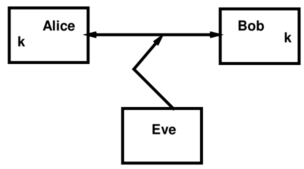
W4231: Analysis of Algorithms

12/3/1999

- RSA

Secure Communication with Secret Key



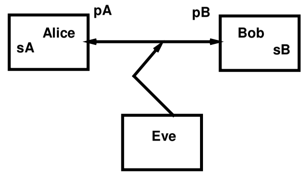
Caesar's Code

The secret key is an integer k in the range $1, \ldots, 25$. To encode a word, take each character, and substitute it with the character that comes k places after it in the alphabet. (After 'z' start again with 'a'.)

Easy to break. Example:

g rutm zosk gmu
ot g mgrgde lgx lgx gcge

Secure Communication with "Public" Key



RSA Function — Key Generation

Start by generating at random two big primes p and q.

Define $n = pq$. Compute $\phi(n) = (p-1)(q-1)$.

Find a e such that $\gcd(e, \phi(n)) = 1$.

Find an inverse of e, i.e. a d such that $ed = 1 \pmod{\phi(n)}$.

Public key: (n, e). Secret key: (n, d).

RSA Function

For n, e as before, define the following function mapping $\mathbf{Z}_n$ into $\mathbf{Z}_n$:

$$RSA_{n,e}(x) = x^e \pmod{n}$$

Invertibility given the private key

We want to prove that when p, q are odd primes, $n = pq$, and $ed = 1 \pmod{\phi(n)}$ then for every $m \in \mathbf{Z}_n$

$$m^{ed} \pmod{n} = m$$

First, we have $ed = 1 + k\phi(n) = 1 + k(p-1)(q-1)$ for some k, so

$$m^{ed} = m^{1+k\phi(n)} = m \cdot (m^{k(q-1)})^{p-1} = m \pmod{p}$$

and

$$m^{ed} = m^{1+k\phi(n)} = m \cdot (m^{k(p-1)})^{q-1} = m \pmod{q}$$

Now consider the system of equations

$$\begin{aligned} x = &\ m \pmod{p} \\ x = &\ m \pmod{q} \end{aligned}$$

We saw $m^{ed} \bmod n$ is a solution, but also m is a solution. And since we can have only one solution in the range $0, \ldots, pq-1$, then $(m^{ed} \bmod n) = m$.

Relation with Factoring

It is conjectured that if n is generated by picking at random two big primes and multiplying them, then factoring n is an intractable problem.

In particular, the problem is beyond current computing power when $p, q \approx 2^{500}$.

If so, then given n so generated it is hard to compute $\phi(n)$.

(Because from $\phi(n)$ we can reconstruct the factorization of n — how?)

And also computing d given e and n is as hard as factoring n.

RSA Assumption and Relation with Factoring

Finding the secret key d given n, e is as hard as factoring n, but it could not be the only way to break the system.

It is conjectured that when n is chosen at random as product of two primes, e is chosen at random among the integers co-prime to $\phi(n)$, m is chosen at random in $\mathbf{Z}_n$, and we give an adversary $m^e \pmod{n}$ and also n and e, it is infeasible for the adversary to reconstruct m.

Here by infeasible we mean that an adversary running in polynomial time has a very low probability of success. Note that this is an "average case" notion of hardness.

This is called the "RSA assumption." It is not known how to prove that that the hardness of factoring implies the RSA assumption. Clearly if factoring is easy then the RSA assumption fails. The RSA assumption has resisted more than twenty years of attacks, and even if it is not known to be equivalent to factoring, it is considered safe.

Security of Encryption

The RSA assumption captures the fact that the entire message cannot be reconstructed by a party that does not the secret key.

This is very far from giving secure encryption.

In a secure encryption:

- Not even *single* bits of the message should leak. In fact not even information about the message should leak.

  $RSA_{n,e}(m)$ always leak the Jacobi symbol of m modulus n. (It does not matter what it is, but it is a bit of information about m).

- Even more importantly, secure encryption should be randomized.

  Every deterministic encryption has the shortcoming that when we send twice the same message the adversary knows that we are sending twice the same message.

Public Key Cryptosystem

A public-key cryptosystem in general is made of three algortihms:

- A "key-generation" algorithm randomly produces matching pairs (public-key,private key).
- An encryption algorithm $E(\cdot,\cdot,\cdot)$ than on input a public key, a message (plaintext), and random bits produces an encryption (ciphertext).
- A decryption algorithm $D(\cdot,\cdot)$ that on input a private key and a ciphertext gives in output the decryption of the message.

Semantic Security

A public key cryptosystem is semantically secure if

- For every property P defined on plaintextes
- For every distribution X of plaintextes
- If there is an efficient algorithm A that given an encryption of a x sampled from x guesses with success probability p whether x has property P.
- Then the same guessing probability could have been achieved without looking at the encryption.

Public Key Cryptosystems Based on RSA

Key Generation:

All the public key cryptosystem based on RSA have the same key generation, as described in previous section: Two primes p and q are randomly generated, then one defines $n = pq$, and picks a random e s.t. $\gcd(e, \phi(n)) = 1$, and computes d such that $ed = 1 \pmod{\phi(n)}$. Then the private key is (d, n) and the public key is (e, n).

Encryption and Decryption — Simplest Case

The simplest public-key cryptosystem based on RSA is just

- $E((e,n), m) = RSA_{e,n}(m) = m^e \pmod{n}$.
- $D((d,n), c) = c^d \pmod{n}$.

where the encryption algorithm uses no randomness. This is very unsecure.

Encryption and Decryption — With Some Randomness

A more reliable implementation, whose security is still not known to relate to the RSA assumption is

- $E((e,n),m,r) = RSA_{e,n}(m2^k + r)$ $= (m2^k + r)^e \pmod{n}$.
- $D((d,n),c)$ computes $z = c^d \pmod{n}$ and then outputs $\lfloor z/2^k \rfloor$.

The random string r is k bits long. Typically $k \approx 100$.

Encryption and Decryption — Bellare Roghaway

Suppose now we have a “cryptographic hash function” h mapping $\mathbf{Z}_n$ into $\mathbf{Z}_n$. Then consider the system

- $E((e,n),m,r) = RSA_{e,n}(r)||(h(r) + m \bmod n)$
- $D((d,n),c_1||c_2)$ computes $r = c_1^d \pmod{n}$ and then outputs $c_2 - h(r) \bmod n$.

Security

If h were a *random function* distributed uniformly among all functions mapping $\mathbf{Z}_n$ into $\mathbf{Z}_n$, then it is possible to prove that the system is semantically secure, unless the RSA assumption fails.

Using cryptographic hash function instead of random functions, one expects the security to remain the same, but this is no more a theorem.

A variant of this scheme (more efficient in terms of length of the encoding, and also satisfying a stronger definition of security — if h is random) is now a standard.

It is possible to provably achieve semantic security by only relying on the RSA assumption, but no really efficient way of doing so is known.

Signature

Signature schemes are cryptographic protocols whose purpose is to let users make a binding committment to the content of a certain message.

As for encryption, the most interesting case is when each user has a public key and a private key.

A signature scheme is specified by a key generation algorithm, a signing algorithm and a verification algorithm.

Components

- The signing algorithm takes in input a message and the secret key and outputs the *signature* of the message.
- The verification algorithm takes in input a message, an alleged signature and a public key and checks whether the signature is correct or not.

The desired property of the protocol is that a user that does not have the secret key should not be able to sign any document.

Simple RSA-based System

Using RSA, we can use standard key generation, and then

- $Sign((d,n),m) = RSA_{d,n}(m)$.
- $Verify((e,n),m,s)$ checks that $m = RSA_{e,n}(s)$.

That is, the signature is a "decryption" of the message. Since we assume that without the secret key it is not possible to do decryption, this system should be secure.

Definition of Security

However one has to be careful on what exactly is the definition of security.

After a user has signed a few documents, other users have seen those signatures and could use them to forge signatures of new documents. This is a standard attack to wich that implementation fails.

After we see the signatures of m_1 and of m_2, it is not hard to forge a signature for $m_1 m_2 \pmod n$.

Hash-and-Decrypt

Suppose that we have again a random function h. Then consider the scheme:

- $Sign((d,n),m) = RSA_{d,n}(h(m))$.
- $Verify((e,n),m,s)$ checks that $h(m) = RSA_{e,n}(s)$.

If h is random, the scheme is secure.