# Master CSI 2 2011-2012

## Cartes à puces

### Attaques sur les cryptosystèmes embarqués

Christophe Giraud
c.giraud@oberthur.com

## 1 Analyse différentielle sur le chiffrement AES

L'objectif de cet exercice est de retrouver en utilisant Matlab la clé d'un AES-128 utilisé lors du chiffrement de secrets sur une carte à puce.
Pour cela, vous avez à votre disposition 1 000 mesures de consommation de courant qui ont été faites lors de l'exécution de cet AES. Ces mesures ont été stockées dans des fichiers `Acqxxxx.dat`, où `xxxx` correspond au numéro de la mesure. La structure de ces fichiers est la suivante :

- les 16 premiers octets contiennent la valeur du message,
- ensuite la mesure de courant représentant l'exécution de l'AES est donnée par un tableau de 3472 floats.

Pour vous aider, la SBox de l'AES en format Matlab est donnée dans le fichier `lookuptable.m`. Pensez à copier les fichiers `Acqxxxx.dat` en local afin de ne pas être pénalisé par les accès réseaux.

*1.* Implémenter une fonction permettant d'ouvrir un fichier et de lire son contenu. Cette fonction devra avoir comme prototype :

*function [message, courbe] = readFile(indiceFichier)*

*2.* Implémenter une fonction prenant en entrée un octet du message et un octet de la clé, et ressortant la valeur correspondante de la sortie de la transformation *SubBytes*. Cette fonction devra avoir comme prototype :

*function byteOutput = outputSBox(byteInput, byteKey)*

*3.a.* Grâce aux fonctions précédentes, implémenter une analyse différentielle utilisant la distance des moyennes (DPA) en utilisant comme fonction de sélection le bit de poids faible du premier octet de la sortie de la transformation *SubBytes* lors du premier tour. Cette fonction devra avoir comme prototype :

$$\textit{function differences} = \textit{DPA(bytePosition)}$$

où *differences* est une matrice contenant les 256 différences des moyennes.

*3.b.* Dans un second temps :

- Développez une fonction qui calcule la valeur de la sous-clé la plus probable. Cette fonction devra avoir comme prototype :
  $$\textit{function mostProbableKey} = \textit{searchKey(differences)}$$
- Affichez un graphique représentant les 256 distances des moyennes (la courbe correspondant à l'hypothèse la plus probable devant être en rouge, les autres en bleu).

*4.* Faire de même pour les 15 autres octets et donner la valeur de la clé de l'AES la plus probable en hexadécimal.

*5.* Reprendre l'implémentation précédente en utilisant cette fois-ci comme fonction de sélection le poids de Hamming de la sortie de la transformation *SubBytes* lors du premier tour (si HW$<4$ alors groupe 0 sinon groupe 1). La fonction calculant la poids de Hamming devra avoir comme prototype :

$$\textit{function [hw]} = \textit{hammingWeight(input)}$$

Donner la valeur de la clé la plus probable en hexadécimal et expliquer la différence par rapport à l'attaque précédente.

*6.* Implémenter une analyse différentielle utilisant le coefficient de correlation de Pearson (CPA) en utilisant le poids de Hamming de la sortie de la transformation *SubBytes* lors du premier tour comme fonction de sélection.

$$\rho_{\hat{k}}(t) = \frac{n\sum_{i=1}^{n} P_i C_i(t) - \sum_{i=1}^{n} P_i \sum_{i=1}^{n} C_i(t)}{\sqrt{n\sum_{i=1}^{n} P_i^2 - (\sum_{i=1}^{n} P_i)^2}\sqrt{n\sum_{i=1}^{n} C_i(t)^2 - (\sum C_i(t))^2}}$$

Cette fonction devra avoir comme prototype :

$$\textit{function correlations} = \textit{CPA(bytePosition)}$$

où *correlations* est une matrice contenant les 256 corrélations.
Utilisez ensuite la fonction *searchKey* pour afficher un graphique représentant les 256 corrélations (la courbe correspondant à l'hypothèse la plus probable devant être en rouge, les autres en bleu).

*7.* Modifier les fonctions DPA et CPA afin qu'elles calculent aussi les maximums des courbes de différences / correlation toutes les 100 acquisitions afin de pouvoir tracer les courbes de convergence des courbes de différences / correlation. Ceci nous permettra de savoir à partir de combien d'acquisitions la bonne sous-clé arrive en tête de nos hypothèses. Ces fonctions devront donc avoir comme prototype :

*function [differences,convergences] = DPA2(bytePosition)*
*function [correlations,convergences] = CPA2(bytePosition)*

# 2 DFA sur le chiffrement DES

Un attaquant à réussi à perturber l'exécution d'un chiffrement DES en injectant une erreur sur la partie droite de l'entrée du $16^{\text{ème}}$ tour.

L'objectif de cet exercice est d'implémenter un outil en C permettant de retrouver la clé DES à partir des 20 couples (chiffré correct / chiffré erroné) contenus dans le fichier `couples.inc` (chaque chiffré est représenté sur deux unsigned int, le bit 0 étant à droite).

Les transformations du DES nécessaires à cet exercice sont données dans le fichier `transformationsDES.inc`. Pour information, les permutations `invP`, `IP` et `E` fonctionnent comme suit : le $i^{\text{ème}}$ bit de la sortie de la permutation $P$ sera égal au $P[i]^{\text{ème}}$ bit de l'entrée de cette permutation.

*1.* Implémenter une fonction permettant de retrouver les parties droites de l'entrée ($R_{15}$) et de la sortie ($R_{16}$) du $16^{\text{ème}}$ tour du DES à partir de la valeur du chiffré. Cette fonction devra avoir comme prototype :

*void inversionFP(unsigned int *chiffre, unsigned int *$R_{16}R_{15}$)*

*2.* Implémenter une fonction effectuant la permutation expansive sur $R_{15}$. Cette fonction devra avoir comme prototype :

*void permutationExpansive(unsigned int* $R_{15}$*, unsigned int *E*$R_{15}$*)*

*3.* Calculer maintenant la différentielle en entrée des SBox (i.e. $E(R_{15}) \oplus E(R'_{15})$). Ceci vous permettra de lister les SBox impactées par la faute.

*4.* A partir de $(R_{16}, R'_{16})$, implémenter une fonction calculant la différentielle attendue en sortie des SBox (i.e. $P^{-1}(R_{16} \oplus R'_{16})$). Cette fonction devra avoir comme prototype :

*unsigned int differentielleSortieSbox(unsigned int* $R_{16}$*, unsigned int* $FR_{16}$*)*

*5.* Implémenter une fonction calculant la sortie de la $i^{\text{ème}}$ boîte-S. Cette fonction devra avoir comme prototype :

*unsigned char sortieSBox(unsigned char* $SBox\_nb$*, unsigned char input)*

*6.* Implémenter une fonction permettant de tester les 8 équations et incrémenter le compteur associé aux sous-clés les vérifiant :

$$f_i(R_{15}, k_{16,i}) \oplus f_i(R'_{15}, k_{16,i}) = P^{-1}(R_{16} \oplus R'_{16})_i, \quad i \in \{0, \cdots, 7\}$$

Cette fonction devra avoir comme prototype :

*void rechercheSousCles(* *unsigned int* differentielle_entree_SBoxes,*
*unsigned int** $ER_{15}$*,*
*unsigned int** $EFR_{15}$*,*
*unsigned int differentielle_sortie_SBoxes,*
*unsigned int keyCounters*[8][64]*)*

*7.* Une fois les 20 couples $(C, C')$ traités, afficher la sous-clé la plus probable pour chacune des 8 SBox.

*8.* Quelle est la valeur de la clé du DES en utilisant le fait que le texte clair 0x54BA21EB7DD685D7 donne comme chiffré 0x092506C86D5CBBFD ?