Parcours École Ingénieur – 2ème année

PRJ 1401 – Rapport final

Attaque par cannaux auxiliaires : Cryptanalyse du système RSA

14 mai 2024

Par Etienne RICHARD, Matthieu CIZDZIEL et Mathis MOGUEDET

Table des matières

Introduction

Introduction...

1 Définitions et notations

1.1 Définitions

BigInt : Grand entier, généralement représenté sur plusieurs mots mémoire.

Chunk : Un groupe de bits qui est utilisé pour stocker une partie d'un grand entier.

Débordement : Un événement se produisant lorsqu'un calcul dépasse les capacités de stockage d'un type de données, nécessitant une gestion spéciale.

1.2 Notations

n : désigne le modulo utilisé pour les opérations modulaires

e : désigne l'exposant public

d : désigne l'exposant privé

m : désigne le message

c : désigne le chiffré

2 Contexte du projet

2.1 Introduction

En terme de sécurité informatique les méthodes de chiffrement sont très importantes notamment dans le domaine des télécommunication. La cryptographie est donc une notion très importante dans la cybersécurité actuelle. Les 3 principes de la cryptographie sont la confidentialité, l'authenticité et l'intégrité. La confidentialité assure que le contenu d'un message chiffré ne peut être lu que par son destinataire. L'authenticité assure l'origine du message, c'est à dire l'identité du messager. Enfin l'intégrité assure la non-modification d'un message. Dans notre cas nous mettrons en oeuvre des moyens pour s'attaquer à la confidentialité d'un protocole cryptographique. Ainsi nous allons vous présenter un algorithme connu pour sa robustesse et utilisé dans le monde entier, le protocole RSA.

2.2 Généralités sur le RSA

Nous ne pouvons pas commencer la présentation du projet sans présenter le protocole RSA. Le protocole RSA utilise une clé publique pour chiffrer le message et une clé privé pour le déchiffrer. Les clés publique et privé n'étant pas les mêmes, on parle de chiffrement asymétrique. La robustesse du RSA repose dans la difficulté de factoriser un produit de deux grands nombres premiers. Sans rentrer dans le détail de sa robustesse nous allons vous expliquer son fonctionnement :
Génération des clés Soit p et q deux nombres premiers distincts. On pose alors $n = pq$ et $\phi(n) = (p-1)(q-1)$. On choisit ensuite un nombre e premier avec $\phi(n)$ et strictement inférieur à $\phi(n)$ On peut enfin calculer d l'inverse modulaire de e modulo $\phi(n)$. Ainsi le couple (n,e) constitue la clé publique et d la clé privée.
Chiffrement Soit un message représenté par un entier naturel M strictement inférieur à n et C le message chiffré. Nous avons la relation suivante :

$$C \equiv M^e \pmod{n}$$

Ainsi nous remarquons que la base du chiffrement et du déchiffrement repose sur une exponentiation modulaire. Le problème de l'exponentiation utilisé par le RSA est qu'elle ne peut pas être naïve, c'est à dire qu'on ne peut pas faire $x^n = x * x * x... * x$ n fois. En effet, le RSA utilise des nombres de 2048 bits ce qui rend impossible l'utilisation d'un algorithme naïf. Ainsi nous allons vous présenter l'algorithme d'exponentiation rapide.

2.3 L'exponentiation rapide

L'algorithme d'exponentiation rapide permet de réduire drastiquement le temps de calcul en basant le calcul sur la représentation binaire de l'exposant. Ainsi nous pouvons analyser l'algorithme suivant : Nous pouvons remarquer que l'algorithme fonctionne de manière séquentielle. Si l'exposant n est impair on effectue deux multiplication et une division. Si n pair on effectue une multiplication et une division. On appelle également cet algorithme "Square and Multiply" et on comprend aisément pourquoi. Il est important de noter qu'en binaire un nombre pair possède un bit de poids faible égal à 0 tandis qu'un nombre impaire possède un bit de poids faible égal à 1. Ainsi en parcourant l'exposant du bit de poids fort au bit de poids faible on utilise des opérations différentes en fonction du bit rencontré. De ce fait l'algorithme d'exponentiation rapide est de complexité $log_2(n)$ tandis que l'algorithme d'exponentiation naïve est de complexité n ce qui représente une sacrée différence sur des grand nombres. Nous le verrons par la suite mais cette implémentation du RSA avec l'algorithme d'exponentiation est sensible aux attaques par canaux auxiliaires.

2.4 Attaques par canaux auxiliaire

Il est désormais temps de vous introduire au concept d'attaque par canaux auxiliaires. Selon Wikipedia une attaque par canaux auxiliaires est une : "Attaque informatique qui, sans remettre en cause la robustesse théorique des méthodes et procédures de sécurité, recherche et exploite des failles dans leur implémentation, logicielle ou matérielle." En reprenant ce que nous vous avons présenter précédemment nous ne remettons pas en compte la robustesse du protocole RSA mais son implémentation utilisant l'algorithme d'exponentiation rapide. Ainsi il existe une multitude d'attaques par canaux auxiliaires comme les attaques temporelles basées sur le temps mis par l'algorithme pour effectuer certaines opération, les attaques pas sondage qui consiste à analyser un circuit en y posant une sonde ou encore les attaques par consommation de courant. Dans notre cas nous allons nous intéresser aux attaques par consommation de courant.

3 Implémentation du RSA

Après vous avoir introduit le contexte du projet, nous allons maintenant vous expliquer comment nous avons fait pour implémenter le crypto-système RSA.

3.1 Implémentation sur 32 bits

Comme nous l'avons vu dans la section précédente, pour implémenter le RSA, nous avons besoin des algorithmes suivant :

— Exponentiation modulaire (déterminer a tel que $a \equiv x^n \bmod n$) ;
— Inverse modulaire (étant donné b, trouver a tel que $ab \equiv 1 \bmod n$).

Voici le pseudo code de ces deux algorithmes :

Algorithm 1: Exponentiation Modulaire

```
Input: base, exponent, modulus
Output: result
result ← 1;
base ← base mod modulus;
while exponent > 0 do
    if exponent est impair then
        result ← (result × base) mod modulus;
    end
    exponent ← exponent ÷ 2;
    base ← (base × base) mod modulus;
end
return result;
```

Algorithm 2: Inverse Modulaire

Input: a, m
Output: inverse modulaire
$m0 \leftarrow m$;
$y \leftarrow 0$;
$x \leftarrow 1$;
if $m = 1$ **then**
  return 0;
end
while $a > 1$ **do**
  $q \leftarrow a \div m$;
  $t \leftarrow m$;
  $m \leftarrow a \mod m$;
  $a \leftarrow t$;
  $t \leftarrow y$;
  $y \leftarrow x - q \times y$;
  $x \leftarrow t$;
end
if $x < 0$ **then**
  $x \leftarrow x + m0$;
end
return x;

Nous pouvons voir que pour le premier algorithme, celui d'exponentiation rapide modulaire, nous utilisons un algorithme efficace contrairement à l'algorithme naïf qui consisterait a faire : $\underbrace{x \times x \times \ldots \times x}_{\text{n fois}}$. Afin de mettre en valeur la différence de complexité entre ces deux version, vous pouvez retrouver Figure 1 la graphe représentant le nombre d'opération requis en fonction de l'exposant.

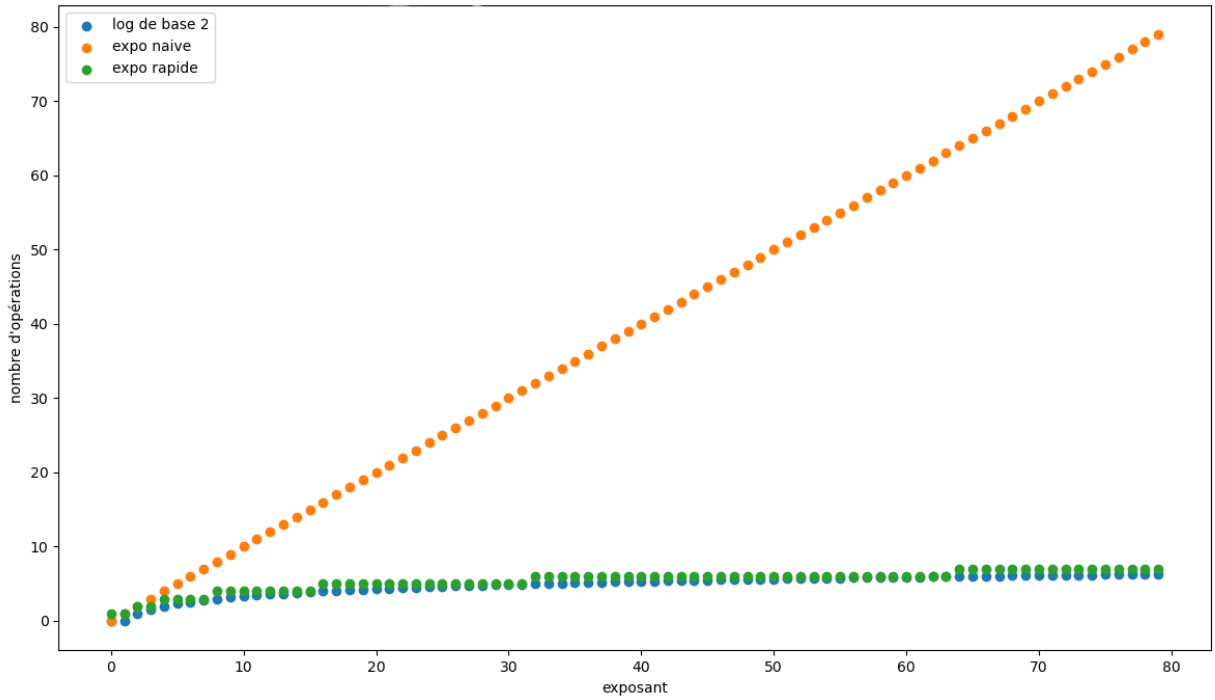

FIGURE 1 – Comparaison des complexités des deux algorithmes d'exponentiation

Il est aussi important de relever que la présence de l'algorithme d'exponentiation rapide n'est pas un hasard, c'est grâce à lui que le système est faillible. En effet, si il rencontre un 1 dans la représentation binaire du nombre il fait 2 calculs, sinon, il en fait 1 seul. Grâce à ça, il a une fuite de la clé qui s'opèrera sur la trace de consommation électrique.
Notre implémentation en C de ces deux algorithmes se trouve dans l'archive final du projet. Une fois ces primitives implémentées, il a été facile d'implémenter la génération de clé, le chiffrement et le déchiffrement, vous pouvez également retrouver ces implémentations en C dans l'archive final du projet.

3.2 Objectif 2048 bits

D'après les recommandation de l'ANSSI [2], un RSA implémenté avec les types de bases du C (32 bits), n'est pas du tout sécurisé. En effet, nous pouvons retrouver page 19 les règles suivantes :

1. La taille minimale du module est de 2048 bits, pour une utilisation ne devant pas dépasser la fin de l'année 2030.
2. Pour une utilisation à partir de l'année 2031, la taille minimale du module est de 3072 bits.
3. Les exposants secrets doivent être de même taille que le module.

4. Pour les applications de chiffrement, les exposants publics doivent être strictement supérieurs à $2^{16} = 65536$.

Afin d'attaquer une version du RSA qui pourrait être utilisé dans un cadre réel, nous allons donc devoir l'implémenter avec un module, notée n précédemment, de 2048 bits ! Or, comme vu précédemment nous allons utiliser une machine en 32 bits, ce qui signifie que les types de bases sur lesquels les opérations élémentaires sont implémentées, ne dépasse pas 32 bits. Nous allons devoir créer et implémenter des opérations sur les grands nombres, nous appellerons BigInt ces derniers dans le reste du rapport car c'est l'usage.

3.3 Implémentation du big int

Pour représenter un grand entier dans un système embarqué 32 bits, nous utilisons un découpage en chunks de 16 bits. Imaginons que nous souhaitons un nombre de 64 bits. Nous divisons cet entier en mots de 16 bits. Ainsi, chaque chunk de 16 bits contient une partie du grand entier.

Prenons un exemple concret : supposons que notre grand entier soit `0xABCD1234567890EF`. Pour le découper en chunks de 16 bits, nous aurons :

— Chunk 1 : `0xABCD`
— Chunk 2 : `0x1234`
— Chunk 3 : `0x5678`
— Chunk 4 : `0x90EF`

Chaque chunk peut être représenté par un entier de 16 bits dans la mémoire du système embarqué. En utilisant cette méthode de découpage, nous pouvons stocker et manipuler des grands entiers même dans des systèmes avec des limitations de taille de données.

Si nous utilisons des sous-entiers de 16 bits et pas de 32 bits c'est parce que le 32 bits servira de type d'overflow, en effet lorsqu'on aura une sous opération qui dépassera la taille d'un chunk alors un entier de 32 bits permettra de résoudre ce problème.

Sachant que chaque chunk est sur 16 bits c'est comme si nous avons une représentation du nombre en base 2^{16}, nous n'avons "plus" qu'a considéré un chunk comme un symbole de la base 2^{16} et implémenté les opérations usuelles dessus.

Cette méthode fonctionne pour l'addition et la soustraction, mais pour les opérations d'inverse modulaire, de multiplication modulaire et de division, cela ne fonctionne plus et il faut utiliser des algorithmes plus poussés, nous nous sommes basé sur ceux du livre "Handbook of Applied Cryptography" [1].

3.4 Conclusion

4 Attaque

Conclusion

Introduction...

Références

[1] P. van Oorschot A. Menezes et S. Vanston. *Handbook of Applied Cryptography*. www.cacr.math.uwaterloo.ca/hac. 1996. URL : https://cacr.uwaterloo.ca/hac/about/chap14.pdf.

[2] ANSSI. *Guide des mécanismes cryptographiques*. Version 2.04 - 2020-01-01. Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques. 2021. URL : https://cyber.gouv.fr/publications/mecanismes-cryptographiques.