RSA cryptosystem

We describe the public key encryption algorithm by Rivest, Shamir, Adleman (1977).

Suppose Bob wants to send a secret message to Alice, but they did not have the opportunity to agree on some secret code beforehand. The solution is as follows:

Alice chooses randomly two very large prime numbers p and q and computes $n := pq$. Let us denote[1] $\varphi(n) := (p-1)(q-1)$. Then Alice chooses some number $i \in \{2, 3, \ldots, \varphi(n)-1\}$ such that i is coprime with $\varphi(n)$. She can do that by choosing i randomly and then checking that $\gcd(i, \varphi(n)) = 1$ by Euclid's algorithm. The Euclid's algorithm then produces a number j such that $ij - k\varphi(n) = 1$ (in other words, $ij \equiv 1 \pmod{\varphi(n)}$). This equation actually has infinitely many solutions, but we can choose one such that $j \in \{2, 3, \ldots, \varphi(n)-1\}$.

Now the pair (n, i) is called the **public key** and Alice can send it to Bob or put it on the Internet. She keeps the rest of the data (primes p and q and the number j) private. Note that in principle, it is possible to compute these data from the public key. (We just do the prime decomposition of n and then run the Euclid's algorithm.) Nevertheless, if the primes p and q are large enough, it is computationally practically impossible.[2]

Now, suppose Bob wants to send a message to Alice. In this setting a message is a number[3] x, $0 < x < n$. Before sending the message through a public channel he encrypts it as follows: He computes $y := x^i \bmod n$, that is, the remainder when dividing x^i by n (based on the public key). Try to think about how to do such an exponentiation quickly!

Now Alice receives y. The claim is that the original message x can be recovered as $y^j \bmod n$, that is, as the remainder when dividing y^j by n.

Before proving this, let us have a look on an example.

Example. Suppose Alice chooses $p = 11$, $q = 13$, $n = pq = 143$, $\varphi(n) = (p-1)(q-1) = 120$, $i = 17$. Let us compute j. This is easy in this case as $120 = 7 \cdot 17 + 1$. So, $1 = 120 - 7 \cdot 17 = -16 \cdot 120 + 113 \cdot 17$, so $j = 113$. Alice publishes the public key $n = 143$, $i = 17$.

Now, suppose Bob wants to send the number $x = 69$ to Alice. But he would feel somewhat embarrassed to send such a number publicly, so he wants to encrypt it. So, he needs to compute $69^{17} \bmod 143$. How to do this effectively? Using *exponentiation by squaring* (the following computation goes mod 143):

$$\begin{aligned}
69 &\equiv 69 \\
69^2 &= 4761 \equiv 42 \\
69^2 &\equiv 42^2 \equiv 48 \\
69^8 &\equiv 48^2 \equiv 46 \\
69^{16} &\equiv 16^2 \equiv 113
\end{aligned}$$

Finally, $69^{17} = 69^{16} \cdot 69 \equiv 113 \cdot 69 \equiv 75$. So, $y = 75$, which looks pretty innocent, so Bob can send this to Alice.

Now Alice is wondering, what is Bob sending to her, so she wants to decrypt the message. Therefor she needs to compute $y^{113} \bmod 143$. Try to do the computation yourself beforehand!

$$\begin{aligned}
75 &\equiv 75 \\
75^2 &\equiv 48 \\
75^4 &\equiv 48^2 \equiv 16 \\
75^8 &\equiv 16^2 \equiv 113 \\
75^{16} &\equiv 113^2 \equiv 42 \\
75^{32} &\equiv 42^2 \equiv 48 \\
75^{64} &\equiv 48^2 \equiv 46
\end{aligned}$$

Now, $113 = 64 + 32 + 16 + 1$ (in other words $113 = (1110001)_2$), so $75^{113} = 75^{64} \cdot 75^{32} \cdot 75^{16} \cdot 75 \equiv 46 \cdot 48 \cdot 42 \cdot 75 \equiv 69$.

[1] This is actually the *Euler's totient function*. We will learn about it later in the course.

[2] In fact, this is still an open question, i.e. it is not proven yet, that there is no quick algorithm for prime decomposition. We just do not know any.

[3] Remember that all data in a computer is stored as numbers.

So, it really works! Now, you may feel that doing the prime decomposition of 113 is actually fairly easy and, in particular, it is much easier than the rest of the stuff we did here. But now imagine that we double the primes. Then the prime factorization will take (about) twice as long, but the exponentiation or the Euclid's algorithm takes (about) just one more step. Double it again and the same happens. Once the primes p and q are large enough, the prime factorization becomes impossible, while the encryption/decryption process is still quite easy to handle.

Now, we prove that the algorithm works.

Theorem. Let p, q be prime numbers. Denote $n := pq$ and $\varphi(n) := (p-1)(q-1)$. Let $i, j \in \mathbb{N}$ satisfy $ij \equiv 1 \pmod{\varphi(n)}$. Then for every $x \in \mathbb{Z}$, we have $x^{ij} \equiv x \pmod{n}$.

Proof. First, recall that $ij \equiv 1 \pmod{\varphi(n)}$ means that $ij = k\varphi(n) + 1 = k(p-1)(q-1) + 1$ for some k. Secondly, note that $x^{ij} \equiv x \pmod{n}$ is equivalent to $x^{ij} \equiv x \pmod{p}$ and $x^{ij} \equiv x \pmod{q}$ (try to prove!). So, we will prove that $x^{ij} \equiv x \pmod{p}$ and the proof for q is then literally the same.

Suppose first that $x \perp p$. Then by the little Fermat's theorem, we have $x^{p-1} \equiv 1 \pmod{p}$. We can raise this to the power $k(q-1)$ and then multiply by x to obtain

$$\begin{aligned} x^{p-1} &\equiv 1 \pmod{p} \\ x^{k(p-1)(q-1)} &\equiv 1^{k(q-1)} = 1 \pmod{p} \\ x^{ij} = x^{k(p-1)(q-1)+1} &\equiv x (\bmod p) \end{aligned}$$

Now, suppose that $x \not\perp p$. This means that $x = ap$ for some a. But then $x \equiv 0 \pmod{p}$ as well as $x^{ij} \equiv 0 \pmod{p}$. $\square$