Introduction à la sécurité

Chapitre 5

Les attaques physiques
Canaux auxiliaires
Injections de fautes

Pablo Rauzy <pr@up8.edu>
pablo.rauzy.name/teaching/is

Les attaques physiques

Attaques physiques

- Un algorithme cryptographique peut être vu de deux façons :
  - d'un côté, c'est un objet mathématique abstrait,
  - de l'autre, c'est un code logiciel qui va finir par être exécuté sur du matériel.
- Le premier point de vue correspond à celui de la cryptanalyse classique.
- Le second correspond à celui de la sécurité physique.
- Les attaques physiques tirent partie des caractéristiques spécifiques des implémentations pour retrouver les paramètres secrets utilisés pendant le calcul.
- Ces attaques sont donc moins générales que celles de la cryptanalyse classique, mais elles sont aussi beaucoup plus puissantes.

Catégorisation haut niveau

- Il existe de nombreux types d'attaques physiques.
- À haut niveau, on peut déjà les classer selon deux axes :
  - invasives ou non-invasives : faut-il ouvrir ou casser en partie l'implémentation, ou au contraire n'exploiter que des informations naturellement (bien que non-intentionnellement) émises ?
  - active ou passive : l'attaque agit-elle sur l'implémentation ou se contente-t-elle de l'observer ?

Canaux auxiliaires

Les sources de fuites

- Les attaques passives exploitent des grandeurs physiques observables durant l'exécution du code, qui dépendent des données sensibles.
- Ces grandeurs peuvent être
  - le temps,
  - la consommation de courant,
  - des émanations électromagnétiques,
  - la chaleur,
  - le bruit,
  - ...

Attaques temporelles

- Si on ne fait pas attention, le temps d'exécution de la plupart des algorithmes dépend des données.
- Par exemple, pour l'exponentiation modulaire $B^E \bmod M$:

```
1 r := 1
2 b := b % m
3 tant que e != 0:
4     si e & 1 = 1 alors:
5         r := (r * b) % m
6     fin
7     b := (b * b) % m
8     e := e >> 1
9 fin
```

- Ce genre d'attaque peut fonctionner à travers le réseau.

- La contre-mesure est évidente : rendre l'exécution des programmes constante en temps.
- C'est par défaut le cas de la plupart des algorithmes de chiffrement symmétrique par exemple.
- Mais c'est parfois plus subtil qu'il n'y parait :
  - accès mémoire,
  - cache des processeurs.
- De fait, on sait depuis 2005 que AES peut-être vulnérable à ce genre d'attaque.

Analyse de consommation de courant

- Ces attaques sont très puissantes.
- La raison de cela est qu'on arrive très bien à modéliser la consommation de courant, et qu'elle est fortement corrélée aux données.
- En pratique, les mesures de consommations sont très proportionnelles
  - au poids de Hamming des valeurs (nombre de bits à 1),
  - à la distance de Hamming entre les valeurs qui se succèdent dans un même bus ou registre (nombre de bitflips).
- Les attaques par analyse d'émanations électromagnétiques correspondent à la même chose (c'est en fait l'activité électrique qui produit ces émanations).

- Il existe différentes formes d'attaques.
- Les deux principales sont
  - la SPA, pour simple power analysis, et
  - la DPA, pour differential power analysis.

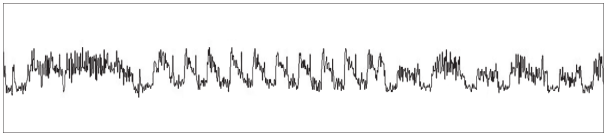
- Monter une attaque par analyse de consommation de courant nécessite plusieurs appareils.
- Au minimum, il est nécessaire d'avoir :
  - la cible (smartcard, FPGA, etc.) et une "board" permettant de l'utiliser.
  - une sonde (antenne EM, résistance, etc.),
  - un outil d'acquisition de trace de consommation (par exemple un oscilloscope qui peut prendre au moins 1 Gsample/s, et sensible au µA), et
  - un ordinateur pour analyser les traces de consommation.
- Avec autant d'appareil, le gros défi est de minimiser le bruit et/ou de réussir à isoler l'information pertinente dans les traces.

Analyse simple de consommation

- Le principe d'une SPA est de ne regarder qu'une seule trace de consommation, correspondant à une seule exécution du système cible.
- Cela peut déjà donner énormément d'informations :

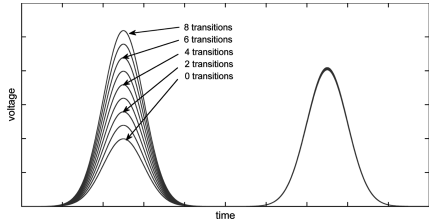
Analyse simple de consommation

- Le principe d'une SPA est de ne regarder qu'une seule trace de consommation, correspondant à une seule exécution du système cible.
- Cela peut déjà donner énormément d'informations :
  - par exemple sur le même code que pour l'attaque temporelle, on voit les activités différentes à chaque tour en fonction de la valeur des bits de l'exposant (1 vs 2 activités, voire 2 activités différentes si le `square` est optimisé),
  - on peut aussi identifier l'algorithme (par exemple en voyant le nombre de tours),
- Exemple de trace de consommation lors de l'exécution d'un AES 128 :



Analyse différentielle de consommation

- Le principe de la DPA est d'utiliser plusieurs (parfois des centaines de milliers !) traces de consommation.
- Des outils statistiques permettent alors d'exploiter les dépendances entre les données.
- Typiquement, avec les consommations moyennes toujours à un même point dans le temps on peut évaluer le nombre de bitflips à cette étape du calcul en utilisant le modèle de fuite dont on a déjà parlé (la distance de Hamming).



- Il est aussi possible de déjouer certaines contre-mesures qui rajoutent du bruit grâce à certaines méthodes statistiques.

Autres techniques

- Il existe bien d'autres techniques d'analyse de consommation, dont certaines encore plus poussées :
  - attaque par templates,
  - ASCA (pour Algebraic Side-Channel Attack),
  - CPA (pour Correlation Power Analysis),
  - ...

Contre-mesures

- Contre ce type d'attaques, il existe des contre-mesures aux niveaux matériel et logiciel.
- Les plus efficaces sont bien sûr celles qui mêlent les deux : des contre-mesures logicielles supportées par du matériel spécialisé.
- En tant qu'informaticien·nes, nous allons nous intéresser principalement aux contre-mesures logicielles.
- Il y a deux familles de contre-mesures :
  - les contre-mesures palliatives, et
  - les contre-mesures curatives.

Contre-mesures palliatives

- Les contre-mesures palliatives tentent d'utiliser de l'aléatoire pour ajouter du bruit dans les fuites d'information et les rendre inexploitables.
- Cependant, cela est fait sans vrai fondement théorique et en dehors d'un cadre formel.

Contre-mesures curatives

- Les contre-mesures curatives tentent de faire disparaître complètement l'information pertinente dans les fuites pour les rendre inexploitables.
- Pour cela elles s'appuient sur une formalisation des objectifs de sécurité.
- Il existe essentiellement deux telles stratégies :
  - le masquage consiste à rendre la fuite aussi décorrélée que possible des données sensibles,
  - l'équilibrage consiste à rendre la fuite constante, c'est à dire indépendante des données sensibles.

- L'idée de masquage et de mixer les données sensibles avec des nombres aléatoires pendant qu'on les manipule, de façon à en décorréler la fuite.
- Les avantages de cette techniques sont :
  - son indépendance vis-à-vis du matériel,
  - l'existence de systèmes prouvés.
- Cependant elle a aussi des inconvénients :
  - la possibilité d'attaques assez puissantes pour passer outre le masquage,
  - sa forte demande de nombres aléatoires, très coûteux à générer et pouvant se révéler être une faille supplémentaire.

Équilibrage

- Le but de l'équilibrage est de rendre la fuite constante, c'est à dire de la rendre totalement indépendante des données sensibles.
- Pour cela, il y a besoin d'une collaboration de la part du matériel : il faut que celui-ci fournissent deux ressources qui soient indistinguables du point de vue des canaux auxiliaires, c'est à dire qu'elles doivent fuir de manière identique.
- Cela peut paraître étonnant mais ça n'est pas du tout évident !
- Ces deux ressources vont être utilisées dans ce qu'on appelle un protocole double rail.

- La contre-mesure DPL (Dual-rail with Precharge Logic) consiste à faire tous les calculs sur une représentation redondante : chaque bit y est représenté par une paire $(y_{\text{False}}, y_{\text{True}})$.

- La paire de bits est utilisées en respectant un protocole en deux phases :
  - la précharge, lors de laquelle les deux bits de la paires sont remis à zéro : $(y_{\text{False}}, y_{\text{True}}) = (0, 0)$;
  - l'évaluation, lors de laquelle la paire $(y_{\text{False}}, y_{\text{True}})$ est mise à $(1, 0)$ si le bit logique y vaut 0 ou à $(0, 1)$ si le bit logique y vaut 1.

→ Pour étudier la mise en œuvre de cette contre-mesure au niveau logiciel :
Formally Proved Security of Assembly Code Against Power Analysis: A Case Study on Balanced Logic
Pablo Rauzy and Sylvain Guilley and Zakaria Najm, PROOFS 2014.

Injections de fautes

Attaques par injection de fautes

- Le principe d'une attaque par injection de faute est d'induire une erreur dans le calcul pendant celui-ci, par un moyen physique.
- Ce moyen peut être
  - un pulse électromagnétique / un laser visant le système,
  - une variation courte dans l'alimentation électrique du système,
  - ...
- Cela peut provoquer
  - un saut d'instructions,
  - la mise à une valeur aléatoire d'une variable intermédiaire du calcul,
  - la mise à zéro d'une variable intermédiaire du calcul.
- Les techniques de visées en temps et en localisation sont de plus en plus puissantes et précises.

Exploitation des fautes

- Les fautes peuvent être utilisées de différentes façons.
- Si on est très précis, on peut contourner un test de contrôle d'accès en sautant des instructions ou en changeant une valeur interprétée comme un booléen.

Exploitation des fautes

- Les fautes peuvent être utilisées de différentes façons.
- Si on est très précis, on peut contourner un test de contrôle d'accès en sautant des instructions ou en changeant une valeur interprétée comme un booléen.
- Dans le cadre d'une attaque cryptographique, où l'on cherche à retrouver la clef secrète, c'est différent : l'attaquant récupère le résultat fauté du calcul par la sortie normale du système, et essaye d'exploiter ce résultat.

RSA (rappel)

- RSA est un algorithme de cryptographie asymétrique dont la sécurité repose sur la difficulté du calcul des facteurs premiers de grands nombres
- RSA peut servir au chiffrement où à la signature de message.
- Les deux opérations sont similaires donc on va se concentrer aujourd'hui sur la signature.

Définition formelle

- Soit $N = p \cdot q$ le module de notre RSA, avec p et q deux grands nombres premiers.
- Soient e et d tels que $d \cdot e \equiv 1 \mod \varphi(N)$
  - (N, e) est la clef publique,
  - (N, d) est la clef privée.
  - retrouver d à partir de la clef publique est compliqué car il faut calculer son inverse modulo $\varphi(N) = (p-1)(q-1)$ ce qui suppose de connaître p et q.

Définition formelle

- Soit $N = p \cdot q$ le module de notre RSA, avec p et q deux grands nombres premiers.
- Soient e et d tels que $d \cdot e \equiv 1 \mod \varphi(N)$
  - (N, e) est la clef publique,
  - (N, d) est la clef privée.
  - retrouver d à partir de la clef publique est compliqué car il faut calculer son inverse modulo $\varphi(N) = (p-1)(q-1)$ ce qui suppose de connaître p et q.
- Soit m notre message.
- Alors $s \equiv m^d \mod N$ est la signature du message m.
- Et $m \equiv s^e \mod N$ permet la vérification de la signature.

Exponentiation modulaire (rappel)

- On avait déjà vu l'algorithme square-and-multiply pour effectuer une exponentiation modulaire rapide :

```
1 s := 1
2 m := m % N
3 tant que d != 0:
4     si d & 1 = 1 alors:
5         s := (s * m) % N
6     fin
7     m := (m * m) % N
8     d := d >> 1
9 fin
```

Exponentiation modulaire (rappel)

- On avait déjà vu l'algorithme square-and-multiply pour effectuer une exponentiation modulaire rapide :

```
1 s := 1
2 m := m % N
3 tant que d != 0:
4     si d & 1 = 1 alors:
5         s := (s * m) % N
6     fin
7     m := (m * m) % N
8     d := d >> 1
9 fin
```

- Pour se protéger des attaques par canaux auxiliaires qu'on a vu ensemble, on pourrait proposer ceci :

```
1 s := 1
2 m := m % N
3 tant que d != 0:
4     t := (s * m) % N
5     s := s * (1 - (d & 1)) + t * (d & 1)
6     m := (m * m) % N
7     d := d >> 1
8 fin
```

Exponentiation modulaire (rappel)

- On avait déjà vu l'algorithme square-and-multiply pour effectuer une exponentiation modulaire rapide :

```
1 s := 1
2 m := m % N
3 tant que d != 0:
4     si d & 1 = 1 alors:
5         s := (s * m) % N
6     fin
7     m := (m * m) % N
8     d := d >> 1
9 fin
```

- Pour se protéger des attaques par canaux auxiliaires qu'on a vu ensemble, on pourrait proposer ceci :

```
1 s := 1
2 m := m % N
3 tant que d != 0:
4     t := (s * m) % N # faute ici
5     s := s * (1 - (d & 1)) + t * (d & 1)
6     m := (m * m) % N
7     d := d >> 1
8 fin
```

Exponentiation modulaire (rappel)

- On avait déjà vu l'algorithme square-and-multiply pour effectuer une exponentiation modulaire rapide :

```
1 s := 1
2 m := m % N
3 tant que d != 0:
4     si d & 1 = 1 alors:
5         s := (s * m) % N
6     fin
7     m := (m * m) % N
8     d := d >> 1
9 fin
```

- Pour se protéger des attaques par canaux auxiliaires qu'on a vu ensemble, on pourrait proposer ceci :

```
1 s := 1
2 m := m % N
3 tant que d != 0:
4     t := (s * m) % N # faute ici = safe-error attack
5     s := s * (1 - (d & 1)) + t * (d & 1)
6     m := (m * m) % N
7     d := d >> 1
8 fin
```

CRT-RSA

- En pratique, dans les systèmes embarqués du type smartcard, les contraintes de ressources sont telles qu'on utilisent une variante optimisée de RSA : CRT-RSA.
- CRT-RSA permet de gagner un facteur 4 en vitesse :
  - il remplace l'exponentiation modulaire par deux exponentiations modulaires avec des exposants deux fois plus petits,
  - ces calculs vont 8 fois plus vite,
  - après il ne reste qu'une recombinaison rapide à effectuer.

Définition formelle

- Soit $N = p \cdot q$ le module de notre RSA, avec p et q deux grands nombres premiers.
- Soient e et d tels que $d \cdot e \equiv 1 \mod \varphi(N)$ et
  - $d_p \doteq d \bmod (p-1)$,
  - $d_q \doteq d \bmod (q-1)$,
  - $i_q \doteq q^{-1} \bmod p$,
  - (N, e) est la clef publique,
  - (p, q, d_p, d_q, i_q) est la clef privée.
- Soit m notre message.
- Alors sa signature se calcule comme suit :
  - $s_p = m^{d_p} \bmod p$,
  - $s_q = m^{d_q} \bmod q$,
  - $s = s_q + q \cdot (i_q \cdot (s_p - s_q) \bmod p)$.
- Et $m \equiv s^e \mod N$ permet toujours la vérification de la signature.

BellCoRe

- L'attaque BellCoRe (de Bell Communication Research) consiste à retrouver p et q en injectant une faute à peu prêt n'importe où dans le calcul.
- C'est la première attaque par injection de faute (1997).
- Si s_p (resp. s_q) est fauté comme $\widehat{s_p}$ (resp. $\widehat{s_q}$), l'attaquant
  - récupère une signature fautée $\hat{s}$,
  - peut retrouver q (resp. p) en calculant $\mathrm{pgcd}(N, s - \hat{s})$.

- Pour tout entier x, $\text{pgcd}(N, x)$ ne peut prendre que 4 valeurs :
  - 1, si N et x sont premiers entre eux,
  - p, si x est un multiple de p,
  - q, si x est un multiple de q,
  - N, si x est un multiple de p et de q, i.e., de N.

- Si s_p est fauté (i.e., remplacé par $\widehat{s_p} \neq s_p$) :
  - $s - \hat{s} = q \cdot \left((i_q \cdot (s_p - s_q) \mod p) - (i_q \cdot (\widehat{s_p} - s_q) \mod p)\right)$,

  $\Rightarrow \text{pgcd}(N, s - \hat{s}) = q$.

- Si s_q est fauté (i.e., remplacé par $\widehat{s_q} \neq s_q$):
  - $s - \hat{s} \equiv (s_q - \widehat{s_q}) - (q \mod p) \cdot i_q \cdot (s_q - \widehat{s_q}) \equiv 0 \mod p$,

  $\Rightarrow \text{pgcd}(N, s - \hat{s}) = p$.

Comment ça marche ?

- Pour tout entier x, $\text{pgcd}(N, x)$ ne peut prendre que 4 valeurs :
  - 1, si N et x sont premiers entre eux,
  - p, si x est un multiple de p,
  - q, si x est un multiple de q,
  - N, si x est un multiple de p et de q, i.e., de N.

- Si s_p est fauté (i.e., remplacé par $\widehat{s_p} \neq s_p$) :
  - $s - \hat{s} = q \cdot \left((i_q \cdot (s_p - s_q) \mod p) - (i_q \cdot (\widehat{s_p} - s_q) \mod p)\right)$,

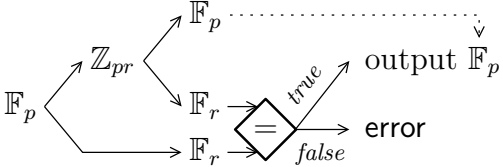
  $\Rightarrow \text{pgcd}(N, s - \hat{s}) = q$.

- Si s_q est fauté (i.e., remplacé par $\widehat{s_q} \neq s_q$):
  - $s - \hat{s} \equiv (s_q - \widehat{s_q}) - (q \mod p) \cdot i_q \cdot (s_q - \widehat{s_q}) \equiv 0 \mod p$,

  $\Rightarrow \text{pgcd}(N, s - \hat{s}) = p$.

Contre-mesures : l'extension modulaire

- Les contre-mesures contre l'attaque BellCoRe sont légions :
  - ∼20 articles depuis 1999,
  - aussi bien du côté académique qu'industriel.
- La plupart sont basées sur une même idée : l'extension modulaire.



→ Pour étudier les mises en œuvre de ce type de contre-mesure :
Countermeasures Against High-Order Fault-Injection Attacks on CRT-RSA
Pablo Rauzy and Sylvain Guilley, FDTC 2014.