

Cryptography

Public Key Crypto: RSA - analysis and implementation

Professor: Marius Zimand

Public Key Cryptography.

- Main application: the key distribution problem (what if Alice and Bob are from the onset far apart?)
- Idea: instead of having a “lock” with one key that can lock and unlock, we could use a lock which has a key that can only lock, and a second key which can only unlock.
- Consequence: Alice distributes locks and locking keys in public and keeps the unlocking keys secret. So, everybody can take a lock and the locking key for it, put the message in the box, lock the box, and send the box to Alice. Only Alice can unlock the box and read the message.
- PK crypto realizes the above idea mathematically.
- Diffie and Hellman “New directions in cryptography”, IEEE Trans. in Information Theory, 22(1976), pp 644–654. They had the idea but no concrete implementation. Rivest, Shamir, and Adleman had the first concrete implementation, the RSA PK cryptosystem, in 1977.
- British Intelligence Services officers had the same ideas a little bit earlier: James Ellis (1970) and Clifford Cocks (1973), but kept it classified.

Theoretical foundations of PKC

- in PKC, each user has a key pair: (public key, private key). The public key is distributed to everyone and is used for encryption. The private key is used for decryption.
- Note that anyone can encrypt. But only someone who has the the private key can reverse the encryption operation, i.e., decrypt.
- The only (currently) known way to implement this idea is based on some classical problems in number theory.
- These problems provide examples of **one-way trapdoor functions**. These are functions that are easy to compute, but hard to invert except one knows some secret information. We will discuss this concept more thoroughly later.

The RSA public key cryptosystem

- Key set-up

1. first choose p, q - two large random prime numbers.
2. take $n = p \cdot q$
3. take e such that $\gcd(e, (p-1)(q-1)) = 1$
4. take d such that $e \cdot d = 1 \pmod{(p-1)(q-1)}$
5. the keys are ready:

the public key is (n, e)

the private key is (d, p, q)

- encryption The message is m , a number such that $m < n$.

Calculate $c = m^e \pmod{n}$. c is the ciphertext.

- decryption

Calculate $c^d \pmod{n}$. This is equal to m .

Why does it work?

Reason: $c^d = (m^e)^d \pmod{n} = m^{e \cdot d} \pmod{n}$.

Note that $\phi(n) = (p-1)(q-1)$ and $ed = 1 \pmod{(p-1)(q-1)}$.

Consequently: if $\gcd(m, n) = 1$, then $m^{e \cdot d} = m \pmod{n} = m$ (last equality holds because $m < n$).

What if $\gcd(m, n) \neq 1$? The decryption still works for the following reason.

Say $m = 0 \pmod{p}$.

Then $m^{ed} = m \pmod{p}$ (both sides are 0).

$m^{ed} = m \pmod{q}$ (by Fermat's Theorem).

So both p and q divide $m^{ed} - m$.

Therefore $m^{ed} - m = 0 \pmod{n}$.

One example:

Bob chooses:

1. $p = 885320963$

$$q = 238855417$$

$$n = p \cdot q = 211463707796206571$$

$$e = 9007$$

$$d = e^{-1} \pmod{(p-1)(q-1)} = 116402471153538991$$

Bob's public key is $(n, e) = (211463707796206571, 9007)$.

Bob's private key is $(d, p, q) = (116402471153538991, 885320963, 238855417)$.

2. Alice wants to send to Bob the message "cat". First she needs to convert it to a number.

Let's say that their convention is that $a \leftrightarrow 01$, $b \leftrightarrow 02$, $c \leftrightarrow 03$, ..., $z \leftrightarrow 26$

Then "c a t" converts to 03 01 20, i.e., to the number $m = 30120$.

Alice computes $c = m^e \pmod{n} = 30120^{9007}$

$$= 113535859035722866 \pmod{211463707796206571}$$

(she knows e and n and the operation is modular exponentiation, which can be done fast).

She sends c to Bob.

Bob decrypts by calculating $c^d \pmod n = 113535859035722866^{116402471153538991} \pmod{211463707796206571} = 30120 = 3\ 01\ 20 = \text{c a t}$.

Analysis

Security of RSA relies on the difficulty of finding m given n, e and c . This problem is called RSAP (RSA Problem).

It amounts to showing that the RSA encryption function is a one-way function.

Currently, there is no mathematical proof that this is so. Even worse, at this moment there is no mathematical proof that any function is one-way. However, there are some functions that people strongly believe to be one-way.

DEFINITION: Given two problems, A and B , we say that A reduces in polynomial time to B , (notation $A \leq_p B$), if given access to an oracle that solves B , one can solve A efficiently (efficiently = polynomial time).

Note that $\text{RSAP} \leq_p \text{FACTORING}$. (If we can do FACTORING, then we can find p and q , and then easily compute $d = e^{-1} \pmod{(p-1)(q-1)}$, and next determine $m = c^d \pmod n$.)

The basic assumption on which the security of RSA is based is:

FACTORING LARGE INTEGERS IS NOT FEASIBLE.

(of course, factoring n can be done in principle, but it takes too much time if n is large -something like the age of universe or similar).

So what can an attacker do?

- Eve knows: n, e, c .
- Eve does not know: p, q, d, m .

1. Eve would love to have d (because then she could decrypt any message).

$d = e^{-1} \pmod{(p-1)(q-1)}$; so d is easy to calculate if p and q are known. But under our assumption, Eve cannot get p and q from what she knows (which is n).

Is there another way to find d ?

Fact 1 *If we know d and e , then we can efficiently factor n (the algorithm is probabilistic and there is a very small probability that it fails).*

So, there is no efficient way to find d (under our basic assumption).

2. But maybe Eve does not need d . In fact $c = m^e \pmod{n}$, so why doesn't she just take the e -th root of $c \pmod{n}$? This is called the ROOT problem. The ROOT problem is believed to be hard, but it is not known to be as hard as FACTORING.

However, recall that he have seen that taking the SQRT \pmod{n} is equivalent to FACTORING (in the sense that if you can do one of them, then you can efficiently do the other one).

Taking the e -th root does not seem to be easier than taking the 2-root (SQRT). So there is evidence that the RSA problem is indeed hard (more precisely, as hard as FACTORING).

How hard is it to factor?

The best currently-known algorithm for factoring work in time $O(e^{\sqrt{\log n \cdot \log \log n}})$.

Note that the length of n in binary is $\approx \log n$. So the above algorithm is an exponential time algorithm.

Here is just a list of some of these algorithms:

- General purpose:
 - General Number Field Sieve (GNFS)
 - Quadratic sieve (QS)
 - lattice sieving

- Special purpose (some of them are heuristic - not known to succeed on all numbers)
 - elliptic curve method
 - Pollard's ρ method
 - Pollard's $p - 1$ method
 - ...

Comparison between GNFS and QS:

- for numbers with ≤ 110 digits: QS is better
- for numbers with ≥ 120 digits: GNFS is better

Records in factoring (a good way to assess the state of the art)

Rivest's estimation in 1977: to factor a 129-digit number requires 40000 trillion years = $40 \cdot 10^{15}$ years (age of universe estimated at around 10^{14} years).

That was the first RSA challenge: RSA-129, award \$100.

In fact, RSA-129 was factored in 1994 - Atkins, Graff, Lenstra, Leland + 600 volunteers using 1600 computers for about one year.

Last RSA-challenge that was factored: RSA-640 in Dec. 2005. This is a number with 640 bits (193 digits).

Current recommendations

Individual users: n should have 768 bits (231 digits)

Organizations (short term): 1024 bits (308 digits)

Organizations (long term): 2048 bits (616 digits)

Attacks on RSA

The bottom line: RSA is safe (as long as FACTORING IS HARD), but some choices of the parameters are weak. So implementation has to be done carefully.

Fact 2 *If $d < \frac{1}{3}n^{1/4}$, then from (n, e) one can find efficiently d .*

So d must be large.

Fact 3 *Suppose $n = p \cdot q$ has t digits. If we know the first (or the last) $t/4$ digits of p , then we can efficiently factor n .*

So p and q must be really random.

For example, suppose we want to get p with 120 digits. We choose 60 random digits and form the number N , then take $N' = N \cdot 10^{60}$, then try $N' + 1, N' + 2, \dots$ till we hit a prime number and we take that one as p .

What's wrong with this scheme?

Fact 4 *Sharing the modulus is bad.*

Assume that for efficiency each user has

- the same n
- different public/private exponents: user i has (e_i, d_i) .

Suppose I am user one: can I find the private d_2 of user two?

YES: Having d_1 , I can find p and q (if I don't have them to start with).

Then $d_2 = e_2^{-1} \pmod{(p-1)(q-1)}$.

So each user can find every other users key.

Now suppose that the attacker is not one of the people sharing n .

Suppose Alice sends the same message m to two people with public keys: (n, e_1) and (n, e_2) (note that they have the same n).

Eve will see

$$c_1 = m^{e_1} \pmod{n}$$

$$c_2 = m^{e_2} \pmod{n}$$

.

It is likely that $\gcd(e_1, e_2) = 1$.

Eve computes positive numbers t_1, t_2 so that

$$t_1 e_1 - t_2 e_2 = 1.$$

Now Eve computes

$$c_1^{t_1} c_2^{-t_2} = m^{e_1 t_1} m^{e_2 (-t_2)} = m^{e_1 t_1 - e_2 t_2} = m^1 \pmod{n} = m.$$

Primality Testing

To implement RSA, we need to choose p and q , two large prime numbers. To do this we want to proceed as follows: pick a random large number and check if it is prime or not. If it is, we take it as p . If not we try one more time.

But how do we check if a large number is prime or not? This is called PRIMALITY TESTING. Recall that we cannot factor large numbers.

Good news:

PRIMALITY TESTING is easy. FACTORING is hard.

Miller-Rabin Primality Test

The Miller-Rabin Test is derived from Fermat theorem.

Example. Recall that if p is prime then $2^{p-1} = 1 \pmod{p}$.

let us check if 35 is prime, without attempting to factor it.

$2^2 = 4 \pmod{35}$, $2^4 = 16 \pmod{35}$, $2^8 = 256 = 11 \pmod{35}$, $2^{16} = 121 = 16 \pmod{35}$, $2^{32} = 256 = 11 \pmod{35}$.

So $2^{34} = 2^2 \cdot 2^{32} = 4 \cdot 11 = 9 \neq 1 \pmod{35}$.

So 35 is not prime (and we did not factor it).

However $2^{560} = 1 \pmod{561}$ and 561 is not prime.

Another useful observation.

Fact 5 *If we find two integers x, y with $x \not\equiv \pm y \pmod{n}$, and $x^2 \equiv y^2 \pmod{n}$, then n is composite.*

Proof. $x^2 \equiv y^2 \pmod{n} \Leftrightarrow (x + y)(x - y) \equiv 0 \pmod{n}$.

Let $d = \gcd(x - y, n)$. If $d = 1$ then $x + y$ is divisible by n , so $x \equiv -y \pmod{n}$ which is not true. So $d \neq 1$.

If $d = n$, then $x - y$ is divisible by n , so $x \equiv y \pmod{n}$ which is not true. So $d \neq n$.

It means that d is a proper divisor of n , so we factored n .

Miller-Rabin Probabilistic algorithm

Input: $n > 1$.

Goal: Is n prime or composite?

Let $n - 1 = 2^k \cdot m$, with m odd.

Choose a random $a \in \{1, \dots, n - 1\}$. (this is the probabilistic step)

Perform the following $Test(n, a)$.

$Test(n, a)$:

Compute $b_0 = a^m \pmod{n}$. If $b_0 = \pm 1$, stop, and declare " n probably prime".

If not, $b_1 = b_0^2 \pmod{n}$.

If $b_1 = 1 \pmod{n}$, then n is composite (because $b_0^2 = 1^2 \pmod{n}$, and we use the above fact). So stop.

If $b_1 = -1 \pmod{n}$, then stop and declare " n probably prime".

Else let $b_2 = b_1^2$ and do the same we did for $b_1 \dots$

...

Continue until stopping or reaching b_{k-1}

If $b_{k-1} \neq -1 \pmod{n}$, then n is composite.

If $b_{k-1} = -1 \pmod{n}$, then declare " n probably prime".

Example. $n = 561$. $n - 1 = 560 = 2^4 \cdot 35$. So $k = 4, m = 35$.

Let's say we pick $a = 2$.

$$b_0 = 2^{35} = 263 \pmod{561}$$

$$b_1 = b_0^2 = 166 \pmod{561}$$

$$b_2 = b_1^2 = 67 \pmod{561}$$

$$b_3 = b_2^2 = 1 \pmod{561}.$$

So 561 is composite.

Fix a . If $Test(n, a)$ declares n is composite, then n **is** composite for sure.

If $Test(n, a)$ declares n is probably prime, then n may be in fact composite.

Definition 6 *We say that n is a strong pseudoprime, if n is composite and there is some a such that $Test(n, a)$ is declaring "n is probably prime".*

Strong pseudoprimes are bad because they are fooling the Test for some a .

From 1 to 10^{10} there are 455052511 primes and only 3291 strong pseudoprimes.

But the situation is even better.

Fact 7 *For any composite n , the probability over the choices of a that $Test(n, a)$ declares n probably prime is $< 1/4$.*

So if we try, say, 10 a 's, the probability that n is fooling every one of them is $< \frac{1}{4^{10}} \approx 10^{-6}$.

NOTE: In the Summer of 2002, Agrawal, Kayal, and Saxena discovered a deterministic (no error!) polynomial-time algorithm for PRIMALITY TESTING. This was a major breakthrough in mathematics (made the first page of New York Times).

Still not as practical as Miller-Rabin.

How do we find p and q

Let's say we want a prime number p with 120 digits. This is how we obtain it.

- pick randomly a number t with 120 digits.
- check t with the Miller-Rabin test to see it is prime or not.
- if it is, we are done. We take p to be t .
- if it's not, we try again.

How many attempts should we expect?

There are $\pi(10^{120}) - \pi(10^{119})$ prime numbers with 120 digits. This is approximately

$$\frac{10^{120}}{\ln(10^{120})} - \frac{10^{119}}{\ln(10^{119})} \approx \frac{10^{120} - 10^{119}}{\ln(10^{120})}.$$

So the probability that a randomly-chosen number t with 120 digits is prime is approximately

$$\frac{1}{\ln(10^{120})} \approx \frac{1}{270}.$$

So approximately, one in 270 randomly-chosen numbers of 120 digits turns out to be prime.

But we can skip even numbers, multiple of 3, of 5, etc. In practice we try less than 120 numbers till we hit a prime number with 120 digits.

Generation of RSA keys

- Choose e . (Popular choices are $e = 3$ - but there are some problems with this choice- or $e = 2^{16} + 1$.)
- Choose p and q as discussed above using Miller-Rabin. Check that $\gcd(e, p-1) = 1$ and $\gcd(e, q-1) = 1$. If not try again.
- Calculate $n = p \cdot q$.
- Calculate with the extended Euclidean algorithm $d = e^{-1} \pmod{(p-1)(q-1)}$. Check if $d > n^{1/4}$. If not, try again.

This is the so-called "textbook" RSA. It is just a raw crypto primitive. We'll see later that there are some security problems with it. In real-world applications, textbook-RSA is only a tool that is used in more complex protocols (to be discussed later).