Cryptography and Number theory

othmane.rih

February 2024

1 Introduction

In an increasingly connected world, data protection has never been more important. This can range from a simple attack to leak video game trailers, to an attack on a hospital system in the middle of a war, to the theft of data from a dating site. In this course, we will take a look at two of the best-known security protocols and see which attacks are the most effective.

2 The RSA cryptosystem

2.1 Plain RSA

RSA is probably one of the most famous cryptosystems. Today, it remains one of the most secure under certain conditions, as we shall see. How does it work in concrete terms?

Suppose Alice and Bob are preparing a surprise gift for their friend Eve. They want to be able to exchange messages with each other without Eve being able to read them. Let us say that they use RSA. Well, first, they need to agree on the key that will be used for encryption. To do that, they need to choose what is called an *RSA modulus*, that is, a positive integer N of the form $N = pq$, where p and q are large prime numbers.

The message Alice wants to send will be converted to an integer m such that $m < N$. Alice chooses a *public exponent* e such that $(e, \phi(N)) = 1$. Here, ϕ is the Euler totient function. Alice sends to Bob $M = m^e \bmod N$.

Bob computes $d = e^{-1} \bmod \phi(N)$ and retrieves the original message by computing $M^d \bmod N$. An easy exercise is to show that $m = M^d \bmod N$.

Suppose now that Eve has heard that Alice and Bob are up to something. She doesn't like surprises and wants to know what Alice and Bob have prepared for her. Hence, she set out to decipher the messages exchanged between her friends.

In general, breaking the RSA cryptosystem can be reduced to solving number theoretical problems. One of them, and maybe the most famous one, is the factorisation problem.

2.2 Factorisation problem

Apart from being able to break the RSA cryptosystem, being able to factorise a number can be applied to other problems. For example, one can show that computing square roots in $\mathbb{Z}_n^*$ is equivalent to factorising n. Thus, we want to be able to solve this problem in an efficient way. To measure efficiency, we introduce the following function

$$L_x(t, \gamma) = e^{(\gamma + o(1))(\log(x))^t (\log(\log(x)))^{1-t}}$$

where $0 \leq t \leq 1$ and $x, \gamma \in \mathbb{R}_{\geq 0}$. Here, x is the complexity parameter and t is used to measure complexity. If $t = 0$, we say that the complexity is polynomial, and if $t = 1$, we say this exponential. In between, we say that the complexity is subexponential.

For now, the best algorithms that allow us to solve this problem do it in subexponential time; well, this is true if we are working with a classical computer. Thanks to the Shor algorithm, we can factorise an integer in polynomial time, if you manage to have a sufficiently powerful quantum computer. We will now see different algorithms.

2.3 Trial division

Well, we know that for $n \in \mathbb{N}_{>1}$, there exists $p \leq \sqrt{n}$ such that $p \mid n$. Hence, one dumb way to solve the factorisation problem is just to try every prime until $\sqrt{n}$ until we find a factor. What is the complexity of this algorithm ?

2.4 Squares to the rescue

The idea is quite simple : Suppose that we can write n as $n = a^2 - b^2 = (a-b)(a+b)$ for which $a - b > 1$, then we get a non-trivial factorisation. Practically, suppose that there exists x, y such that $x^2 = y^2 \bmod n$, then we can hope to find a non-trivial factor by computing $\gcd(x-y, n)$ or $\gcd(x+y, n)$. The question is: How do we find such x and y that yield a nontrivial divisor of n?

The method we are going to see is known as Dixon's method. This method tries to avoid solving $x^2 \equiv y^2 \pmod n$ directly, but solves it by combining (via linear algebra) some congruences mod n that are easier to generate. Let $B > 0$ be a real number. Let $P(B)$ be the set of all prime numbers $\leq B$ (the factor base), and let $m = \#P(B)$. The idea of the algorithm is to generate integers v, $1 \leq v \leq n$ such that

$$v^2 \equiv \prod_{p \in P(B)} p^{e_p(v)} \pmod n,$$

for some exponents $e_p(v)$, that is, such that $v^2 \pmod n$ is B-smooth. Once we have generated more than m such congruences, we can try to combine them (using linear algebra over $\mathbb{F}_2$ only) to produce a right-hand side of the congruence that is a perfect square. The phase where we generate the congruences will be called the relation generation phase, whereas the phase where we solve the linear system will be called the linear algebra phase.

Algorithm 1: Dixon's method

Data: An integer n, a bound B
Result: a factor of n
$P := \{p_1, p_2, \ldots, p_k\}$ be all primes $\leq B$
for $i = 1$ *to* $k+1$ **do**
  Choose $0 < z_i < N$ such that $z_i^2 \equiv N \pmod{z_i^2 \bmod N}$ is B-smooth;
  Let $a_i := \{a_{i1}, a_{i2}, \ldots, a_{ik}\}$ such that $z_i^2 \bmod N = \prod_{p_j \in P} p_j^{a_{ij}}$
end
Find non-empty $T \subseteq \{1, 2, \ldots, k+1\}$ such that $\sum_{i \in T} a_i \equiv \vec{0} \pmod 2$;
Let $x := \left(\prod_{i \in T} z_i\right) \bmod N$ Let $y := \left(\prod_{p_j \in P} p_j^{\left(\sum_{i \in T} a_{ij}\right)/2}\right) \bmod N$;
if $x \equiv \pm y \pmod N$ **then**
  Abort
end
return $\gcd(x+y, N)$ and $\gcd(x-y, N)$

2.5 Pollard's $p-1$ method

This time let's try to be smarter and try to see which result from number theory involving modular arithmetics we can use. One particular basic result that seems pretty interesting to us is Fermat's Little Theorem. This is how the algorithm will use it.

This algorithm will typically work for integers that have a factor p that is B-smooth. This means that all the prime divisors of p are smaller than B. If the algorithm fails, this means that we need to change the smoothness bound B. If $\gcd(x-1, n) = 1$, we need to choose a larger bound; if $\gcd(x-1, n) = n$, we need a smaller bound. The complexity of this algorithm is $\mathcal{O}\left(B \log(B) \log(n)^2\right)$.

We can also make this algorithm fancier (and more efficient) by using elliptic curves. First, let us define what an elliptic curve is.

Definition 2.1. Let K be a finite field, and $a, b \in K$. We define an elliptic curve as the set of points in K^2 that satisfy $y^2 = x^3 + ax + b$, plus the point at infinity $\mathcal{O}$. We denote this set by $E_{a,b}(K)$.

Algorithm 2: Pollard's $p-1$ method

Data: $B, n \geq 0$
Result: a factor of n
$i \leftarrow 1$;
$x \leftarrow 2$;
while $\gcd(x-1, n) \neq 1$ *and* $i \leq B$ **do**
  $x \leftarrow x^i$;
  $i \leftarrow i+1$;
end
if $\gcd(x-1, n) \in \{1, n\}$ **then**
  return *failure*
end
return $\gcd(x-1, n)$

This definition is quite basic and does not contain all the possible elliptic curves that exist. But it suffices for our purposes. An elliptic curve is not just a set, but it is also a group.

- The neutral element is the point at infinity.
- for $P = (x_P, y_P) \in E_{a,b}(K)$, we define the inverse as $-P = (x_P, -y_p)$.
- for $P, Q \in E_{a,b}(K)$ and $Q \neq -P$, we define $R = P + Q$ such that :

$$\lambda = \begin{cases} \frac{y_Q - y_P}{x_Q - x_P} & \text{if } x_P \neq x_Q \\ \frac{3x_P^2 + a}{2y_P} & \text{if } x_P = x_Q \end{cases}$$
$$x_R = \lambda^2 - x_P - x_Q$$
$$y_R = (x_P - x_R)\lambda - y_P$$

For the group operation, we assumed that $char(K) \notin \{2, 3\}$. If that is the case, the formula changes slightly.

Exercise : Try to come with an algorithm using points of elliptic curves. (One can replace the field K by a ring R, the operations do not change)

3 Diffie-Hellman Key Exchange

3.1 Diffie-Hellman Key Exchange protocol

One of the most basic symmetric cryptosystems is the Vernam cipher. Unlike RSA, where we have part of the key that is public, the Vernam cipher requires the key to be secret, and it should be very hard to retrieve it, since it was used to encrypt and decrypt the message. The problem is : How can two parties share a key in a safe manner ?

The DHKE is meant to answer this question. How does it work? Suppose that Alice and Bob (or whoever) want to agree on a key.

- First, we need a cyclic group of prime order q with generator g. We usually do that by taking a subgroup of $(\mathbb{Z}/p)^*$.
- Alice and Bob pick at random a and b, respectively, in $(\mathbb{Z}/q)\backslash\{0\}$ and send to each other $A = g^a$ and $B = g^b$. A and B are called public keys, while a and b are called secret keys.
- Alice computes B^a and Bob computes A^b. The shared secret key is $K = B^a = A^b = g^{ab}$.

Remark 3.1. One can generalise this for any kind of group, not necessarily subgroups of $\mathbb{Z}/n$. Can you produce a version of the protocol using elliptic curves ?

If Eve wants to be able to listen to Alice and Bob's conversation, the only possible way is to retrieve the key. The main strategy is to find the secret key of Alice or Bob from their public key. If we can do that, we will basically win. The problem we are trying to solve is called the discrete log problem.

3.2 The DL problem

We can rephrase the problem as follow :

Problem 1. *Let G a group of order q with generator g, operation $\cdot$ and $X \in G$. Find $x \in \mathbb{Z}/q$ such that $g^x = X$.*

This problem, as the factorisation problem, is among the foundations of today's cryptography. Unfortunately (or fortunately), we don't have for now an efficient algorithm that can solve this problem. Again, this is true only if you are working with a classical computer. If you manage to find a quantum computer, you can apply Shor's algorithm and solve this problem in polynomial time.

I will present now two algorithms that solve this problem in exponential time.

3.3 The Baby step-Giant step algorithm

The idea is quite simple. Suppose that we are in the setting of the discrete log problem and we have $X \in G$. We know that there exists some $x \in \mathbb{Z}/q$ such that $X = g^x$. Suppose now that we choose some $l \in \mathbb{Z}/q$. Then by Euclidean division, there exist $r \leq l - 1$ and $k \in \mathbb{Z}/q$ such that $x = kl + r$. The goal now is to find for which k and r it works. Unfortunately, there is no clever way to find them, we need to do that with some kind of brute force argument, using some precomputations.

Algorithm 3: Baby step-Giant step algorithm

Data: A group G with generator g,$X \in G$, a bound l
Result: the logarithm of X in base g
Precomputation for $i = 0, \ldots, \lfloor q/l \rfloor$ **do**
  insert $(g^{i\ell}, i)$ into a hash table
end
Algorithm for $j = 0, \ldots, l - 1$ **do**
  compute $z = Xg^{-j}$;
  if (z, i) *is in the hash table* **then**
    return $x = il + j$
  end
end