes nécessaires pour deviner correctement la clé de chiffrement avec une probabilité donnée. Ainsi pour notre algorithem AES 128 bits, la guessing entropy est de 2^128.

Méthode élémentaire de calcul de l’entropie de devinette

Un autre moyen d’évaluer cette entropie de devinette est de déterminer le nombre minimal de traces de consommation nécessaires à la bonne évaluation de la clef de chiffrement.

Pour cela, j’exécute le code en diminuant le nombre de traces `Nt` utilisées jusqu’à ce que cela affecte le résultat de la clef.

- **avec un pas de 1000**

```
%for Nt = 20000:-1000:1000

Nt = (20000) : clef trouvée avec succès !
Nt = (19000) : clef trouvée avec succès !
Nt = (18000) : clef trouvée avec succès !
Nt = (17000) : clef trouvée avec succès !
Nt = (16000) : clef trouvée avec succès !
Nt = (15000) : clef trouvée avec succès !
Nt = (14000) : clef trouvée avec succès !
Nt = (13000) : clef trouvée avec succès !
Nt = (12000) : clef trouvée avec succès !
Nt = (11000) : échec !
Nt = (10000) : échec !
Nt = (9000) : échec !
Nt = (8000) : échec !
Nt = (7000) : échec !
Nt = (6000) : échec !
Nt = (5000) : échec !
Nt = (4000) : échec !
Nt = (3000) : échec !
Nt = (2000) : échec !
Nt = (1000) : échec !
```

  On observe sur la figure ci-dessus, que le seuil de traces nécessaires se situe entre 11000 et 12000 traces. On va donc recentrer la recherche en diminuant le pas

- **avec un pas de 100**

```
%for Nt = 12000:-100:10500
Nt = (12000) : clef trouvée avec succès !
Nt = (11900) : clef trouvée avec succès !
Nt = (11800) : échec !
Nt = (11700) : échec !
Nt = (11600) : échec !
```

Le seuil semble se situer autour de 11850 traces. On affine une dernière fois avec un pas de 10 pour confirmer cette hypothèse.

- **avec un pas de 10**

```
%for Nt = 11900:-10:11800
Nt = (11900) : clef trouvée avec succès !
Nt = (11890) : clef trouvée avec succès !
Nt = (11880) : clef trouvée avec succès !
Nt = (11870) : clef trouvée avec succès !
Nt = (11860) : clef trouvée avec succès !
Nt = (11850) : clef trouvée avec succès !
Nt = (11840) : clef trouvée avec succès !
Nt = (11830) : clef trouvée avec succès !
Nt = (11820) : clef trouvée avec succès !
Nt = (11810) : échec !
Nt = (11800) : échec !
```

En conclusion, on estime qu'en dessous de 11810 traces de consommation étudiées, la clef de chiffrement déterminée a un risque important de ne pas être correcte.

Afin d'avoir une métrique plus complète, il serait intéressant d'élargir cette conclusion en traçant un graphique présentant la probabilité de justesse de la clef en fonction du nombre traces utilisées.

Difficultés rencontrées

Parsing

Le parsing a été effectué en Python afin de gagner en efficacité en terme de temps d'exécution. Toutefois, l'optimisation de ce dernier a demandé une certaine **réflexion sur les bonnes fonctions ou bibliothèques à utiliser pour la manipulation de grands datasets** (fonctions natives, numpy, pandas, etc.)

Le temps a ainsi été diminué de 20 min à environ 1min30. La partie la plus longue étant celle d'écriture des matrices (2/3 du temps), le parsing prenant 1/3 du temps.

```
Matrices générées en 38.34010076522827s
mat_key : (20000, 16)
mat_cto : (20000, 16)
mat_pti : (20000, 16)
mat_traces : (20000, 4000)
Fichiers générés en 88.39318490028381s
```

Utilisation du shiftrow et influence sur la clef

Une fois le choix des points d'attaques réalisé, le calcul des différents états à Z0 et Z3 a été laborieux. En effet, une difficulté de compréhension quant à l'utilisation du shiftrow est survenue.

En effet, dans notre méthode d'attaque (cf schéma de la figure 8), les plaintexts utilisés pour Z0 et Z3 sont différents, il y a un décalage d'indice correspondant au shiftrow.

La figure 24 présente le résultat de l'algorithme lorsque ce décalage est oublié : on constate que les clefs de la première ligne (=colonne sur la figure 24) sont correctes (car non sujettes au shiftrow). Pour les autres, la corrélation n'est pas satisfaisante. J'en ai alors déduit un problème d'application du shiftrow, résolu avec l'aide du professeur encadrant.

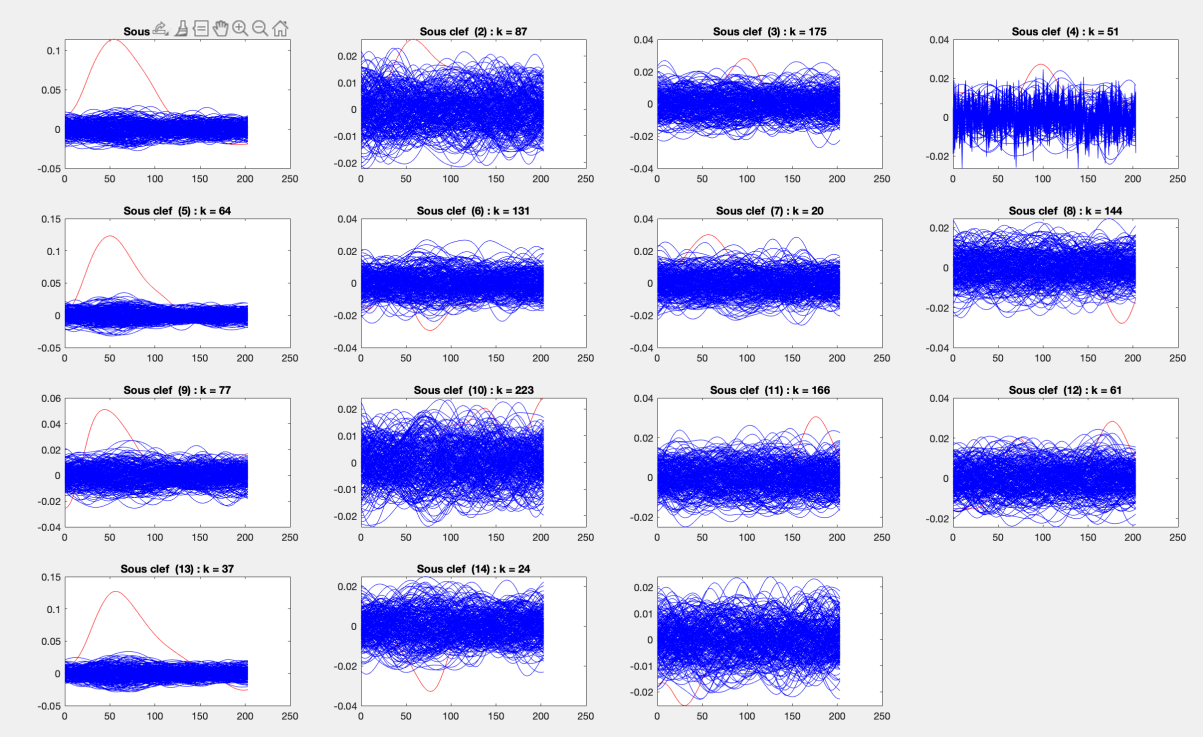

Figure 24 : Affichage des courbes de corrélation sans application du shiftrow

```
W10 calculée :
    60    64    77    37 #Première ligne OK
    87   131   223    24
   175    20   166   255
    51   144    61   159
```

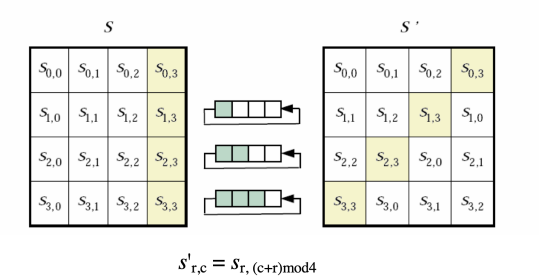

Figure 26 : Schéma du principe shiftrow

La solution est donc d’appliquer le shiftrow tel que dans l’extrait de code suivant :

```
Z0 = uint8(repmat(data_cto(:,shiftrow(i)),1,256));
cto_etendue = repmat(data_cto(:,i),1,256);
Z1 = uint8(bitxor(cto_etendue, assumption_etendue));
```

Conclusion

Ce projet a permis de mettre en application les techniques de l’algorithme AES-128 vues en TP et d’en approfondir la compréhension. Ce fut aussi l’occasion de renforcer mon aisance avec Matlab, étant plus naturellement habitué à utiliser Python.