A First Guide of RSA Public Key Encryption By Example

DogtorDoggo

Last modified May 21, 2020

This document as well as its tex file can be obtained from

`https://github.com/dogtordoggo/rsa-guide`

and distributed, modified for your own educational or any other purpose freely as long as credit is given to the original author.

The purpose of this writeup is to serve as simple as possible an explanation of RSA public-key encryption as well as a demo showing how it works.

Prerequisites

Basic understanding of **modulo arithmetic** as well as **Euler's Theorem** and **Fermat's Little Theorem** are assumed prerequisites before heading into this document. Their proof isn't too hard though if anyone is interested in figuring them out, but here they are (from wikipedia):

Theorem 1 (Euler's Theorem): If n and a are coprime positive integers, $\phi(n)$ is Euler's totient function, then

$$a^{\phi(n)} = 1 \bmod n$$

Corollary 1.1 (Corollary of Euler's Theorem): For any integer $M < n = p \cdot q$, where p and q are two prime numbers, and any integer r, there is

$$M^{r \cdot \phi(n)+1} = M \bmod n$$

See Appendix for the proof of this corollary.

Theorem 2 (Fermat's Little Theorem): If p is a prime number, then for any integer a

$$a^p = a \bmod p$$

or

$$a^{p-1} = 1 \bmod p$$

What we need:

1. Two large prime numbers p and q and $n = p \cdot q$, for the sake of demonstration and simplicity we pick $p = 41, q = 67$. We can then calculate

$$\begin{aligned} n &= p \cdot q = 41 \cdot 67 = 2747 \\ \phi(n) &= p \cdot q \cdot (1 - \frac{1}{p}) \cdot (1 - \frac{1}{q}) = (p-1) \cdot (q-1) = 40 \cdot 66 = 2640 \end{aligned}$$

   Note that $\phi(n)$ is Euler's totient function of n.

2. A small prime number $s = 13$.

From here we can calculate other variables needed for our RSA algorithm. First of all, according to Corollary of Euler's Theorem, for any integer $M < n$, and any integer r, there is

$$M^{r \cdot \phi(n)+1} = M \bmod n$$

s is a (small) prime number, therefore $\gcd(s, \phi(n))) = 1$, so the inverse of s mod $\phi(n)$ exists (See Lemma 1 in Appendix), note it as $t = s^{-1} \bmod \phi(n)$. t could be calculated using *Euclid's algorithm*. See Appendix for a source code doing Euclid's algorithm calculating those coefficients. What this also means is that there exists some integer r such that $s \times t = r \times \phi(n) + 1$.

tl;dr: We need two large prime numbers p and q, and a smaller prime number s, then $(s,\ p \cdot q)$ would be public key pair. Another integer $t = s^{-1} \bmod \phi(p \cdot q)$ is calcualted, and $(t,\ p \cdot q)$ would be private key pair.

RSA

Now we can deploy our public-key encryption algorithm. Make (s, n) the public key pair, and (t, n) private key pair. Denote M as plaintext message, E as encrypted message.

Encryption:

$$E = M^s \ mod \ n$$

Decryption:

$$M = E^t \mod n$$

Proof:

$$\begin{aligned} E^t \bmod n &= (M^s)^t \bmod n \\ &= M^{st} \bmod n \\ &= M^{r\phi(n)+1} \bmod n \\ &= M \bmod n \\ &= M \end{aligned}$$

Why is this secure?

Without the knowledge of p and q, we can't find $\phi(n) = (p-1)\cdot(q-1)$, therefore we can't calculate $t = s^{-1} \bmod \phi(n)$ based on the information of (s, n). This means we can't calculate private key pair's t from the public key pair (s, n).

Example:

Let's continue with the example where we chose $p = 41, q = 67$ then we have $n = 41 \cdot 67 = 2747, \phi(n) = 40 \cdot 66 = 2640$, then we choose $s = 13$. Then by Euclid's Algorithm, we have

$$st + r\phi(n) = 13 \times -203 + 1 \times 2640 = 1$$
$$t = -203 \bmod 2640 = 2437$$

Therefore in this case public key is $(s, n) = (13, 2747)$, the private key pair is $(t, n) = (2437, 2747)$. Suppose our original message is an integer $M = 2017$, the year of this writeup. We can then go ahead and calculate encrypted message

$$E = M^s \bmod n = 2017^{13} \bmod 2747 = 800$$

There is a binary-like efficient algorithm to calculate the above modulo due to the simple fact that

$$a \cdot b = (a \bmod n) \cdot (b \bmod n) \bmod n$$

A matlab command of mod(2017^13, 2747) would produce *wrong* result. An efficient algorithm calculating modulos of large exponential modulos is given in Appendix. To recover original message from the encrypted message, we can do

$$M = E^t \bmod n = 800^{2437} \bmod 2747 = 2017$$

And we recovered the original message of 2017 from the encrypted message of 800.

Appendix

Lemma 1 For two positive integers s and n, $n > 1$, there exists s's inverse mod n, noted as $s^{-1} \bmod n$, if and only if $\gcd(s, n) = 1$.

Proof

Necessity: If $\gcd(s, n) = 1$, by Euclid's algorithm, there exist integers t and r such that

$$t \cdot s + r \cdot n = 1$$

Therefore $t = s^{-1}$ *mod* n.

Sufficiency: If there exists an integer $t = s^{-1} \bmod n$, it means for some integer r there is

$$t \cdot s = r \cdot n + 1$$

If $\gcd(s, n) = k > 1$, we can divide both sides of above equation by k and get

$$t \cdot \frac{s}{k} = r \cdot \frac{n}{k} + \frac{1}{k}$$

Which is impossible.■

Proof of Corollary of Euler's Theorem:

When M is coprime of n, the above equation is direct result of Euler's Theorem.

When not, M must be multiple of p, $M = kp$ where $1 \leq k < q$ or multiple of q, $M = kq$ where $1 \leq k < p$. Suppose the first case, $M = kp$, $1 \leq k < q$, M is coprime of q obviously, according to Fermat's little theorem, we have

$$M^{q-1} = 1 \bmod q$$

Therefore

$$M^{r \cdot (p-1) \cdot (q-1)} = M^{r \cdot \phi(n)} = 1 \bmod q$$

This can be re-written as

$$M^{r \cdot \phi(n)} = zq + 1$$

for some integer z. Multiple both sides of the equation by $M = k \cdot p$, we have

$$M^{r \cdot \phi(n)+1} = z \cdot k \cdot p \cdot q + M = z \cdot k \cdot n + M$$

Which is equivalent to

$$M^{r \times \phi(n)+1} = M \bmod n$$

The $M = k \cdot q$ case is symmetrical thus proof is omitted. ■

A piece of code doing Euler's Algorithm, calculating gcd and the coefficients s and t of two integers a and b such that

$$sa + tb = \gcd(a, b)$$

can be found at https://github.com/dogtordoggo/rsa-guide/EuclidsAlgorithm.kt.

A piece of code calculating modulos of large exponentials like

$$800^{2437} = 2017 \bmod 2747$$

can be found at https://github.com/dogtordoggo/rsa-guide/BigNumberModulo.kt.