Handout #17: RSA(*)

In a classical cryptosystem, the decoding key d_k is easily derived from the encoding key e_k (if not exactly the same). The idea behind a public key system is that it is "very hard" to compute d_k even if we know e_k. The RSA cryptosystem is such a system based on number theory. It can be described as follows.

To setup the system, each participant a (let us imagine that they are: Alice, Bob, ...) does the following:

1) Construct two secret large (100 digits) prime numbers: p_a and q_a.

2) Compute
$$n_a := p_a\, q_a,$$
$$\varphi(n_a) = (p_a - 1)\,(q_a - 1).$$

3) Choose at random in the set $\{2, \ldots, n_a - 1\}$ a integer e_a such that
$$(e_a, \varphi(n_a)) = 1.$$

4) Compute an inverse ($f_a \bmod \varphi(n_a)$) of e_a.

We will describe in the sequel how to do step 1). Step 2) is easy, Step 4) has already been described in a previous handout, and to realize Step 3) we only need to choose e_a at random and test if $(n, a) = 1$, if not we just choose another e_a and keep doing this until the select number passes the test. It can be shown that after a "few" trials one will get a number that works.

After the initial setup, each participant "publishes" under his/her name the values n_a and e_a. Thus the name public key system, since everyone knows the encoding keys of the participants.

To send a message to b = Bob, a = Alice starts by translating her message into an integer m by replacing each plaintext letter (including spaces and punctuation) by a number ($\mathrm{a} \mapsto 10, \mathrm{b} \mapsto 11, \mathrm{c} \mapsto 12, \ldots$). If needed the message is cut into pieces so that each piece corresponds to an integer $< n_b$. To be sure that only Bob will be able to decode her message, Alice encodes m in the following manner
$$c := (m^{e_b} \mod n_b)$$
We have seen that this computation can be done quite easily (and rapidly). As we will explain below, to decode the cypher c, Bob need only do the following
$$m = (c^{f_b} \mod n_b)$$
Moreover, if Alice and Bob want to be sure that only Alice could have sent the message and only Bob can decode it, then they can proceed as follows. Alice first starts to "encode" her message using

(*) Stands for: Rivest, Shamir, and Adleman.

her secret key f_b, and then goes on the encode the result using Bob's public key e_b. More precisely,

$$c := ((m^{f_a} \mod n_a)^{e_b} \mod n_b)$$

To decode, Bob does the reverse steps:

$$m = ((c^{f_b} \mod n_b)^{e_a} \mod n_a)$$

The fact that all this works hangs on the following theorem

Theorem (Euler-Fermat) If a is relatively prime to n then

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Proof. We first observe that, for a relatively prime to n, the function

$$m_a : \{0, \ldots, n-1\} \longrightarrow \{0, \ldots, n-1\}$$

such that

$$m_a : b \mapsto a\, b,$$

is a bijection (why ?). Now, let

$$\{x_1, x_2, \ldots, x_{\varphi(n)}\}$$

be the set of all numbers between 1 and n that are relatively prime to n. Then

$$\{a\, x_1, a\, x_2, \ldots, a\, x_{\varphi(n)}\} = \{x_1, x_2, \ldots, x_{\varphi(n)}\}$$

since m_a is a permutation.

The product of all the numbers in the first set is the product of all of the numbers in the second set.

$$x_1 x_2 \cdots x_{\varphi(n)} \equiv a\, x_1 a\, x_2 \cdots a\, x_{\varphi(n)} \pmod{n}$$

All of the x_i's have inverses so they can be canceled off of both sides of the equation. so we also have:

$$1 \equiv a \cdot a \cdots a \equiv a^{\varphi(n)} \pmod{n}.$$

To explain why decoding is possible in RSA, we need only observe that

Theorem. For m relatively prime to n, if

$$e\, f \equiv 1 \pmod{\varphi(n)}$$

then

$$(m^e)^f \equiv m \pmod{n}$$

Proof. We have $e\,f = q\,\varphi(n) + 1$, hence

$$\begin{aligned}(m^e)^f &\equiv m^{e\,f} \pmod{n}\\ &\equiv m^{q\,\varphi(n)+1} \pmod{n}\\ &\equiv (m^{\varphi(n)})^q\,m \pmod{n}\\ &\equiv 1^q\,m \pmod{n}\\ &\equiv m \pmod{n}\end{aligned}$$

The probability that the message is not relatively prime to n_a is so small (why ?) that we can always suppose that this is the case.

The reason why RSA is considered to be "secure" when p_a and q_a are large enough (with some more conditions on their properties), is because "factorization" of large integers (namely n_a) is thought to be computationally infeasible. At the moment factoring algorithms are able to crack numbers of up to 130 digits, so that our choice of size for p and q makes n a lot larger (200 digits).

We can rebuild $f = f_a$ out of $n = p\,q$ and $e = e_a$, if we know the value of $\varphi(n)$. It is easy to see that knowing $\varphi(n)$, we can easily compute p and q since the equations

$$\begin{aligned}p\,q &= n\\ (p-1)\,(q-1) &= m\end{aligned}$$

can be solved to get p and q, but even better, since

$$n + 1 - m = p + q,$$

the roots of the polynomial

$$x^2 - (n+1-m)\,x + n = (x-p)\,(x-q)$$

are exactly p and q. Hence we can conclude in this case, that the knowledge of $\varphi(n) = (p-1)\,(q-1)$ is equivalent to the knowledge of the factorization of $n = p\,q$. It is believed (although not proved) that breaking RSA is "polynomially equivalent" to factoring n.

Need for more Number Theory

As we have seen, to setup RSA, we need to be able to produce big prime numbers. Since factorization is hard, we would like to do it some other way. We will achieve this by selecting a number n at random, and then testing for its primality using a probabilistic approach due to Strassen and Solovay. If n happens to be prime then it will pass of the test. On the other hand, if n is not

a prime it will pass the test with probability less than $(1/2)^k$, with k large (50). In doing so the probability that we draw the wrong conclusion is less than 1 in 2^{50} and these are odds which are considerably smaller than those we do stake our lives upon daily!

To describe the primality test, we will need more number theory, and this will be presented in an up coming handout.

Attacks on RSA

Learning the decryption exponent

It is interesting to note that if we know the decryption exponent f, then we can use it to factor n rapidly. Hence to keep the system secure if f is revealed, we cannot simply choose another e, but we need to build a new n. Knowing f, the procedure described below will factor n with probability $1/2$, otherwise it will give the answer "try again". The probability that it fails to factor after m iteration is $1/2^m$, and if m is large enough ($m \geq 20$) this becomes vanishingly small, so that we are "sure" to get the factorization rapidly since each iteration is very fast.

We first recall that we have shown (handout #16) that, for a prime p, the congruence

$$x^2 \equiv 1 \pmod{p}$$

has only two solutions and these are $x \equiv \pm 1 \pmod{p}$. Thus if we know that n factors as the product of the primes p and q, then

$$x^2 \equiv 1 \pmod{n} \iff x \equiv \pm 1 \pmod{p} \quad \text{and} \quad x \equiv \pm 1 \pmod{q}$$

and the resulting 4 solutions can be found using the Chinese Remainder Theorem. Two of these are $x \equiv \pm 1 \pmod{n}$, and the other two are negatives of one another. If x is a non trivial ($\neq \pm$) solution, then

$$n \mid (x-1)(x+1)$$

and n divides neither factor on the right side. This implies that

$$(x+1, n) = p \qquad \text{or} \qquad (x+1, n) = q,$$

and the left side of each of these equation is easy to compute. Thus if we can find one of the 2 non trivial solutions of $x^2 \equiv 1 \pmod{n}$ (with only the knowledge of f), then we have succeeded in factoring n.

1) **Choose** k at random between 1 and $n-1$.

2) **Compute** $x := (k, n)$.

3) **If** $1 < x < n$ **then** x is a factor of n and it must be equal to p or q, so we are **finished**.

4) **Write** $e\,f - 1 = 2^s\,r$, with r odd.

5) **Compute** $m := k^r \pmod{n}$.

6) **If** $m \equiv 1 \pmod{n}$ **then** **try again**.

7) **Find** the least j $(0 \leq j \leq s)$ such that $m^{2^j} \equiv 1 \pmod{n}$, and set

$$x := m^{2^{j-1}}$$

thus $x^2 \equiv 1 \pmod{n}$.

8) **If** $x \equiv -1 \pmod{n}$ **then** **try again**,

else $(x + 1, n)$ is a factor of n and it must be equal to p or q, so we are **finished**.

Observe that step 7) is sure to succeed since

$$e\,f - 1 = 2^s\,r \equiv 0 \pmod{n}.$$

Bad protocol

If we know that Alice has decided to encrypt and send separately each letter of her message to BOB using RSA (even with very large n and e) then it is easy to decrypt her message without knowing Bob's secret decryption key. We need only encrypt, using Bob's public key, each plaintext letter, and compare the cypher letters of Alice's message to our table of cypher letters.

This observation can be adapted to attack RSA if we know which are the possible messages and if the number of these is "small". For instance sending a PIN of 4 digits with RSA is not secure if the opponent knows that this is what you are sending.

Further attacks

A lot of research paper have been written on various weaknesses or strength of RSA, but the system is still considered to be secure if some care is taken in setting it up.