CPEN 400Q Lecture 12
Supplement: crash course on RSA

Friday 17 February 2023

Public-key cryptosystems

Two different types of cryptosystems:

- Symmetric: the key used to decode is the same as (or can easily be obtained from) the one used to encode
- Asymmetric / public-key: the key used to decode is different than the one used to encode

Both are used in modern infrastructure and each has its own advantages/disadvantages, attack vectors, and vulnerabilities.

RSA

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem.

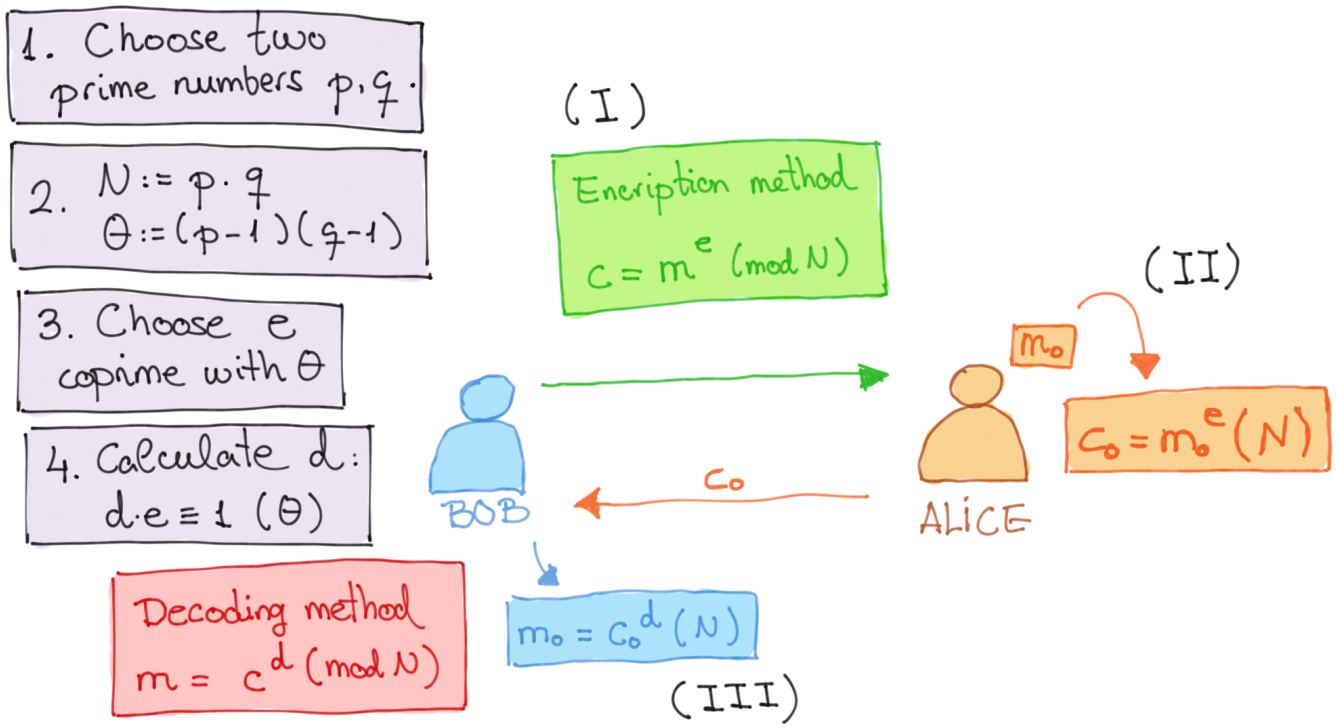

Image credit: Xanadu Quantum Codebook node S.5

RSA: brief review of number theory concepts

The math behind RSA involves modular arithmetic and a few other number-theoretic ideas:

Greatest common divisor (gcd): $gcd(a, b)$ is the largest integer factor that divides perfectly into both a and b

Co-prime: a and b are co-prime if $gcd(a, b) = 1$.

Modular inverse: Given a number a and modulus N, the inverse of a is the integer b such that $ab \equiv 1 \mod N$. This exists *only if a and N are co-prime*.

Fermat's little theorem: if p prime and a is an integer, then $a^p = a \mod p$. If a is not divisible by p, then $a^{p-1} \equiv 1 \mod p$.

Step 1

Choose two *prime numbers*, p and q.

Step 2

Compute:

$$N = p \cdot q$$
$$\theta = (p-1)(q-1)$$

Step 3

Choose a value e that is *co-prime* with θ.

Step 4

Compute the inverse of e mod θ, i.e., find d s.t.

$$d \cdot e \equiv 1 \bmod \theta$$

The *public key* is the pair (e, N).

The *private key* is the pair (d, N).

Encoding

Acquire the public key (e, N) of the party you wish to send something to. To send the message m, encode it as

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

Decoding

If you receive c and have the private key (d, N), decode like so:

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

Two cases to consider to understand why this works.

Since $ed = 1 \bmod \theta$, there exists integer k such that $ed = 1 + k\theta$.

Case 1: m co-prime with N

$$\begin{aligned} c^d \bmod N &= (m^e)^d \bmod N \\ &= m^{1+k\theta} \bmod N \\ &= m m^{k\theta} \bmod N \end{aligned}$$

It is a known result in number theory that if m and N are co-prime, then $m^\theta \equiv 1 \bmod N$ where $\theta = (p-1)(q-1)$. Thus,

$$\begin{aligned} c^e \bmod N &= m m^{k\theta} \bmod N \\ &= m \bmod N \end{aligned}$$

Case 2: m not co-prime with N

Then, $gcd(m, N) > 1$. Must be p or q, because those are the only two factors of N.

Suppose $gcd(m, N) = p$. Then, p also divides m,

$$m \equiv 0 \bmod p, \quad m^{ed} \equiv 0 \equiv m \bmod p$$

But q does not. q is prime, so $gcd(q, m) = 1$. So by Fermat's little theorem,

$$m^{(q-1)} \equiv 1 \bmod q, \quad m^{(p-1)(q-1)} = m^{\theta} \equiv 1 \bmod q$$

Case 2: m not co-prime with N

Again, since $ed = 1 \bmod \theta$, there exists k such that $ed = 1 + k\theta$.

$$\begin{aligned} m^{ed} \bmod q &= mm^{k\theta} \bmod q \\ &= m \bmod q \end{aligned}$$

So we have

$$m^{ed} \equiv m \bmod p$$
$$m^{ed} \equiv m \bmod q$$

It follows that

$$m^{ed} \equiv m \bmod N$$

RSA and factoring

- To decrypt the message, we must learn d
- We know e, and that $de \equiv 1 \bmod \theta$

We have only e, and N.

However, N and θ are based on the same two prime numbers:

$$N = p \cdot q$$
$$\theta = (p-1)(q-1)$$

If we can factor $N = pq$, we find θ and can decode the message!

The security of RSA relies on the fact that factoring for large numbers is computationally intractable.

Computational complexity of breaking RSA

Current recommended key (N) sizes are 2048- and 4096-bit.

The largest key size cracked so far is 829 bits in 2020. See: `https://en.wikipedia.org/wiki/RSA_numbers`

Best-known classical algorithm:

General number field sieve

From Wikipedia, the free encyclopedia

In number theory, the **general number field sieve** (**GNFS**) is the most efficient classical algorithm known for factoring integers larger than 10^{100}. Heuristically, its complexity for factoring an integer n (consisting of $\lfloor \log_2 n \rfloor + 1$ bits) is of the form

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}\right) = L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$$

Computational complexity of breaking RSA

On a quantum computer, Shor's algorithm solves the problem in *polynomial time*.

But it is still going to be a long time before that happens.

RSA-2048	**Old estimates**				**Current estimates**			
p_g	n_ℓ	n_p	quantum resources	time	n_ℓ	n_p	quantum resources	time
10^{-3}	6190	19.20	1.17	1.46	8194	22.27	0.27	0.34
10^{-5}	6190	9.66	0.34	0.84	8194	8.70	0.06	0.15

Table 2. RSA-2048 security estimates. Here n_ℓ denotes the number of logical qubits, n_p denotes the number of physical qubits (in millions), time denotes the expected time (in hours) to break the scheme, and *quantum resources* (*quantum resources*) are expressed in units of megaqubitdays. The corresponding classical security parameter is 112 bits.

Image: V. Gheorghiu and M. Mosca, *A resource estimation framework for quantum attacks against cryptographic functions - recent developments.* Feb. 15 2021.