Survey: Lattice Reduction techniques to attack RSA

David Wong

March 2015

Résumé

RSA, carrying the names of **Ron Rivest**, **Adi Shamir** and **Leonard Adleman**, is one of the first practicable **public-key** cryptosystem. The algorithm was publicly described for the first time in **1977** and has been since the most used cryptosystem when it comes to asymmetric problems. For now more than **30 years**, Cryptanalists and Researchers have looked for ways to **attack RSA**.
In 1995, **Coppersmith** released a paper on how to attack RSA using **Lattices** and **Lattice reduction techniques** such as **LLL**. A few years later, **Howgrave-Graham** revisited Coppersmith's algorithm and made it easier to understand and apply.
Attacks based on Lattice reduction techniques caught up and several researches were done on the subject. Years after **Wiener** found that you could successfully break RSA if the private exponent was too small. In 2000, **Boneh** and **Durfee** found a better attack, still based on Lattice, that was simplified afterwards by a work from **Herrmann** and **Mayers**.
In the 1st and 2nd sections of this survey I will briefly explain how RSA and Lattice work. In section 3 we will see in what **model** the attacks are taking place. Section 4 will be an overview of the Coppersmith attack revisited by Howgrave-Graham. Section 5 will be an overview of the Boneh and Durfee attack revisited by Herrmann and May. Finally we will see **some applications** of those attacks.

1 RSA

Let's quickly recall **what is** and **how** RSA works :
RSA is an **asymmetric cryptosystem**. A generator algorithm derives two kind of keys : a **public key** and a **private key**, these can be swapped around thanks to the asymmetric property of RSA to allow us to use the system as an **encryption** system or as a **signature** system.

1.1 Generation

To use RSA we need a public key to encrypt and a private key to decrypt. We first generate the private key pair as follow :

We generate p, q **primes**. For security issues they should be around the **same size**. Those primes are the **core elements** of RSA. Knowing one of those allows us to compute the private key, thus allowing us to break the system. They can also be used to speed up calculations using the Chinese Remainder Theorem. We will see all about that later.
Knowing p and q we can then compute the **modulus** $N = p \times q$ which will be part of the public key as well as the private key. And you will see why.

Now comes the interesting part, we need to find a **public exponent** which will be used for **encryption**. For computation purpose a **Fermat prime** $(2^m + 1)$ is often used as public exponent (it makes things faster in **binary exponentiation**). In the case of a **signature scheme**, we would want the speed up to occur for the **private exponent** so we would use such a number as a private exponent and we would reverse the following steps.
Anyway, any kind of exponent can theoretically be chosen, as long as it is coprime with $\varphi(N)$ (The **Euler's Totient function**).

$$e \leftarrow \mathbb{Z}^*_{\varphi(N)}$$

if e is **coprime** with $\varphi(N)$ then it is part of the multiplicative group $(\mathbb{Z}/\varphi(N)\mathbb{Z})^*$ and thus **invertible** in $\mathbb{Z}/\varphi(N)\mathbb{Z}$.

Now it is pretty easy to find the private exponent d by inverting our public exponent e.

All of this is possible because we can easily compute $\varphi(N)$:

$$\varphi(N) = (p-1) \times (q-1)$$

And here resides the **security** of RSA. Imagine for a moment that we could easily factor N into p and q, then we would be able to invert the public exponent e. That's why we say that **the security of RSA is reduced to the Factorization Problem**.

Now let the **private key** be **(N, d)** with the addition of (p, q) if we need to speed up calculations. And let the **public key** be **(N, e)**.

1.2 Encryption/Decryption

To **encrypt** a message m, with $m < N$ we just do :

$$c = m^e \pmod N$$

And to **decrypt** :

$$m = c^d \pmod N$$

This works because the decryption step gives us :

$$c^d = (m^e)^d \pmod N$$

And e being d's inverse tells us that :

$$\begin{aligned} e &= d^{-1} \pmod{\varphi(N)} \\ \implies ed &= 1 \pmod{\varphi(N)} \\ \implies ed &= \varphi(N) + 1 \end{aligned}$$

Coupled with **Euler's Theorem** stating that if a and n are coprime then :

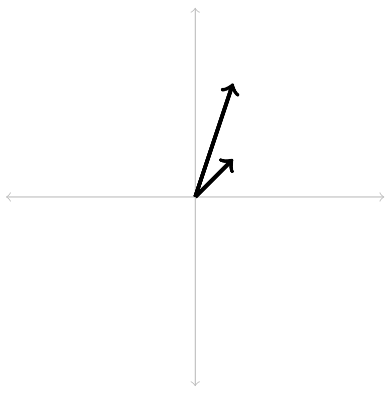
$$a^{\varphi(n)} = 1 \pmod n$$

Tells us that $m^{ed} = m \pmod N$

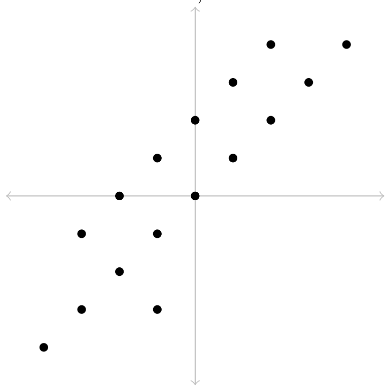
2 Lattice

2.1 Introduction

The attacks we will describe latter both make use of the **Lenstra–Lenstra–Lovász lattice basis reduction algorithm**. Hence it is necessary for us to understand what are Lattices and why is this **LLL** algorithm so useful.
Think about Lattices like **Vector Spaces**. Imagine a simple vector space of two vectors. You can add them together, multiply them by scalars (let's say numbers of $\mathbb{R}$) and it spans a vector space.



Now imagine that our vector space's **scalars are the integers**, taken in $\mathbb{Z}$. The space spanned by the vectors is now **discrete**. Meaning that for any point of this lattice there is a ball centered around that point, of radius different from zero, that contains only that point. Nothing else.



Lattice are interesting in cryptography because we seldom deal with real numbers and they bring us a lot of tools to deal with numbers.

2.2 basis

2.3 LLL

3 Attacks

4 Coppersmith

This survey is no replacement for the original papers of Coppersmith[1] and Howgrave-Graham[2]. If you want to get a real understanding of those techniques I also advise you to read a survey from May[3].

Références

[1] Don Coppersmith *Finding Small Solutions to Small Degree Polynomials*

[2] Nicholas Howgrave-Graham *Finding Small Roots of Univariate Modular Equations Revisited*

[3] Alexander May *Using LLL-Reduction for Solving RSA and Factorization Problems*