Examen – Attaques sur carte à puce

Durée : 54mn

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.

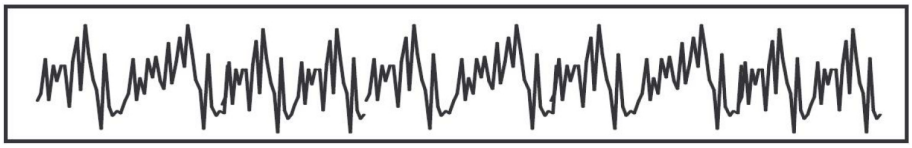


FIGURE 1 – Courbe de consommation observée

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.

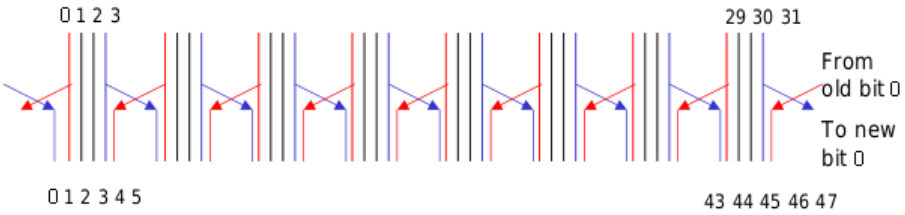
Quelles variables temporaires de l'AES peuvent être attaquées par analyse différentielle en faisant une hypothèse sur 16 bits de clé si l'attaquant dispose uniquement de la valeur du texte chiffré ? La fin de l'AES est schématisée dans le transparent 84 du cours.

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

En cours nous avons vu la description détaillée du DES (transparents 65 à 67). En particulier, la permutation expensive E est représentée par la table suivante :

```
E={31, 0, 1, 2, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 9,10,11,12,11,12,13,14,15,16,
   15,16,17,18,19,20,19,20,21,22,23,24,23,24,25,26,27,28,27,28,29,30,31, 0};
```

qui se schématise par :



Nous supposons qu'un attaquant peut perturber un seul bit de la partie droite de l'entrée du dernier tour du DES. Cependant la position du bit perturbé est toujours la même à chaque exécution du DES.
Combien de bits de la clé pourra-t-il retrouver au maximum grâce à une analyse DFA ?

Exercice 4.

1. Une implémentation du RSA sur une carte 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.
   (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 ?
   (c) Donner l'équivalent de l'algorithme *Square-and-Multiply Always* pour la multiplication scalaire des courbes elliptiques.
2. Le développeur remarque le surcoût de cette méthode. Il décide donc d'utiliser la méthode suivante dite *Atomique* :

Algorithme 1 Signature RSA utilisant une exponentiation atomique

---

ENTRÉES: Le message m, l'exposant privé $d = (d_{n-1}, \ldots, d_0)_2$ et le modulus N
SORTIE: La signature $S = m^d \bmod N$

---

1. $R_0 \leftarrow 1$
2. $R_1 \leftarrow m$
3. $i \leftarrow n - 1$
4. $k \leftarrow 0$
5. **while** $(i \geq 0)$ **do**
6. $\quad R_0 \leftarrow R_0 \cdot R_k \bmod N$
7. $\quad k \leftarrow k \oplus d_i$
8. $\quad i \leftarrow i - (k \oplus 1)$
9. **return** R_0

(a) Expliquer pourquoi l'algorithme 1 retourne $m^d \bmod N$.

(b) Est-ce que l'algorithme 1 est plus efficace que la méthode du *Square-and-Multiply Always* si le coût d'une multiplication est le même que celui d'une élévation au carré ? Donner l'éventuel gain/perte moyen comparé à cet algorithme.

(c) Est-ce que l'algorithme 1 est résistant à l'analyse simple observant les événements qui dépendent de la valeur de la clé ?