Chapter 11

Public key encryption

11.1 Introduction

In this chapter, we consider again the basic problem of encryption. As a motivating example, suppose Alice wants to send Bob an encrypted email message, even though the two of them do not share a secret key (or even a key with some key distribution center as in Section **??**). Surprisingly, this can be done using a technology called **public-key encryption**.

The basic idea of public-key encryption is that the receiver, Bob in this case, runs a key generation algorithm G, obtaining a pair of keys:

$$(\mathrm{pk}, \mathrm{sk}) \xleftarrow{R} G().$$

The key pk is Bob's *public key*, and sk is Bob's *secret key*. As their names imply, Bob should keep sk secret, but may publicize pk.

To send Bob an encrypted email message, Alice needs two things: Bob's email address, and Bob's public key pk. How Alice reliably obtains this information is a topic we shall explore later; for the moment, one might imagine that this information is placed by Bob in some kind of public directory to which Alice has read-access.

So let us assume now that Alice has Bob's email address and public key pk. To send Bob an encryption of her email message m, she computes the ciphertext

$$c \xleftarrow{R} E(\mathrm{pk}, m).$$

She then sends c to Bob, using his email address. At some point later, Bob receives the ciphertext c, and decrypts it, using his *secret key*:

$$m \leftarrow D(\mathrm{sk}, c).$$

A few points deserve further discussion:

- Once Alice obtains Bob's public key, the only interaction between Alice and Bob is the actual transmission of the ciphertext from Alice to Bob: no further interaction is required. In fact, we chose encrypted email as our example problem precisely to highlight this feature, as email delivery protocols do not allow any interaction beyond delivery of the message.

- As we will discuss later, the same public key may be used many times. Thus, once Alice obtains Bob's public key, she may send him encrypted messages as often as she likes. Moreover, other users besides Alice may send Bob encrypted messages using the same public key pk.

- As already mentioned, Bob may publicize his public key pk. Obviously, for any secure public-key encryption scheme, it must be hard to compute sk from pk, since anyone can decrypt using sk.

11.1.1 Two further examples

One can do a lot of neat things with public-key encryption. We give to more example applications.

Sharing encrypted files

On the Windows operating system, one can store encrypted files in a public directory to which others have read access. The owner of the file can selectively allow others to read the unencrypted contents of the file. This is done using a combination of public-key encryption and ordinary, symmetric cipher.

Here is how it works. Alice encrypts a file f under a key k, using an ordinary, symmetric cipher. The resulting ciphertext c is stored on disk. If Alice want to grant Bob access to the contents of the file, she encrypts k under his public key; that is, she computes $c' \xleftarrow{R} E(\mathrm{pk}, k)$, where pk is Bob's public key. The ciphertext c' is then stored on the disk near the ciphertext c. In fact, in the Windows file system implementation, c' is stored in the file header, along with other meta-data associated with the file. Now when Bob wants to read the file f, he can decrypt c' using his secret key sk, obtaining k, using which he can decrypt c using the symmetric cipher. Of course, if things are set up properly, all Alice and Bob are really doing is pointing and clicking on some icons, with the operating system executing all the necessary encryption and decryption algorithms.

This scheme scales very nicely if Alice wants to grant access to f to a number of users. Only one copy of the encrypted file is stored on disk, which is very nice if the file is quite large (such as a video file). For each user that is granted access to the file, only an encryption of the key k is stored in the file header. Each of these ciphertexts is fairly small (on the order of a few hundred bytes), even if the file itself is very big.

Key escrow

11.2 Security against eavesdropping

We begin by defining the basic syntax and correctness properties of a public-key encryption scheme.

Definition 11.1. *A **public-key encryption scheme** $\mathcal{E} = (G, E, D)$ is a triple of algorithms: a **key generation algorithm** G, an **encryption algorithm** E, a **decryption algorithm** D.*

- *G is a probabilistic algorithm that is invoked as $(\mathrm{pk}, \mathrm{sk}) \xleftarrow{R} G()$, where* pk *is called a **public key** and* sk *is called a **secret key**.*

- *E is a probabilistic algorithm that is invoked as $c \xleftarrow{R} E(\mathrm{pk}, m)$, where* pk *is a public key (as output by G), m is a message, and c is a ciphertext.*

- *D is a deterministic algorithm that is invoked as $m \leftarrow D(\mathrm{sk}, c)$, where* sk *is a secret key (as output by G), c is a ciphertext, and m is either a message, or a special* reject *value (distinct from all messages).*
- *As usual, we require that decryption undoes encryption; specifically, for all possible outputs* $(\mathrm{pk}, \mathrm{sk})$ *of G, and all messages m, we have*

$$\Pr[D(\mathrm{sk},\ E(\mathrm{pk},\ m)\ ) = m] = 1.$$

- *Messages m are assumed to lie in some finite* ***message space*** *$\mathcal{M}$, and ciphertexts in some finite* ***ciphertext space*** *$\mathcal{C}$. We say that $\mathcal{E} = (G, E, D)$ is defined over $(\mathcal{M}, \mathcal{C})$.*

We next define the notion of semantic security for a public-key encryption scheme.

Attack Game 11.1 (semantic security). For a given public-key encryption scheme $\mathcal{E} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b **(**$b = 0, 1$**):**

- The challenger computes $(\mathrm{pk}, \mathrm{sk}) \xleftarrow{R} G()$, and sends pk to the adversary.
- The adversary computes $m_0, m_1 \in \mathcal{M}$, of the same length, and sends them to the challenger.
- The challenger computes $c \xleftarrow{R} E(\mathrm{pk}, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\mathrm{SSAdv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Note that in the above game, the events W_0 and W_1 are defined with respect to the probability space determined by the random choices made by the key generation and encryption algorithms, and the random choices made by the adversary. See Fig. 11.1 for a schematic diagram of Attack Game 11.1.

Definition 11.2 (semantic security). *A public-key encryption scheme $\mathcal{E}$ is* ***semantically secure*** *if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{SSAdv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Just to be clear, for our purposes, the term **cipher** refers to a **computational cipher** as defined in Section 2.3.1, which we may also call this a **symmetric cipher**, so as to distinguish it from a public-key encryption scheme.

11.2.1 Mathematical details

11.3 Encryption based on trapdoor function schemes and RSA

In this section, we show how to use a trapdoor function scheme (see Section 10.2) to build a semantically secure public-key encryption scheme. In fact, this scheme makes use of a hash function, and our proof of security works only when we model the hash function as a random oracle. We then present a concrete instantiation of this scheme, based on RSA (see Section 10.3).

Our encryption scheme is called $\mathcal{E}_{\mathrm{TF}}$, and is built out of several componets: