Cryptography

Public Key Cryptosystems

Professor: Marius Zimand

We have seen one example of a PKC - RSA.

We discuss next general aspects of PKCs, and see other examples.

General concept of a public-key cryptosystem (PKC)

The Concept:

- M - a set of messages.
- K - a set of keys.
- G an algorithm that generates random keys in K, of a given size.
- For each key $k \in K$, there is an encryption function E_k and a decryption function D_k.

Requirements:

1. $D_k(E_k(m)) = m$, for every $m \in M, k \in K$.

   It is also desirable to have $E_k(D_k(m)) = m$, for every $m \in M, k \in K$ (for digital signatures).

2. For every $k \in K, m \in M$, $E_k(m)$ and $D_k(m)$ are easy to compute (formally, "easy" means in "polynomial-time".

3. For almost every $k \in K$, if someone knows E_k, it is infeasible to determine D_k.

RSA satisfies these properties:

- K is the set of tuples $k = (p, q, e, d, n)$, with p, q large primes, $n = p \cdot q$ and $e \cdot d = 1 (\text{mod } (p-1)(q-1))$.
- M is the set of numbers $m < n$.
- $E_k(m) = m^e (\text{mod } n)$.
- $D_k(c) = c^d (\text{mod } n)$.

Requirements (2) and (3) are based on the existence of so called *trapdoor one-way functions.* These are functions that are easy to calculate and hard to invert unless we are in the possession of an extra piece of information (the trapdoor).

For RSA, the underlying one-way function is:

on input e, n, m, calculate the output $c = m^e (\text{mod } n)$.

This is easy to do, but the inverse (finding m from c, e and n) is unfeasible (unless you know the trapdoor information p and q).

As we have seen the hardness of the above problem is related to the hardness of the FACTORING problem (i.e., factor a large number n which is the product of two large primes).

How does a PKC work

- each user (e.g., Alice) generates a key k.
- the user makes E_k public and keeps D_k secret.
- Everyone wishing to send message m to the user will encrypt it as $c = E_k(m)$ (using the public E_k).
- The user will calculate $D_k(c) = m$ (by Requirement (1)).

Many PKCs can also be used for digital signatures. Let's say that Alice wants to send a signed message to Bob.

- Alice has E_{kA}, D_{kA}.
- Bob has E_{kB}, D_{kB}.
- Alice wants to encrypt and send message m.
- Alice computes $E_{kB}(D_{kA}(m))$.
- Bob applies D_{kB} and next E_{kA} and retrieves m.
- Moreover Bob knows that the message was sent by Alice, because only she could have used the right D_{kA} (assuming that most random messages are meaningless).
- Also, Alice cannot deny that she sent the message.

Besides FACTORING, there are a few other problems that people believe are hard that have been used to build PKCs.

Discrete Log Problem (DLP)

- Input: q - prime number, α - primitive root of q, y - a residue mod q.
- Goal: Find k such that $\alpha^k = y(\text{mod } q)$. (In other words, find the position of y in the huge list $\{\alpha, \alpha^2, \ldots, \alpha^{q-1}\}$.

In short, DLP is: given $\alpha^x(\text{mod } q)$, find x.

Example

14 is a primitive root of 19.

The powers of 14 (mod 19) are in order:

14 6 8 17 10 7 3 4 18 5 13 11 2 9 12 16 15 1

For example $L_{14}(10) = 5 \bmod 19$, because $14^5 = 10 (\bmod\ 19)$.

A related problem is DHP - Diffie-Hellman Problem. The parameters q and α are the same as in DLP.

- Input: $a = \alpha^x (\bmod\ q)$ and $b = \alpha^y (\bmod\ q)$, where q and α are given.
- Goal: Find c such that $c = \alpha^{xy} (\bmod\ q)$.

In short, DHP is: given $\alpha^x (\bmod\ q)$ and $\alpha^y (\bmod\ q)$, find $\alpha^{xy} (\bmod\ q)$.

Of course if we can solve DLP then we can solve DHP. This can be formalized as follows: if we would live in a world where there is an oracle that gives the answer to DLP questions and we can ask the oracle such questions, then in that world we'd be able to solve DHP questions fast.

This is how: Let $\alpha^x (\bmod\ q)$ and $\alpha^y (\bmod\ q)$ be the input to the DHP. We ask the DLP oracle about $\alpha^x (\bmod\ q)$ and the oracle provides us x. Then we ask the DLP oracle about $\alpha^y (\bmod\ q)$ and the oracle provides us y. Now with x and y in our hand, we just calculate quickly $\alpha^{xy} (\bmod\ q)$.

What we have shown is a *polynomial-time reduction* from DHP to DLP. The notation is DHP $\leq^p$ DLP.

This implies that DHP is not harder than DLP. Actually people believe that DHP is about as hard as DLP.

Let us look at a very important protocol that is based on discrete logs. It's the Diffie-Hellman protocol for key exchange. This protocol was in the paper by Diffie and Hellman in which they introduced the concept of PK crypto.

Alice and Bob can only communicate via an open channel and at the end of the protocol they will share a secret number that can be used as a key in their future communications.

Diffie-Hellman key exchange protocol

Set-up. Alice and Bob choose q, a prime number, and α a primitive root of q. Recall that this means that $\{\alpha, \alpha^2, \ldots, \alpha^{q-1}\} (\text{mod } q) = \{1, 2, \ldots, q-1\}$.

The protocol

1. Alice chooses a random number $x_A < q$. Bob chooses a random number $x_B < q$.
2. Alice calculates $y_A = \alpha^{x_A} (\text{mod } q)$ (because of the hardness of DLP, y_A is "x_A in a locked box.")

   Bob calculates $y_B = \alpha^{x_B} (\text{mod } q)$ (because of the hardness of DLP, y_B is "x_B in a locked box.")
3. Alice sends y_A to Bob. Bob sends y_B to Alice.
4. Alice calculates $y_B^{x_A} (\text{mod } q)$. Bob calculates $y_A^{x_B} (\text{mod } q)$. These two numbers are equal and Alice and Bob can use it as a shared secret (for example, as a key in DES).

It works because:

$$y_B^{x_A} = (\alpha^{x_B})^{x_A} = \alpha^{x_A \cdot x_B} (\text{mod } q)$$

and

$$y_A^{x_B} = (\alpha^{x_A})^{x_B} = \alpha^{x_A \cdot x_B} (\text{mod } q).$$

The security of the Diffie-Hellman protocol is based on the hardness of DHP (which is related to the hardness of the DLP - the Discrete Log Problem).

Why: Eve knows q and α (these are public), and she also sees y_A and y_B. Finding the key $K = y_B^{x_A} = y_A^{x_B} (\text{mod } q)$ from this information amounts to solving the DHP, which Eve cannot do.

More formally this argument shows that with the help of an oracle which breaks the Diffie-Hellman protocol we can solve DHP. Thus DHP $\leq^p$ Breaking-Diffie-Hellman-Protocol.

Let us look now at another PKC, the El Gamal PKC, which is based on the Discrete Log Problem assumption.

El Gamal public-key cryptosystem

key set-up:

Each party (say Bob) chooses the following parameters.

- p - large prime number.
- α - primitive root of p.
- $a \in \{1, 2, 3, \ldots, p-1\}$ - random.
- $\beta = \alpha^a (\text{mod } p)$ (”a” in a locked box).

$$(p, \alpha, \beta) - \text{ (Bob's public key).}$$

$$a - \text{ (Bob's secret key).}$$

encryption:

Choose a random $k \in \{1, \ldots, p-1\}$. The message is a number $x < p$.

$E_{public-key,k}(x) = (\alpha^k (\mathrm{mod}\ p), x \cdot \beta^k (\mathrm{mod}\ p)$. So the ciphertext is a pair of two numbers. The first one masks k, and the second one masks the message x.

decryption:

$D_{secret-key}(y_1, y_2) = y_2 \cdot (y_1^a)^{-1} (\mathrm{mod}\ p))$.

Let's check that decryption does the right job.

$$y_2 \cdot (y_1^a)^{-1} = x \cdot \beta^k \cdot (\alpha^{ak})^{-1} = x \cdot \alpha^{ak} \cdot (\alpha^{ak})^{-1} = x (\mathrm{mod}\ p) = x.$$

Observation. El Gamal encryption is randomized; it depends on random k. So the same x has many encryptions.

Example. $p = 43$, $\alpha = 3$ (3 is a primitive root of 43).

Alice chooses $a = 7$ as her secret key.

$\beta = 3^7 = 37 (\mathrm{mod}\ 43)$.

Let Bob encrypt a plaintext message $x = 14$. Bob picks a random $k = 26$ and computes:

$$y_1 = 3^{26} = 15 (\mathrm{mod}\ 43), y_2 = 37^{26} \cdot 14 = 31 (\mathrm{mod}\ 43).$$

The ciphertext is $(15, 31)$ which he sends to Alice.

To decrypt the ciphertext $(15, 31)$, Alice computes

$$31 \cdot (15^7)^{-1} = 14 (\mathrm{mod}\ 43).$$

Since the message is < 43, Alice knows that the message is 14.

Security is based on the Discrete Log Problem.

Informal discussion on the security of El Gamal PKC.

Eve cannot find k: she has some info about k which is $\alpha^k \pmod p$, but finding k from this info means solving the DLP.

Eve cannot find a: she only has $\beta = \alpha^a \pmod p$, which is "a in a box."

Without knowing k and a, it seems that she cannot decrypt.

We can actually prove that El Gamal is secure against ciphertext-only attacks provided that the DHP is hard.

Theorem 1 *Assume that DHP is hard (Formally: DHP cannot be done in polynomial time).*

Then "Breaking El Gamal" is hard (Formally: there is no polynomial-time algorithm algorithm that given an arbitrary El Gamal ciphertext calculates the corresponding plaintext).

Proof. We show that DHP is polynomial-time reducible to "Breaking El Gamal" (which means finding an El Gamal ciphertext from its plaintext). So let $\mathcal{O}$ be an oracle with the following property: if we give the oracle the public key and an arbitrary El Gamal ciphertext, the oracle will give us back the corresponding plaintext. We show that using $\mathcal{O}$ we can efficiently (in polynomial time) solve DHP. From this it follows under our assumption that "Breaking El Gamal" cannot be done in polynomial time.

Thus let $\mathcal{O}$ be an oracle that does the El Gamal decryption. This means that if we give the oracle $\mathcal{O}$ the tuplet of numbers $(p, \alpha, \beta, y_1, y_2)$, the oracle will give us back a number x with the property that $y_1 = \alpha^k$ (for some k) and $y_2 = x \cdot \beta^k$.

Let's see how using this oracle $\mathcal{O}$, we can solve the DHP. Let the input for DHP be α^a and α^k. We want to find α^{ak} (all the exponentials are mod p).

We give the oracle $\mathcal{O}$ the tuplet $(p, \alpha, \alpha^a, \alpha^k, y_2)$, where y_2 is some random residue mod p. Then $\mathcal{O}$ delivers us a number x as above. Now observe that $y_2 \cdot x^{-1} = \alpha^{ak}$, which is just what we wanted.

Deeper look at the security of PKC

Semantic security

We want a PKC to be *semantically secure.*

I will not formalize this notion, but essentially, an encryption system is *semantically secure* if whatever an adversary can compute in polynomial time about the plaintext given the ciphertext, the adversary could also compute without the ciphertext.

So, having the ciphertext does not help at all a polynomial-time adversary.

It has been shown that semantic security is equivalent with the following notion of IND-security (where IND comes from *indistinguishability*). This notion of security is defined using a game between two parties, Alice and Bob.

Let E_k be an encryption function which is given to both Alice and Bob.

Bob chooses two messages m_1 and m_2.

Alice picks one of them, encrypts it, and gives Bob the ciphertext c. Thus Bob knows that

$$c = E_k(m_1) \text{ or } c = E_k(m_2).$$

and he wants to find whether m_1 or m_2 was encrypted.

The encryption scheme is IND-secure if in polynomial time Bob cannot tell which of the two messages was encrypted with probability significantly larger than 1/2.

Let's now see that RSA is not semantically secure (equivalently, not IND-secure).

We need to show that Bob can distinguish "c is an encryption of m_1" from "c is an encryption of m_2."

What Bob does is to just RSA-encrypt m_1, by computing $c' = m_1^e (\text{mod } n)$. This of course can be done in polynomial time. Then if $c' = c$, it means that c is an encryption of m_1, otherwise it is an encryption of m_2.

Non-malleability

Informally, an encryption system is *non-malleable* if given a ciphertext c of a message m it is impossible to determine in polynomial-time a valid encryption of another message m' related to m.

For example if c is an encryption of the message ”\$m”, the adversary should not be able to find the ciphertext for the message ”\$2m”.

RSA is malleable.

The following example shows this.

Let us fix m. Then $c = E(m) = m^e (\text{mod } n)$.

Take $c' = 2^e \cdot c (\text{mod } n)$.

Notice that

$$c' = 2^e \cdot c = 2^e \cdot m^e = (2m)^e (\text{mod } n).$$

Thus c' is an encryption of the message $m' = 2m$. So, if Eve has an encryption of m she can easily compute an encryption of the related message $2m$.

Thus RSA is not

- semantically secure
- non-malleable.

To make a system with these security properties we use a padding scheme.

Not every padding scheme works. Some do not fix the problem or even make the system more vulnerable.

The recommended padding scheme is OAEP (Optimal Asymmetric Encryption Padding)- introduced by Bellare and Rogaway.

OAEP

Let f be any k-bit to k-bit trapdoor one-way permutation.

For example, $k = 1024$ and we take f to be the RSA function $f(m) = m^e (\mod n)$.

Let s and t be two positive integers such that a work effort of 2^s or 2^t is impossible (for example $s = t = 128$, or larger).

Let us take $n = k - s - t$ (in our example $n = 1024 - 128 - 128 = 768$).

Then take two functions

$$G : \{0,1\}^s \to \{0,1\}^{n+t} \tag{1}$$

$$H : \{0,1\}^{n+t} \to \{0,1\}^s \tag{2}$$

The functions G and H should look like random functions and in practice they are obtained from hash functions such as SHA-1 and MD5 (which we will see later on).

OAEP encryption

Let m be a message of n bits.

Then

$$E(m) = f((m0^t \oplus G(R)) \ \ || \ \ (R \oplus H(m0^t \oplus G(R))),$$

where

- $m0^t$ means m followed by t zeroes.
- R is a random binary string of length s,
- $\oplus$ is bitwise XOR

- || denotes concatenation.

OAEP decryption

The ciphertext is

$$E(m) = f((m0^t \oplus G(R)) \;\; || \;\; (R \oplus H(m0^t \oplus G(R))).$$

Using the decryption function f^{-1} (for example, if f is the RSA-encryption, then f^{-1} is the RSA-decryption function), we compute

$$(m0^t \oplus G(R)) \;\; || \;\; (R \oplus H(m0^t \oplus G(R)).$$

We now know

$$T = (m0^t \oplus G(R)) \text{ and } S = (R \oplus H(m0^t \oplus G(R)).$$

We can now compute $H(T)$ and recover $R = H(T) \oplus S$.

Now we compute $G(R)$ and recover $m0^t = T \oplus G(R)$.

Now we remove the last t zeroes and we have m.

Note: if the number of zeroes at the end is not t something has gone wrong and we reject the ciphertext.

The following fact has been proved.

Fact 2 *If G and H are random functions, then RSA + OAEP is*

- *semantically secure against chosen ciphertext attacks*
- *non-malleable*