SFA - Side-channels and fault attacks
Prof. Roman Korkikian et Nicolas Oberli
BA6 - 2023-2024

Récupération de clé d'un AES hardware sur le 9ème round

Attaques par faute différentielles

ANNEN Rayane **DUCOMMUN Hugo** **MARTINS Alexis**

26.05.2024

Table des matières

1 - Introduction

Ce rapport présente les résultats obtenus ainsi que la méthodologie d'attaque pour le deuxième laboratoire du cours « Side-Channels and Fault Attacks ».

1.1 - Préparation

Cette section présente les outils et le matériel utilisés pour effectuer l'attaque.

1.1.1 - Chipwhisperer

Nous utilisons le ChipWhisperer pour effectuer le laboratoire c'est d'abord une carte électronique dont la référence est STM32F303 puis une librairie python / C permettant d'effectuer des attaques en fautes et par canaux auxiliaires sur cette même carte.

La carte permet d'effectuer facilement et de manière reproductible des attaques en fautes, par exemple sauter une instruction au sein d'un programme en glitchant certains composants électroniques de la carte, en l'occurrence on va jouer avec les cycles d'horloges du ChipWhisperer.

L'attaque se fera dans un environnement python, dans Jupyter Lab.

1.1.2 - JeanGrey

Nous avons utilisé une librairie Python open-source contenant une boîte à outils permettant de faciliter les attaques par DFA (Differential Fault Attacks) :

https://github.com/SideChannelMarvels/JeanGrey

Tout particulièrement l'outil `phoenixAES` sur le round AES 9. Il requiert au minimum 4×2 fautes dans le round 9 avant le MixColumn.

2 - Méthodologie d'attaque

2.1 - Caractérisation

Afin de pouvoir facilement glitcher l'appareil, il faut identifier certains paramètres qui, quand ils sont appliqués injectent avec une haute probabilité une faute dans la carte électronique.

Afin d'identifier les paramètres (qui seront expliqués juste après) qui sont intéressants pour injecter des fautes, un programme de test qui est en fait un compteur jusqu'à 2 500 est utilisé.

Ce programme est exécuté plusieurs fois et en même temps qu'on glitch le ChipWhisperer. C'est alors que 3 cas se présentent :

1. Le programme s'exécute correctement et la valeur du compteur n'est pas affectée
2. Le programme s'exécute correctement mais la valeur du compteur est affectée, i.e. la valeur C du compteur n'est pas égale à 2 500. On considère que la faute a été correctement injectée.
3. Le programme crash. La faute n'est pas correctement injectée.

Les paramètres se définissent par un triplet (`ext_offset` , `offset` , `width`) et pour chaque triplet de paramètre le programme exécutant le compteur est lancé 10 fois. Si parmi les 10 fois, il y a au moins 5 fois où une faute a été correctement injectée, alors on considère que le triplet forme un « sweet spot » et sera réutilisé pour effectuer l'attaque en faute, car c'est triplet de paramètre où la probabilité d'avoir une faute correctement injectée est élevée, ce qui réduit le temps d'attente de l'attaque finale.

Le triplet de paramètre peut être décrit de la manière suivante :

- `ext_offset` : c'est essentiellement l'endroit où l'on va injecter la faute, par exemple dans notre cas on va s'intéresser aux 9ème round d'AES, on fera le lien avec un graphique des traces obtenues dans le Chapitre 2.2.
- `offset` : décalage où l'on commence à effectuer le glitch au sein d'un même cycle d'horloge.
- `width` : longueur du glitch (combien de temps), exprimé en pourcentage d'un coup d'horloge.

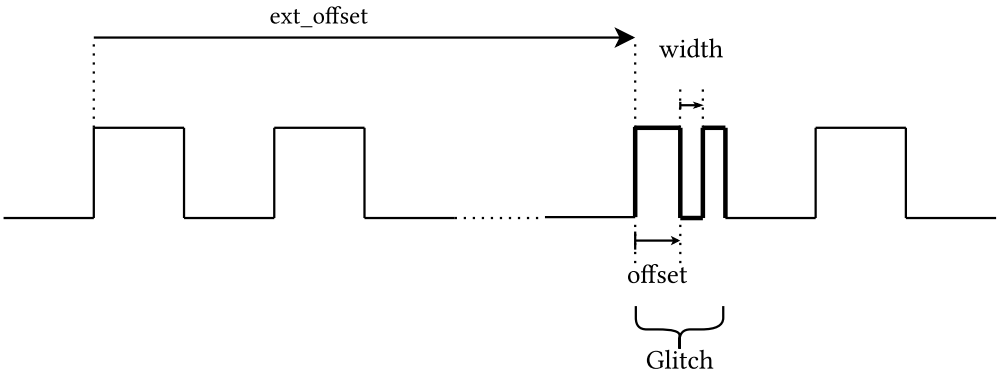

Figure 1 - Illustration des paramètres pour caractériser l'attaque

2.1.1 - Paramètres choisis

Dans notre cas, nous avons estimé qu'une vingtaine de valeurs ayant une forte probabilités de fauter suffisaient pour définir l'espace de notre « sweet spot ».

Concernant le paramètre `width` , l'intervalle est le suivant :$[36.6, 38.8]$

Concernant le paramètre `offset` , l'intervalle est le suivant : $[-31.8, -30.6]$

Concernant le paramètre `ext_offset` , on le définit lors de l'attaque en faute directement, voir Figure 3.

2.2 - Attaque en faute

L'attaque utilisée est une DFA décrite dans [1] de manière très formelle, c'est celle qui est implémentée dans `phoenixAES` .

Nous avons décidé de pratiquer l'attaque sur le round 9 d'AES, plus précisément lors du MixColumn.

On note un couple plaintext (texte en clair), un ciphertext (texte chiffré) : (P, C), le ciphertext comportant une faute est noté C^*.

L'attaque est décrite de la manière théorique : on suppose que la faute s'est injectée entre le round 8 et le round 9.

On considère précisément la sortie de MixColumn au round 9 qu'on note θ_9, celle-ci a été fautée sur un unique byte, toutefois puisque MixColumn à une propriété de diffusion, la faute s'est propagée sur 4 bytes (voir la Figure 2).

1. La première chose effectuée est de calculer toutes les possibilités de différence de 1 byte, i.e. les sorties de $\theta_9(x)$ avec x qui a un poids de Hamming égal à 1, on note cet ensemble e. La taille de cet ensemble équivaut au nombre possible de sorties fautées de MixColumn, ce qui correspond à 255×4 (nombre de sorties avec $\mathrm{HW}(x) = 1 \times$ nombre d'endroits où la faute peut être est injectée $(4$ bytes à la fois$)$).
2. Ensuite on considère notre couple (P, C) définit précédemment, on considère également C^*.
3. On tire K^{10}.
4. On calcule

$$\kappa = \gamma_{10}^{-1} \circ \sigma\big[K^{10}\big](C) \oplus \gamma_{10}^{-1} \circ \sigma\big[K^{10}\big](C^*)$$

où γ_{10}^{-1} est l'opération InvSbox au round 10 et $\sigma[k](a)$ est l'opération de AddRoundKey (dans notre cas un simple XOR) entre a et la clef de ronde k. Si κ se trouve dans l'ensemble e, on le rajoute dans une liste $\mathcal{L}$ de candidats.

5. On considère un nouveau couple (P, C) et C^* correspondant, ensuite on retourne à l'étape 2 en considérant la liste $\mathcal{L}$ de candidats, si le nouveau κ n'est pas dans e, supprimer le candidat de la liste, faire ainsi jusqu'à ne plus qu'avoir un seul candidat.

En pratique le choix de K^{10} est compliquée car on a 2^{128} possibilités (pour AES-128), l'implémentation dans le module python toutefois est plus astucieuse et permet de réduire considérablement le nombre de possibilités mais les détails sont hors de portée de ce rapport, ce qui est important de retenir, c'est que cette amélioration permet, avec deux paires de textes chiffrés dont les fautes sont aux mêmes bytes de récupérer les bytes correspondant de la clef de ronde.

Dans l'attaque réelle, nous pouvons nous passé d'une grande partie de ces considérations très théoriques, en effet, En reprenant les indices des bytes d'un ciphertext final sur une matrice 4×4 :

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

Si un byte a été modifié au début du round 9, voici la propagation de l'erreur induite :

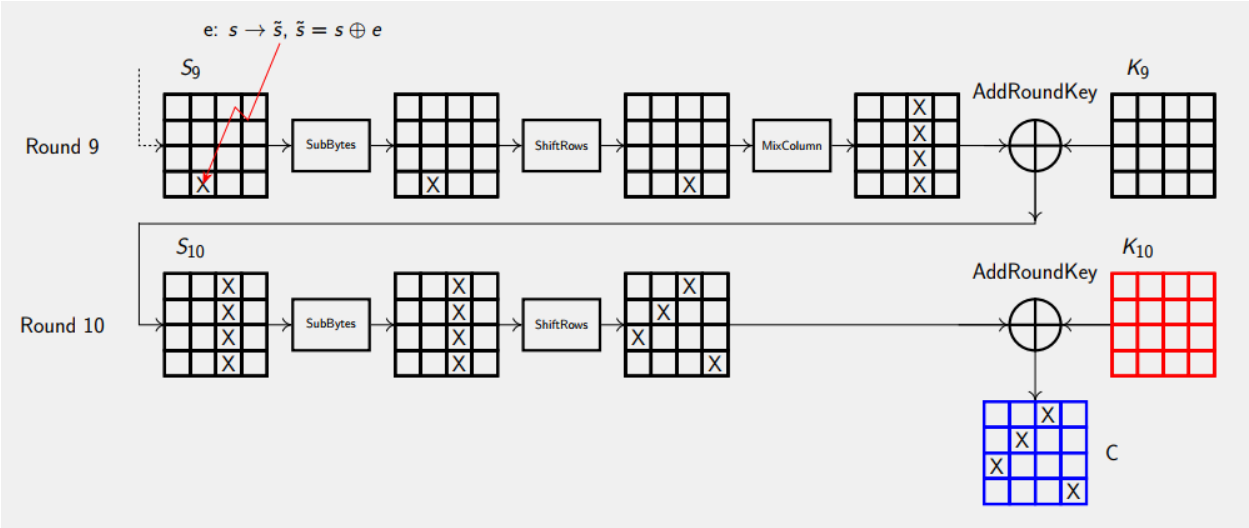

Figure 2 - Propagation de l'erreur injectée au début du round 9

On peut donc en déduire que si l'injection a eu lieu au bon moment, uniquement si les séquences de bytes suivantes seront modifiées dans le ciphertext final :

```
[0, 7, 10, 13]
[1, 4, 11, 14]
[2, 5, 8, 15]
[3, 6, 8, 12]
```

Ce que nous avons fait: moduler le paramètre `ext_offset` (entre 5800 et 6000, voir Figure 3) afin de fauter aux bons endroits, de sorte à ce qu'on arrive à trouver les ciphertexts fautés (C_i^*) qui ont ces 4 bytes de différents par rapport au ciphertext non fauté (C) :

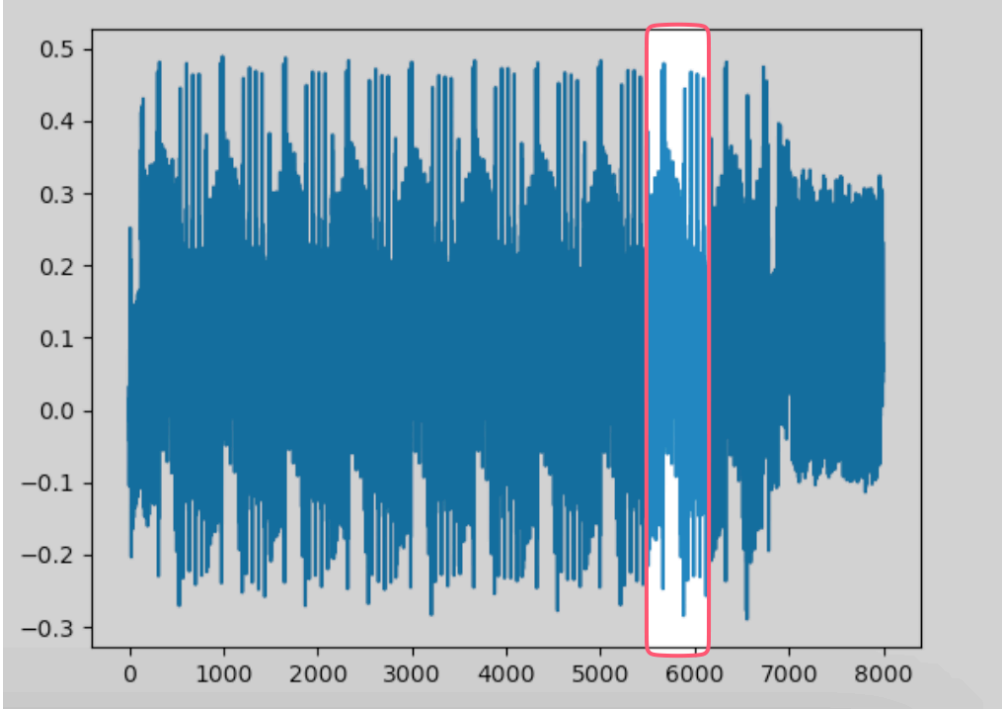

Figure 3 - Trace de la consommation lors du chiffrement AES, hint du lab avec spotlight sur la zone qu'on faute.

```
Original ciphertext :
b6a0d09b603785de90f30b516b341e54

Faulted ciphertexts (4 bytes changed) :

--XX----XX------------XX----XX-- (bytes 1, 4, 11 et 14)
b691d09b953785de90f30b3b6b34ed54
b6d9d09bff3785de90f30bef6b349054

------XX----XX----XX----XX------ (bytes 3, 6, 9 et 12)
b6a0d0e8603761de900d0b51d8341e54
b6a0d0826037e9de90990b51b7341e54

----XX----XX----XX------------XX (bytes 2, 5, 8 et 15)
b6a0619b60c585de88f30b516b341ed6
b6a03b9b604c85de1bf30b516b341ef7

XX------------XX----XX----XX---- (bytes 0, 7, 10 et 13)
d2a0d09b6037858d90f329516baa1e54
c1a0d09b603785cd90f31e516b981e54
```

`phoenixAES` requiert 2 ciphertexts fautés par diagonale. L'utilité de partir du round 9 est de réduire le nombre de ciphertexts fautés en input. Après avoir récolté ces ciphertexts, `phoenixAES` va appliquer une attaque ressemblante à la théorie présentée précédemment (avec des détails d'implémentation permettant d'accélérer le calcul).

2.3 - Informations retrouvées

À l'aide des ciphertexts fautés présentés précédemment on retrouve la clef

$$K_{10} = 0\text{xF9169C42CB5A6DA13033AAE4866C114E}$$

et ainsi la clef de chiffrement AES :

$$\text{Master Key} = \text{SCA}\{\text{Gl1tch\_AES}\}$$

Bibliographie

[1] G. Piret et J.-J. Quisquater, « A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD », in *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, C. D. Walter, Ç. K. Koç, et C. Paar, Éd., in Lecture Notes in Computer Science, vol. 2779. Springer, 2003, p. 77-88. doi: 10.1007/978-3-540-45238-6_7.