Chapter 6

Introduction to Public key cryptography

6.1 What is asymmetric cryptography?

6.1.1 Distribution of keys

As you have seen in the previous chapters, symmetric key cryptography assumes that the communicating parties share a common secret key. If we assume a trusted authority creating such a symmetric key, we need some means of distributing a copy of the secret key to each communicating party. There are a number of possible ways of doing this. At first possible solution would be to distribute the secret key through *physical distribution.* Using trusted couriers and physically locked and secured dispatch cases, keys can be physically transported to a desired location. In fact, until 30-40 years ago, this was the only means of distributing keys. There are obviously a number of huge drawbacks, such as the cost of the system and the fact that the security now depends on couries not missbehaving. There are other problems when symmetric communication keys are distributed long before communication takes part.

As an example, assume that n users are connected in a network and any two of them may want to communicate with each other in a secure manner. This would require each user to securely store $n-1$ different symmetric keys (one for each other user), resulting in a total of $n(n-1)/2$ keys. If the network is connecting 2000 university students, then there will be roughly 2 million different keys that need to be generated and securely distributed and stored within the network. This creates a huge *key management* problem with questions like; How do you add add a new user to the system? What if a user's key is compromised? How long should a key be considered valid and how should we refresh them?

A better solution to the key distribution problem is obtained if we use symmetric *key distribution protocols.* Here we need to assume access to a trusted third party (TTP). Each user has a unique secret key shared with the TTP. When two users would like to communicate, they establish a shared secret key, usually called a *session key,* by interacting with the TTP. Typically, a user requests the TTP to generate the the session key and securely communicate copies of the session key to the two communicating parties. Once they have received their copies of the session key, they can use it for secure communication. Using a key distribution protocol of the above kind will provide many advantages compared to physical distribution. However, there are still drawbacks that can be serious problems in certain situations. For example, we need access and trust to a TTP and we still need to distribute one key shared with the TTP for each user.

In public-key cryptography, we have an extremely elegant solution to the above considered problem. We remove the assumption of a shared secret key between the two communicating parties. Instead, we assume two *different* keys, one for encryption of a plaintext message and another for decryption of a ciphertext message. Furthermore, we allow the encryption key to be *public*! This allows anyone with access to the public encryption key to send an encrypted plaintext to the receiver holding the secret decryption key.

At first glance, it looks counter-intuitive that one can actually do this. You would think that having an encryption key that is public would allow anyone to decrypt a ciphertext as well. But as we will see, this is not at all the case.

6.1.2 One-way functions and trapdoor one-way functions

In order to explain how public-key cryptography works, we need to start discussing the concepts of one-way functions and trapdoor one-way functions.

We give informal definitions of these notions.

Definition 6.1. *A* one-way function $f(x)$ *is a function from a set* $\mathcal{X}$ *to a set* $\mathcal{Y}$ *such that* $f(x)$ *is easy to compute for all* $x \in \mathcal{X}$*, but for "essentially all" elements* $y \in \mathcal{Y}$ *it is "computationally infeasible" to find any* $x \in \mathcal{X}$ *such that* $f(x) = y$.

Due to the informal definition, we need to clarify and discuss the terms used above.

- The terms "easy" and "computationally infeasible" can be rigourously defined, but at this moment it is more useful to think of these expressions in a intuitive way. The phrase stating that f is easy to evaluate means that the number of operations is limited and can be done quickly, for example within a second on a simple computer. On the other hand, "computationally infeasible" means that the required number of operation is so large that it is just impossible to perform them all. For example, we can assume that it menas that it would take more than 100 years for all existing computers in the world to jointly do these operations. Note that even if it is computationally infeasible to find a solution, it still exists!

- The phrase "essentially all elements" relates to the fact that there will always be a small fraction of $\mathcal{Y}$ for which it is easy to find an $x \in \mathcal{X}$ such that $f(x) = y$. For example, if you pick arbitrary values $x \in \mathcal{X}$, compute $y = f(x)$ and store the pairs x, y in a memory, you may easily invert f for the values in $\mathcal{Y}$ that are stored in memory. The key point here is that this fraction of easily inverted elements should be very small. Expressed differently, if you pick a $y \in \mathcal{Y}$ *at random*, the probability that you can (computationally) invert it should be very small. As a consequence, this means that the sets $\mathcal{X}$ and $\mathcal{Y}$ need to be large!

Example 6.1. *This is an example demonstrating the concept on one-way functions. Let $\mathcal{X} = \mathcal{Y} = \mathbb{Z}_{29}$ and consider the function $f : \mathcal{X} \mapsto \mathcal{Y}$ described by*

$$f(x) = 2^x \bmod 29.$$

a) *Roughly, how many operations do you need need to calculate $f(5) = 2^5$?*

One way would be to calculate $2^2 = 2 \cdot 2 = 4$, then $2^4 = 4 \cdot 4 = 16$ and finally multiplying with 2, i.e., $2^5 = 16 \cdot 2 = 3$. This procedure required 3 multiplications.

b) *Roughly, how many operations do you need need to find an $x \in \mathcal{X}$ such that $f(x) = 5$?*

One way would be to calculate $2^1 = 2, 2^2 = 4, 2^3 = 8, \ldots$ until we reach $2^x = 5$. We find $\ldots, 2^{22} = 5$ and this procedure resulted in 21 multiplications.

Note that this is not an example of a one-way function. The idea here was to demonstrate that there can be a large difference in the complexity of calculating $f(x)$ and its inverse!

Example 6.2. *A function that looks like a one-way functions is the function $f : \mathbb{Z}_n \mapsto \mathbb{Z}_n$, with*

$$f(x) = x^3 \bmod n,$$

where n is the number

$$n = 27997833911221327870829467638722601621070446786955428537560009929326128400107609345671052955360856061822351910951365788637105954482006576775098580557613579098734950144178863178946295187237869221823983.$$

Later on, we will see that it is possible to evaluate $f(x)$ for any $x \in \mathbb{Z}_n$ efficiently. The number n given above is reffered to as the RSA-200 number as it has 200 decimal digits.

The second notion we need is the following.

Definition 6.2. *A* trapdoor one-way function *$f(x)$ is a one-way function $f : \mathcal{X} \mapsto \mathcal{Y}$ such that if one knows some specific information T, called the* trapdoor information*, then $f(x)$ is computationally easy to invert f, i.e., for any $y \in \mathcal{Y}$ it is easy to find a $x \in \mathcal{X}$ such that $f(x) = y$. For anyone without knowledge of the trapdoor information T, $f(x)$ is a one-way function.*

In fact, the function given in example 6.2 is a trapdoor one-way function. We said that the function looked like a one-way function. However, there is some secret information about this number that can be used. The number n is in fact the product of two large prime numbers. The trapdoor information T is in this case these two prime numbers. Knowing them, you can easily invert the f function. We will show later how this is done. Furthermore, note that even though you know the number n, you cannot (easily) compute the two prime numbers that are multiplied together to form n. On the other hand, if you start with two large primes p and q, you can easily multiply them together to form the number $n = pq$. If you want to know, the RSA-200 number above was factored in 2003 and the factors are

$$3532461934402770121272604978198464368671197400197625023649303468776121253679423200058547956528088349$$

and

$$7925869954478333033347085841480059687737975857364219960734330341455767872818152135381409304740185467$$

. It took time equivalent to more than 55 years on a single modern PC to factor this number.

In the area of public-key cryptography, we may encounter many different cryptographic primitives, such as digital signatures, electronic cash etc. The conceptually most simple onecould be a *public-key encryption scheme.* A rough definition could be as follows.

Definition 6.3. *A public-key encryption scheme is a set of encryption transformations $\{E_e : e \in \mathcal{K}\}$ and a set of decryption transformations $\{D_d : d \in \mathcal{K}\}$. For each $e \in \mathcal{K}$ there is a corresponding $d \in \mathcal{K}$ such that $D_d(E_e(M)) = M$, $\forall M$. Furthermore, after choosing such a pair (e, d), the* public key *e (or the public parameter) is made public, while the associated* secret key *d is kept secret. For the scheme to be secure, it must be computationally infeasible to compute d as well as computing $E_e^{-1}(C)$, knowing the public value e.*

A public-key encryption scheme can be constructed through a set of trapdoor one-way functions $\{f_i(x)\}$. We pick one such function. This is our public transformation E_e and its parameters are usually named the public key. The corresponding trapdoor information T that is required to invert this function corresponds to our secret key.

It is important to note that the concepts brought forward in this section can only be realized under assumptions of computational limitations. This means in particular that we can never have unconditional security in a public-key cryptosystem. There is always some assumption that is required to be true for the scheme to be secure. ofter, these assumptions are of the form that we assume that the opponent cannot solve a certain (computationally difficult but solvable) problem. For example, this could be to factor a given very large number.

In order to fully understand these concepts, we need an explicit construction. In the next section we describe the most common construction, the RSA public-key encryption scheme.

6.2 The RSA public-key encryption scheme

The RSA public-key encryption scheme was proposed in 1977 by Rivest, Shamir and Adleman. It is probably the most well known construction of a cryptographic primitive.

Definition 6.4. *The* RSA public-key encryption scheme *works as follows. Let $n = pq$, where p and q are two large primes. Let $\mathcal{M} = \mathcal{C} = \mathbb{Z}_n$. Pick a number e relatively prime to $\phi(n)$ and calculate a number d such that $ed = 1 \bmod \phi(n)$. The public key is the two numbers (n, e) and the (public) encryption transformation $E(M)$ is*

$$E(M) = M^e \bmod n.$$

The secret key is the number d (as well as p, q and $\phi(n)$) and the secret decryption transformation $D(C)$ is

$$D(C) = C^d \bmod n.$$

To start with, we need to verify that this works as an encryption scheme, i.e., that decrypting a ciphertext returns the encrypted plaintext. We have

$$D(C) = C^d = (M^e)^d = M^{ed} \bmod n.$$

Now we note that $ed = 1 \bmod \phi(n)$, which means that we can write

$$ed = 1 + t \cdot \phi(n),$$

for some integer t. S we can continue

$$D(C) = M^{ed} = M^{(1+t\cdot\phi(n))} = M \cdot M^{t\cdot\phi(n)} \bmod n.$$

From Euler's formula we know that $x^{\phi(n)} = 1$ for any $x \in \mathbb{Z}_n^*$. So assuming that M is invertible we have

$$D(C) = M \cdot M^{t\cdot\phi(n)} = M \cdot 1 \bmod n.$$

If M is not invertible in $\mathbb{Z}_n^*$ one has to proceed differently, using the CRT. We leave for the reader to check that the same property holds also in this case.

Example 6.3. *Let us illustrate the RSA encryption scheme through a simple (insecure) example. Let $p = 47$ and $q = 167$ be the two primes. Their product is $n = pq = 7849$. Knowing the primes we can compute $\phi(n) = (p-1)(q-1) = 7636$. Next, we must choose a value e such that $\gcd(e, \phi(n)) = 1$. We can check that $e = 25$ is such a value. When $\gcd(e, \phi(n)) = 1$ we know that $e = 25$ has a multiplicative inverse in $\mathbb{Z}_{\phi(n)}$. We use Euclidean algorithm and Bezout's lemma to find the inverse $d = 2749$, i.e., we have two integers $e = 25$ and $d = 2749$ such that $e \cdot d = 25 \cdot 2749 = 1 \bmod 7636$.*

At this point we have generated the parameters of our RSA encryption scheme and we would now publish our public key $(n, e) = (7849, 25)$. The secret key consists of the value $d = 2749$.

Anyone with access to our public key can now send us an encrypted plaintext message $M \in \mathbb{Z}_{7849}$, by calculating the ciphertext as $C = M^e \bmod n$. For example, assume that Bob is holding the secret decryption key and Alice would like to send an encrypted message $M = 2728$. She then computes the ciphertext C as

$$C = 2728^{25} = 2401 \bmod 7849.$$

When Bob receives the ciphertext C, he computes

$$M = C^d = 2401^{2749} \bmod n$$

and, as by magic, $M = 2401^{2749} = 2728 \bmod n$ so he recover the correct plaintext message.

As indicated above, this particular RSA encryption scheme is not secure. The public value n is clearly too small and can easily be factored.

The security of the RSA cryptosystem is based on the assumption that the function $E(M) = M^e \bmod n$ is a one-way function for anyone without access to the trapdoor information. Clearly, if we can factor the number n then we can compute $\phi(n)$ and d. Then we have broken that instance of the RSA cryptosystem since by knowing d we can invert $E(M)$ and obtain the plaintext that generated a fixed arbitrary ciphertext. We say that RSA relies on the *factoring problem.* This does not mean that breaking RSA is equivalent to solving a factorization problem. It is not known whether RSA can be broken without factoring n.

6.3 Implementation issues

As you have seen, RSA relies on the factoring problem. This means that we need to use a really large number n or otherwise the enemy will simply factor the number. A typical RSA modulus could have a length of, say, 1024 bits. With such large numbers, we need to examine how the different arithmetic operations are done.