CS276 Cryptography	**Spring 2004**

Lecture 12 (1) : 3.3.04

Lecturer: David Wagner — *Scribe: Krishnendu Chatterjee*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

12.1 Public key cryptography

Today we will focus on public key cryptography. In the symmetric key encryption setting we assumed that Bob and Alice met beforehand and share a secret key K. The key K is used for encryption and decryption. The assumption however is not always reasonable, e.g., every user in the web cannot share a secret key with Amazon. Hence we are going to relax the assumption that there is a secret shared key between Bob and Alice. Instead we have the following setting: Bob has two keys — a public key, pk, and a private key (or *trapdoor information*) tk. Bob publishes the public key, pk, and remembers the private key, tk. Hence the public key is known to everyone. Our requirement is as follows: anyone knowing pk can encrypt a message but without knowing tk decrypting the message should not be possible. But Bob with the private key tk can easily decrypt any message. The situation is described in Fig 12.1. This was proposed in the Diffie-Hellman paper and the fact that public key cryptography is possible has been one of the major achievements in the field of cryptography.

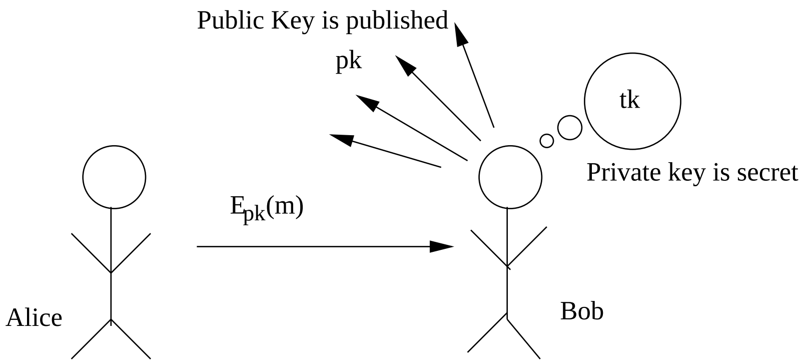

Figure 12.1: The public key cryptography setting

12.1.1 Requirements and Trapdoor functions

We now specify the requirements of the key generation, encryption and decryption algorithm and define *trapdoor functions*.

- Key Generation: $(pk, tk) \leftarrow G()$, where G is a randomized polynomial time algorithm.
- Encryption and Decryption Algorithm: $E_{pk}(m)$ and $D_{tk}(c)$ such that we have $D_{tk}(E_{pk}(m)) = m$ and the algorithm E and D are polynomial time algorithm.

- TRAPDOOR FUNCTIONS: Informally, a trapdoor function f satisfy the the following property: given x it is easy to compute $f(x)$, but it is hard to compute $f^{-1}(x)$ without knowing the trapdoor information. But given the trapdoor information it is easy to compute $f^{-1}(x)$. For our purpose we will consider trapdoor permutation as we require the inverse to be unique. Note the difference from a one-way-function is that whereas for one-way function the inverse is always hard to compute for trapdoor function it is easy to compute given the trapdoor information.

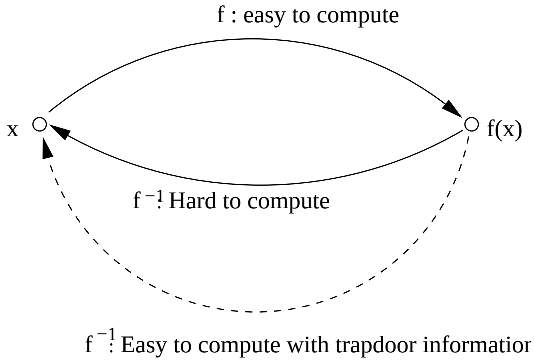

Figure 12.2: Trapdoor permutation

Definition 12.1 (FAMILY OF TRAPDOOR PERMUTATION AND SECURITY) *A family of trapdoor permutation is a triple of algorithms (G, F, I) such that*

1. *$(pk, tk) \leftarrow G()$— G outputs a pair of keys: the public key pk and trapdoor information or private key tk. G is a randomized polynomial time algorithm.*

2. *For every pk, for every x we have $f(x) = F_{pk}(x)$ is bijective and can be computed in polynomial time. In other words, F is a family of functions and for every pk we get a bijective function. Moreover, F is a polynomial time algorithm.*

3. *For every tk, for every x we have $f^{-1}(x) = I_{tk}(x) = F_{pk}^{-1}(x)$ i.e., $I_{tk}(F_{pk}(x)) = x$ and I is a polynomial time algorithm.*

4. *(t, ϵ)-Security: The family of trapdoor function (G, F, I) is (t, ϵ)-secure if for all algorithm A with running time at most t we have*

$$Adv\ A = Pr_{(pk,tk)\leftarrow G(), x\leftarrow \mathcal{M}}[A(pk, F_{pk}(x)) = x] \leq \epsilon$$

   where $\mathcal{M}$ is the domain of x.

A possible use of the family of trapdoor permutation for public key cryptography is as follows: Bob uses G to generate (pk, tk) and publishes pk. Alice uses F_{pk} to encrypt and Bob uses I_{tk} to decrypt.

Notational Shorthands: By a little abuse of notation we will often use the following shorthand:

$$(f, f^{-1}) \leftarrow G()$$

$$\text{Adv } A = Pr_{(f,f^{-1})\leftarrow G()}[A(f, f(x)) = x]$$

12.2 RSA

In this section we will discuss about the RSA scheme for public key cryptography. The scheme is as follows:

- G: pick two large primes p and q at random such that $p \neq q$ and $\gcd(p-1,3) = 1$ and $\gcd(q-1,3) = 1$. Compute $N = p \cdot q$ and $\phi(N) = (p-1)(q-1)$ where ϕ is the Euler Totient function. We now argue that G is a polynomial time procedure.

  Picking random primes

  - there is a polynomial time algorithm to check if a number is prime.
  - there are approximately $\frac{1}{\ln N}$ primes between 2 and N.

  Given the above two facts, we pick numbers at random and check if they are prime. The expected number of iterations to get a prime is $O(\ln N)$. Hence we can generate p and q in randomized polynomial time and then computing N and $\phi(N)$ can also be achieved in polynomial time. Hence we have $(N, \phi(N)) \leftarrow G()$.

- $f(x) = x^3 \mod N$
- $f^{-1}(y) = y^d \mod N$ where $d = 3^{-1} \mod \phi(N)$.
- Since we have polynomial time multiplication and exponentiation procedure the above function f and f^{-1} can be computed in polynomial time. Moreover using Lagrange's theorem we have the following result:

$$f^{-1}((f(x)) \equiv x^{3d} \mod N \equiv x^{3d \mod \phi(N)} \mod N \equiv x^1 \mod N \equiv x \mod N$$

- *Security*: RSA is not secure if factoring is easy (polynomial time computable). This is because if factoring is easy then it is easy to compute the trapdoor information $\phi(N)$ given N. It is an open problem whether RSA is secure if factoring is hard. Instead we make the RSA assumption that RSA is secure. This helps us building complicated protocols with the assumption that RSA is secure.

12.3 Another trapdoor function: Rabin's function

We will now describe another trapdoor function as defined by Rabin. The key generation algorithm G is same as in RSA. So the public key is $N = pq$ and the trapdoor information is p and q. We define the encryption and decryption algorithm below.

- ENCRYPTION: $f(x) = x^2 \mod N$
- DECRYPTION: $f^{-1}(x) = x^{1/2} \mod N$

Note that Rabin's function is different from RSA for the following reason:

1. 2 is not relatively prime to $p-1$ or $q-1$ where p and q are large prime numbers, i.e., $\gcd(p-1,2) = \gcd(q-1,2) = 2$.
2. We do not have a trapdoor permutation as both x and $-x$ are mapped to the same element, namely, $x^2 \mod N$.

We will show that if factoring is hard then Rabin's trapdoor function is secure. Note that to find the inverse of Rabin's function given a we want to find x such that $x^2 \equiv a \mod N$.

Lemma 12.2 *Let p and q be two prime number such that $p \neq q$ and $N = p \cdot q$. Then 1 has four square roots in $(\mathbb{Z}/N\mathbb{Z})^*$.*

Proof: Clearly $1 \mod N$ and $-1 \mod N$ are two roots of the equation $x^2 \equiv 1 \mod N$. We will show that there are two more roots, namely w and $-w$. Consider the equation $x^2 \equiv 1 \mod p$. There are two roots namely $w_1 \equiv 1 \mod p$ and $w_2 \equiv -1 \mod p$. Similar argument hold for the equation $x^2 \equiv 1 \mod q$.

$$\begin{array}{rcrl} x^2 & \equiv & 1 & \mod p \\ w_1 & \equiv & 1 & \mod p \\ w_2 & \equiv & -1 & \mod p \end{array} \qquad \begin{array}{rcrl} x^2 & \equiv & 1 & \mod q \\ w_3 & \equiv & 1 & \mod q \\ w_4 & \equiv & -1 & \mod q \end{array}$$

Hence we will get four roots of the equation $x^2 \equiv 1 \mod N$ from the combination of w_1, w_2, w_3 and w_4. ■

A generalization of the above Lemma is as follows. The following Lemma can be proved using *Chinese Remainder Theorem.*

Lemma 12.3 *For every $y \in (\mathbb{Z}/N\mathbb{Z})^*$ we have y has either 0 or 4 square roots. Moreover, there are $N/4$ elements y that has 4 square roots.*

A pictorial view of the above Lemma is shown in Fig 12.3. In the next half of the lecture we will prove that if factoring is hard then it is hard to compute the inverse of function f. Formally, we will prove the following theorem.

Theorem 12.4 *If factoring is hard, then $Pr_{(N,(p,q))\leftarrow G(),x\leftarrow(\mathbb{Z}/n\mathbb{Z})^*}[A(N, x^2 \mod N) = x]$ is small.*

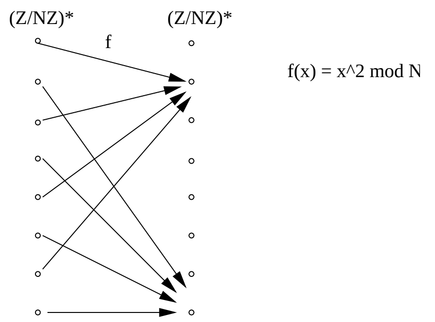

Figure 12.3: The function mapping $f(x) = x^2 \mod N$ from $(\mathbb{Z}/N\mathbb{Z})^*$ to $(\mathbb{Z}/N\mathbb{Z})^*$