Examen – Attaques sur carte à puce

Durée : 1h

Christophe Giraud

c.giraud@oberthur.com

Exercice 1. Un attaquant mesure la consommation d'une carte lors d'une signature RSA et obtient la courbe représentée en Figure 1.

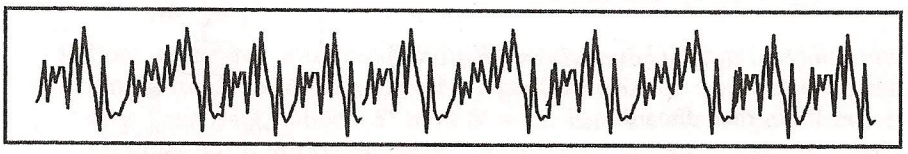


FIG. 1 – Courbe de consommation observée

1. En observant la courbe, combien de motifs différents distinguez-vous ?
2. Quel type d'algorithme est très probablement utilisé par la carte pour implémenter l'exponentiation modulaire ?
3. Comment l'attaquant en déduit-il la valeur de l'exposant ?

Exercice 2. Analyse différentielle de consommation de courant sur l'AES.

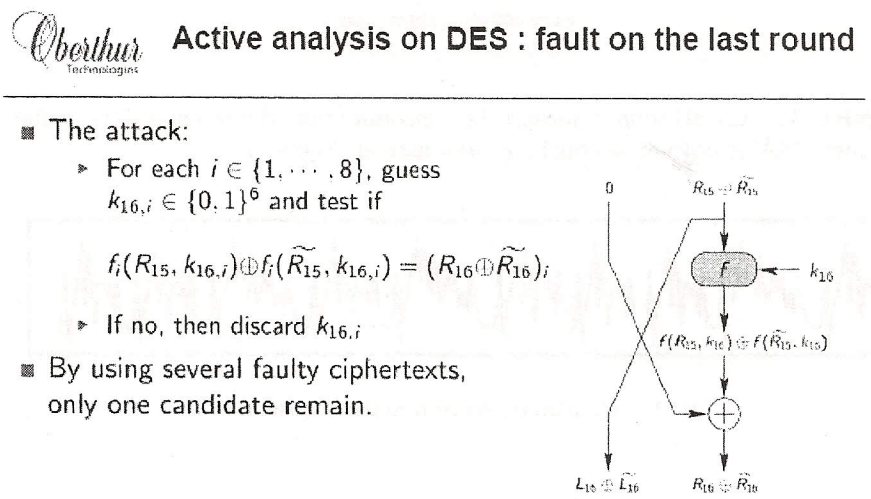
1. Quelles variables temporaires de l'AES peuvent être attaquées par analyse différentielle en faisant une hypothèse sur 8 bits de clé ?
2. Pourquoi voit-on apparaître plusieurs pics lorsque l'on fait de la CPA en prenant comme fonction de sélection la sortie de la transformation *SubBytes* lors du premier tour de l'AES ?
3. En supposant que la transformation *SubBytes* est toujours effectuée avant *ShiftRows*, observe-t-on le même phénomène lorsque l'on attaque l'entrée de la transformation *SubBytes* du dernier tour ?

Exercice 3. Attaque par injection de fautes sur le DES.

Un développeur choisie de vérifier l'intégrité de l'exécution d'un chiffrement DES en déchiffrant le chiffré obtenu et en comparant la sortie correspondante avec le texte clair d'origine.

En supposant qu'un attaquant puisse faire une double faute, où chaque faute

flip la valeur d'un bit dont la position peut être choisie, est-il possible d'obtenir des chiffrés erronés permettant de mener une attaque DFA tel que présentée page 70 du cours et reproduit ci-dessous ?



Exercice 4.

1. Une implémentation du RSA sur une carte à puce utilise l'algorithme *Square-and-Multiply Always* pour effectuer l'exponentiation modulaire de la signature RSA. Cet algorithme est présenté au transparent 33 du cours et reproduit ci-dessous.

```
Inputs : c, d=(d_{n-1}...d_0)_2
Output : m
 temp[0] ← 1
 for (i=n-1; i≥0; i--)
 {
     temp[0] ← temp[0]²
     temp[1] ← temp[0].c
     temp[0] ← temp[di]
 }
 Return temp[0]
```

   (a) Quel est le surcoût de cet algorithme en moyenne par rapport à un *Square-and-Multiply* classique si nous supposons qu'une élévation au carré prend le même temps qu'une multiplication ?

(b) Quel est son intérêt ?

2. Étant donnée la consommation de cette carte lors d'une signature RSA, supposons qu'un attaquant soit capable de détecter si $A = B$ en observant les calculs de $A^2 \bmod N$ et $B^2 \bmod N$, où N désigne le module RSA correspondant.
   (a) Examiner les valeurs prises par $(N-1)^i \bmod N$ selon la parité de i.
   (b) A partir de l'algorithme page 33 et en supposant que l'on signe le message $N-1$, calculer la valeur du résultat temporaire $temp[0]$ à la fin d'un tour de boucle traitant $d_i = 1$ et à la fin d'un tour de boucle traitant $d_i = 0$.
   (c) En déduire une attaque à message choisi permettant à l'attaquant de retrouver l'exposant secret avec la courbe de consommation d'une seule exécution d'une exponentiation utilisant le *Square-and-Multiply Always*.
3. Pour parer aux attaques par injection de faute, le développeur protège l'intégrité de l'exécution de cette signature RSA en vérifiant la signature avec l'exposant public : $S^e \bmod N = m$. Est-ce suffisant pour contrer les attaques par injection de faute ?