Chapter 11

Public key encryption

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 (nor do they share a secret key with some common third party). 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:

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

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 in Section 13.8. 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

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

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*:

$$m \leftarrow D(sk, c).$$

Public-key encryption is sometimes called **asymmetric encryption** to denote the fact that the encryptor uses one key, pk, and the decryptor uses a different key, sk. This is in contrast with *symmetric encryption*, discussed in Part 1, where both the encryptor and decryptor use the same key.

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 Two further example applications

Public-key encryption is used in many real-world settings. We give two more examples.

11.1.1 Sharing encrypted files

In many modern file systems, a user can store encrypted files to which other users 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 an 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 the file system. If Alice wants to grant Bob access to the contents of the file, she encrypts k under Bob's public key; that is, she computes $c_{\mathrm{B}} \xleftarrow{\mathrm{R}} E(pk_{\mathrm{B}}, k)$, where pk_{B} is Bob's public key. The ciphertext c_{B} is then stored on the file system near the ciphertext c, say, as part of the file header, which also includes file metadata (such as the file name, modification time, and so on). Now when Bob wants to read the file f, he can decrypt c_{B} using his secret key sk_{B}, obtaining k, using which he can decrypt c using the symmetric cipher. Also, so that Alice can read the file herself, she grants access to herself just as she does to Bob, by encrypting k under her own public key pk_{A}.

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 the file system, which is good 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.

11.1.2 Key escrow

Consider a company that deploys an encrypted file system such as the one described above. One day Alice is traveling, but her manager needs to read one of her files to prepare for a meeting with an important client. Unfortunately, the manager is unable to decrypt the file because it is encrypted and Alice is unreachable.

Large companies solve this problem using a mechanism called **key escrow**. The company runs a key escrow server that works as follows: at setup time the key escrow server generates a secret key sk_{ES} and a corresponding public key pk_{ES}. It keeps the secret key to itself and makes the public key available to all employees.

When Alice stores the encryption c of a file f under a symmetric key k, she also encrypts k under pk_{ES}, and then stores the resulting ciphertext c_{ES} in the file header. Every file created by company employees is encrypted this way. Now, if Alice's manager later needs access to f and Alice

is unreachable, the manager sends c_{ES} to the escrow service. The server decrypts c_{ES}, obtaining k, and sends k to the manager, who can then use this to decrypt c and obtain f.

Public-key encryption makes it possible for the escrow server to remain offline, until someone needs to decrypt an inaccessible file. Also, notice that although the escrow service allows Alice's manager to read her files, the escrow service itself cannot read Alice's files, since the escrow service never sees the encryption of the file.

11.2 Basic definitions

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 efficient algorithms: a **key generation algorithm** G, an **encryption algorithm** E, a **decryption algorithm** D.*

- *G is a probabilistic algorithm that is invoked as $(pk, sk) \xleftarrow{R} G()$, where pk is called a **public key** and sk is called a **secret key**.*
- *E is a probabilistic algorithm that is invoked as $c \xleftarrow{R} E(pk, m)$, 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 $m \leftarrow D(sk, c)$, 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 (pk, sk) of G, and all messages m, we have*

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

- *Messages 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. We stress that this notion of security only models an eavesdropping adversary. We will discuss stronger security properties in the next chapter.

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 $(pk, sk) \xleftarrow{R} G()$, 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 $c \xleftarrow{R} E(pk, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

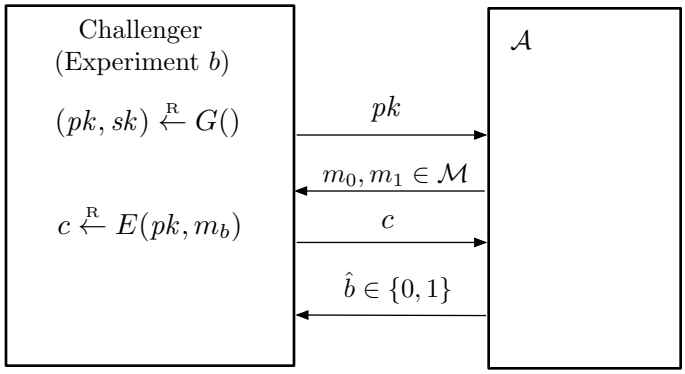

Figure 11.1: Experiment b of Attack Game 11.1

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

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

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 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ is negligible.*

As discussed in Section 2.3.5, Attack Game 11.1 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0,1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.3.5 (namely, (2.11)) applies here as well:

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]. \tag{11.1}$$

11.2.1 Mathematical details

We give a more mathematically precise definition of a public-key encryption scheme, using the terminology defined in Section 2.4.

Definition 11.3 (public-key encryption scheme). *A **public-key encryption scheme** consists of a three algorithms, G, E, and D, along with two families of spaces with system parameterization P:*

$$\mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda} \quad \textit{and} \quad \mathbf{C} = \{\mathcal{C}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

such that

1. $\mathbf{M}$ *and* $\mathbf{C}$ *are efficiently recognizable.*

2. **M** *has an effective length function.*

3. *Algorithm G is an efficient probabilistic algorithm that on input λ, Λ, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, outputs a pair (pk, sk), where pk and sk are bit strings whose lengths are always bounded by a polynomial in λ.*

4. *Algorithm E is an efficient probabilistic algorithm that on input λ, Λ, pk, m, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $(pk, sk) \in \mathrm{Supp}(G(\lambda, \Lambda))$ for some sk, and $m \in \mathcal{M}_{\lambda,\Lambda}$, always outputs an element of $\mathcal{C}_{\lambda,\Lambda}$.*

5. *Algorithm D is an efficient* deterministic *algorithm that on input λ, Λ, sk, c, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $(pk, sk) \in \mathrm{Supp}(G(\lambda, \Lambda))$ for some pk, and $c \in \mathcal{C}_{\lambda,\Lambda}$, outputs either an element of $\mathcal{M}_{\lambda,\Lambda}$, or a special symbol* reject $\notin \mathcal{M}_{\lambda,\Lambda}$.

6. *For all $\lambda, \Lambda, pk, sk, m, c$, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $(pk, sk) \in \mathrm{Supp}(G(\lambda, \Lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, $m \in \mathcal{M}_{\lambda,\Lambda}$, and $c \in \mathrm{Supp}(E(\lambda, \Lambda; pk, m))$, we have $D(\lambda, \Lambda; sk, c) = m$.*

As usual, the proper interpretation of Attack Game 11.1 is that both challenger and adversary receive λ as a common input, and that the challenger generates Λ and sends this to the adversary before the game proper begins. The advantage is actually a function of λ, and security means that this is a negligible function of λ.

11.3 Implications of semantic security

Before constructing semantically secure public-key encryption schemes, we first explore a few consequences of semantic security. We first show that any semantically secure public-key scheme must use a *randomized* encryption algorithm. We also show that in the public-key setting, semantic security implies CPA security. This was not true for symmetric encryption schemes: the one-time pad is semantically secure, but not CPA secure.

11.3.1 The need for randomized encryption

Let $\mathcal{E} = (G, E, D)$ be a semantically secure public-key encryption scheme defined over $(\mathcal{M}, \mathcal{C})$ where $|\mathcal{M}| \geq 2$. We show that the encryption algorithm E must be a randomized, otherwise the scheme cannot be semantically secure.

To see why, suppose E is deterministic. Then the following adversary $\mathcal{A}$ breaks semantic security of $\mathcal{E} = (G, E, D)$:

- $\mathcal{A}$ receives a public key pk from its challenger.
- $\mathcal{A}$ chooses two distinct messages m_0 and m_1 in $\mathcal{M}$ and sends them to its challenger. The challenger responds with $c := E(pk, m_b)$ for some $b \in \{0, 1\}$.
- $\mathcal{A}$ computes $c_0 := E(pk, m_0)$ and outputs 0 if $c = c_0$. Otherwise, it outputs 1.

Because E is deterministic, we know that $c = c_0$ whenever $b = 0$. Therefore, when $b = 0$ the adversary always outputs 0. Similarly, when $b = 1$ it always outputs 1. Therefore

$$\mathrm{SSadv}[\mathcal{A}, \mathcal{E}] = 1$$

showing that $\mathcal{E}$ is insecure.

This generic attack explains why semantically secure public-key encryption schemes must be randomized. All the schemes we construct in this chapter and the next use randomized encryption. This is quite different from the symmetric key settings where a deterministic encryption scheme can be semantically secure; for example, the one-time pad.

11.3.2 Semantic security against chosen plaintext attack

Recall that when discussing symmetric ciphers, we introduced two distinct notions of security: semantic security, and semantic security against chosen plaintext attack (or CPA security, for short). We showed that for symmetric ciphers, semantic security *does not* imply CPA security. However, for public-key encryption schemes, semantic security *does* imply CPA security. Intuitively, this is because in the public-key setting, the adversary can encrypt any message he likes, without knowledge of any secret key material. The adversary does so using the given public key and never needs to issue encryption queries to the challenger. In contrast, in the symmetric key setting, the adversary cannot encrypt messages on his own.

The attack game defining CPA security in the public-key setting is the natural analog of the corresponding game in the symmetric setting (see Attack Game 5.2 in Section 5.3):

Attack Game 11.2 (CPA 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 $(pk, sk) \xleftarrow{R} G()$, and sends pk to the adversary.
- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots$, the ith query is a pair of messages, $m_{i0}, m_{i1} \in \mathcal{M}$, of the same length.

  The challenger computes $c_i \xleftarrow{R} E(pk, m_{ib})$, and sends c_i 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, then we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

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

Definition 11.4 (CPA security). *A public-key encryption scheme $\mathcal{E}$ is called* ***semantically secure against chosen plaintext attack****, or simply* ***CPA secure****, if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{CPAadv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Theorem 11.1. *If a public-key encryption scheme $\mathcal{E}$ is semantically secure, then it is also CPA secure.*

> *In particular, for every CPA adversary $\mathcal{A}$ that plays Attack Game 11.2 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger, there exists an SS adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{CPAadv}[\mathcal{A}, \mathcal{E}] = Q \cdot \mathrm{SSadv}[\mathcal{B}, \mathcal{E}].$$

Proof. The proof is a straightforward hybrid argument, and is very similar to the proof of Theorem 5.1. Suppose $\mathcal{E} = (G, E, D)$ is defined over $(\mathcal{M}, \mathcal{C})$. Let $\mathcal{A}$ be a CPA adversary that plays Attack Game 11.2 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger.

We describe the relevant hybrid games. For $j = 0, \ldots, Q$, Hybrid j is played between $\mathcal{A}$ and a challenger who works as follows:

$(pk, sk) \xleftarrow{R} G()$
Send pk to $\mathcal{A}$
Upon receiving the ith query $(m_{i0}, m_{i1}) \in \mathcal{M}^2$ from $\mathcal{A}$ do:
    if $i > j$
        then $c_i \xleftarrow{R} E(pk, m_{i0})$
        else $c_i \xleftarrow{R} E(pk, m_{i1})$
    send c_i to $\mathcal{A}$.

Put another way, the challenger in Hybrid j encrypts

$$m_{11}, \ldots, m_{j1}, \quad m_{(j+1)0}, \ldots, m_{Q0},$$

As usual, we define p_j to be the probability that $\mathcal{A}$ outputs 1 in Hybrid j. Clearly,

$$\mathrm{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = |p_Q - p_0|.$$

Next, we define an appropriate adversary $\mathcal{B}$ that plays Attack Game 11.1 with respect to $\mathcal{E}$:

First, $\mathcal{B}$ chooses $\omega \in \{1, \ldots, Q\}$ at random.

Then, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$: it obtains a public key pk from its own challenger, and forwards this to $\mathcal{A}$; when $\mathcal{A}$ makes a query (m_{i0}, m_{i1}), $\mathcal{B}$ computes its response c_i as follows:

if $i > \omega$ then
    $c \xleftarrow{R} E(pk, m_{i0})$
else if $i = \omega$ then
    $\mathcal{B}$ submits (m_{i0}, m_{i1}) to its own challenger
    c_i is set to the challenger's response
else // $i < \omega$
    $c_i \xleftarrow{R} E(pk, m_{i1})$.

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

The crucial difference between the proof of this theorem and that of Theorem 5.1 is that for $i \neq \omega$, adversary $\mathcal{B}$ can encrypt the relevant message using the public key.

For $b = 0, 1$, let W_b be the event that $\mathcal{B}$ outputs 1 in Experiment b of its attack game. It is clear that for $j = 1, \ldots, Q$,

$$\Pr[W_0 \mid \omega = j] = p_{j-1} \quad \text{and} \quad \Pr[W_1 \mid \omega = j] = p_j,$$

and the theorem follows by the usual telescoping sum calculation. □

One can also consider multi-key CPA security, where the adversary sees many encryptions under many public keys. In the public-key setting, semantic security implies not only CPA security, but multi-key CPA security — see Exercise 11.10.

11.4 Encryption based on a trapdoor function scheme

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 (see Section 8.10.2). We then present a concrete instantiation of this scheme, based on RSA (see Section 10.3).

Our encryption scheme is called $\mathcal{E}_{\text{TDF}}$, and is built out of several components:

- a trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$,
- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a hash function $H : \mathcal{X} \to \mathcal{K}$.

The message space for $\mathcal{E}_{\text{TDF}}$ is $\mathcal{M}$, and the ciphertext space is $\mathcal{Y} \times \mathcal{C}$. We now describe the key generation, encryption, and decryption algorithms for $\mathcal{E}_{\text{TDF}}$.

- The key generation algorithm for $\mathcal{E}_{\text{TDF}}$ is the key generation algorithm for $\mathcal{T}$.
- For a given public key pk, and a given message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$E(pk, m) := \quad x \xleftarrow{\text{R}} \mathcal{X}, \quad y \leftarrow F(pk, x), \quad k \leftarrow H(x), \quad c \xleftarrow{\text{R}} E_{\text{s}}(k, m)$$
$$\text{output } (y, c).$$

- For a given secret key sk, and a given ciphertext $(y, c) \in \mathcal{Y} \times \mathcal{C}$, the decryption algorithm runs as follows:

$$D(sk,\ (y, c)\ ) := \quad x \leftarrow I(sk, y), \quad k \leftarrow H(x), \quad m \leftarrow D_{\text{s}}(k, c)$$
$$\text{output } m.$$

Thus, $\mathcal{E}_{\text{TDF}} = (G, E, D)$, and is defined over $(\mathcal{M}, \mathcal{Y} \times \mathcal{C})$.

The correctness property for $\mathcal{T}$ immediately implies the correctness property for $\mathcal{E}_{\text{TDF}}$. If H is modeled as a random oracle (see Section 8.10), one can prove that $\mathcal{E}_{\text{TDF}}$ is semantically secure, assuming that $\mathcal{T}$ is one-way, and that $\mathcal{E}_{\text{s}}$ is semantically secure.

Recall that in the random oracle model, the function H is modeled as a random function $\mathcal{O}$ chosen at random from the set of all functions $\text{Funs}[\mathcal{X}, \mathcal{K}]$. More precisely, in the random oracle version of Attack Game 11.1, the challenger chooses $\mathcal{O}$ at random. In any computation where the challenger would normally evaluate H, it evaluates $\mathcal{O}$ instead. In addition, the adversary is allowed to ask the challenger for the value of the function $\mathcal{O}$ at any point of its choosing. The adversary may make any number of such "random oracle queries" at any time of its choosing. We use $\text{SS}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{TDF}}]$ to denote $\mathcal{A}$'s advantage against $\mathcal{E}_{\text{TDF}}$ in the random oracle version of Attack Game 11.1.

Theorem 11.2. *Assume $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle. If $\mathcal{T}$ is one-way and $\mathcal{E}_{\text{s}}$ is semantically secure, then $\mathcal{E}_{\text{TDF}}$ is semantically secure.*

In particular, for every SS adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\text{TDF}}$ as in the random oracle version of Attack Game 11.1, there exist an inverting adversary $\mathcal{B}_{\text{ow}}$ that attacks $\mathcal{T}$ as in Attack Game 10.2,

and an SS adversary $\mathcal{B}_{\mathrm{s}}$ that attacks $\mathcal{E}_{\mathrm{s}}$ as in Attack Game 2.1, where $\mathcal{B}_{\mathrm{ow}}$ and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{SS}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{TDF}}] \leq 2 \cdot \mathrm{OW}\mathsf{adv}[\mathcal{B}_{\mathrm{ow}}, \mathcal{T}] + \mathrm{SS}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{11.2}$$

Proof idea. Suppose the adversary sees the ciphertext (y, c), where $y = F(pk, x)$. If H is modeled as a random oracle, then intuitively, the only way the adversary can learn anything at all about the symmetric key k used to generate c is to explicitly evaluate the random oracle representing H at the point x; however, if he could do this, we could easily convert the adversary into an adversary that inverts the function $F(pk, \cdot)$, contradicting the one-wayness assumption. Therefore, from the adversary's point of view, k is completely random, and semantic security for $\mathcal{E}_{\mathrm{TDF}}$ follows directly from the semantic security of $\mathcal{E}_{\mathrm{s}}$. In the detailed proof, we implement the random oracle using the same "faithful gnome" technique as was used to efficiently implement random functions (see Section 4.4.2); that is, we represent the random oracle as a table of input/output pairs corresponding to points at which the adversary actually queried the random oracle (as well as the point at which the challenger queries the random oracle when it runs the encryption algorithm). We also use many of the same proof techniques introduced in Chapter 4, specifically, the "forgetful gnome" technique (introduced in the proof of Theorem 4.6) and the Difference Lemma (Theorem 4.7). □

Proof. It is convenient to prove the theorem using the bit-guessing versions of the semantic security game. We prove:

$$\mathrm{SS}^{\mathrm{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\mathrm{TDF}}] \leq \mathrm{OW}\mathsf{adv}[\mathcal{B}_{\mathrm{ow}}, \mathcal{T}] + \mathrm{SS}\mathsf{adv}^*[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{11.3}$$

Then (11.2) follows by (11.1) and (2.10).

Define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the bit-guessing version of Attack Game 11.1 with respect to $\mathcal{E}_{\mathrm{TDF}}$. We then modify the challenger to obtain Game 1. In each game, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, 1$, we define W_j to be the event that $\hat{b} = b$ in Game j. We will show that $|\Pr[W_1] - \Pr[W_0]|$ is negligible, and that $\Pr[W_1]$ is negligibly close to $1/2$. From this, it follows that

$$\mathrm{SS}^{\mathrm{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\mathrm{TDF}}] = |\Pr[W_0] - 1/2| \tag{11.4}$$

is also negligible.

Game 0. Note that the challenger in Game 0 also has to respond to the adversary's random oracle queries. The adversary can make any number of random oracle queries, but at most one encryption query. Recall that in addition to direct access the random oracle via explicit random oracle queries, the adversary also has indirect access to the random oracle via the encryption query, where the challenger also makes use of the random oracle. In describing this game, we directly implement the random oracle as a "faithful gnome." This is done using an associative array $Map : \mathcal{X} \to \mathcal{K}$. The details are in Fig. 11.2. In the initialization step, the challenger prepares some quantities that will be used later in processing the encryption query. In particular, in addition to computing $(pk, sk) \xleftarrow{\mathrm{R}} G()$, the challenger precomputes $x \xleftarrow{\mathrm{R}} \mathcal{X}$, $y \leftarrow F(pk, x)$, $k \xleftarrow{\mathrm{R}} \mathcal{K}$. It also sets $Map[x] \leftarrow k$, which means that the value of the random oracle at x is equal to k.

Game 1. This game is precisely the same as Game 0, except that we make our gnome "forgetful" by deleting line (3) in Fig. 11.2.

Let Z be the event that the adversary queries the random oracle at the point x in Game 1. Clearly, Games 0 and 1 proceed identically unless Z occurs, and so by the Difference Lemma, we

initialization:

(1) $(pk, sk) \xleftarrow{R} G()$, $x \xleftarrow{R} \mathcal{X}$, $y \leftarrow F(pk, x)$
initialize an empty associative array $Map : \mathcal{X} \to \mathcal{K}$

(2) $k \xleftarrow{R} \mathcal{K}$, $b \xleftarrow{R} \{0, 1\}$

(3) $Map[x] \leftarrow k$
send the public key pk to $\mathcal{A}$;

upon receiving an encryption query $(m_0, m_1) \in \mathcal{M}^2$:

(4) $c \leftarrow E_s(k, m_b)$
send (y, c) to $\mathcal{A}$;

upon receiving a random oracle query $\hat{x} \in \mathcal{X}$:
if $\hat{x} \notin \text{Domain}(Map)$ then $Map[\hat{x}] \xleftarrow{R} \mathcal{K}$
send $Map[\hat{x}]$ to $\mathcal{A}$

Figure 11.2: Game 0 challenger

have

$$|\Pr[W_1] - \Pr[W_0]| \leq \Pr[Z]. \tag{11.5}$$

If event Z happens, then one of the adversary's random oracle queries is the inverse of y under $F(pk, \cdot)$. Moreover, in Game 1, the value x is used only to define $y = F(pk, x)$, and nowhere else. Thus, we can use adversary $\mathcal{A}$ to build an efficient adversary $\mathcal{B}_{\text{ow}}$ that breaks the one-wayness assumption for $\mathcal{T}$ with an advantage equal to $\Pr[Z]$.

Here is how adversary $\mathcal{B}_{\text{ow}}$ works in detail. This adversary plays Attack Game 10.2 against a challenger $\mathbf{C}_{\text{ow}}$, and plays the role of challenger to $\mathcal{A}$ as in Fig. 11.2, except with the following lines modified as indicated:

(1) obtain (pk, y) from $\mathbf{C}_{\text{ow}}$

(3) *(deleted)*

Additionally,

when $\mathcal{A}$ terminates:
if $F(pk, \hat{x}) = y$ for some $\hat{x} \in \text{Domain}(Map)$
then output $\hat{x}$
else output "failure".

To analyze $\mathcal{B}_{\text{ow}}$, we may naturally view Game 1 and the game played between $\mathcal{B}_{\text{ow}}$ and $\mathbf{C}_{\text{ow}}$ as operating on the same underlying probability space. By definition, Z occurs if and only if $x \in \text{Domain}(Map)$ when $\mathcal{B}_{\text{ow}}$ finishes its game. Therefore,

$$\Pr[Z] = \text{OW}\mathsf{adv}[\mathcal{B}_{\text{ow}}, \mathcal{T}]. \tag{11.6}$$

Observe that in Game 1, the key k is only used to encrypt the challenge plaintext. As such, the adversary is essentially attacking $\mathcal{E}_s$ as in the bit-guessing version of Attack Game 2.1 at this

point. More precisely, we derive an efficient SS adversary $\mathcal{B}_s$ based on Game 1 that uses $\mathcal{A}$ as a subroutine, such that

$$|\Pr[W_1] - 1/2| = \mathsf{SSadv}^*[\mathcal{B}_s, \mathcal{E}_s]. \tag{11.7}$$

Adversary $\mathcal{B}_s$ plays the bit-guessing version of Attack Game 2.1 against a challenger $\mathbf{C}_s$, and plays the role of challenger to $\mathcal{A}$ as in Fig. 11.2, except with the following lines modified as indicated:

(2) *(deleted)*
(3) *(deleted)*
(4) forward (m_0, m_1) to $\mathbf{C}_s$, obtaining c

Additionally,

when $\mathcal{A}$ outputs $\hat{b}$:
    output $\hat{b}$

To analyze $\mathcal{B}_s$, we may naturally view Game 1 and the game played between $\mathcal{B}_s$ and $\mathbf{C}_s$ as operating on the same underlying probability space. By construction, $\mathcal{B}_s$ and $\mathcal{A}$ output the same thing, and so (11.7) holds.

Combining (11.4), (11.5), (11.6), and (11.7), yields (11.3). □

11.4.1 Instantiating $\mathcal{E}_{\text{TDF}}$ with RSA

Suppose we now use RSA (see Section 10.3) to instantiate $\mathcal{T}$ in the above encryption scheme $\mathcal{E}_{\text{TDF}}$. This scheme is parameterized by two quantities: the length ℓ of the prime factors of the RSA modulus, and the encryption exponent e, which is an odd, positive integer. Recall that the RSA scheme does not quite fit the definition of a trapdoor permutation scheme, because the domain of the trapdoor permutation is not a fixed set, but varies with the public key. Let us assume that $\mathcal{X}$ is a fixed set into which we may embed $\mathbb{Z}_n$, for every RSA modulus n generated by $\text{RSAGen}(\ell, e)$ (for example, we could take $\mathcal{X} = \{0,1\}^{2\ell}$). The scheme also makes use of a symmetric cipher $\mathcal{E}_s = (E_s, D_s)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, as well as a hash function $H : \mathcal{X} \to \mathcal{K}$.

The basic RSA encryption scheme is $\mathcal{E}_{\text{RSA}} = (G, E, D)$, with message space $\mathcal{M}$ and ciphertext space $\mathcal{X} \times \mathcal{C}$, where

- the key generation algorithm runs as follows:

$$\begin{aligned} G() := \quad & (n,d) \xleftarrow{R} \text{RSAGen}(\ell, e), \quad pk \leftarrow (n,e), \quad sk \leftarrow (n,d) \\ & \text{output } (pk, sk); \end{aligned}$$

- for a given public key $pk = (n, e)$, and message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & x \xleftarrow{R} \mathbb{Z}_n, \quad y \leftarrow x^e, \quad k \leftarrow H(x), \quad c \xleftarrow{R} E_s(k, m) \\ & \text{output } (y, c) \in \mathcal{X} \times \mathcal{C}; \end{aligned}$$

- for a given secret key $sk = (n, d)$, and a given ciphertext $(y, c) \in \mathcal{X} \times \mathcal{C}$, where y represents an element of $\mathbb{Z}_n$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk,\ (y,c)\ ) := \quad & x \leftarrow y^d, \quad k \leftarrow H(x), \quad m \leftarrow D_s(k, c) \\ & \text{output } m. \end{aligned}$$

Theorem 11.3. *Assume $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle. If the RSA assumption holds for parameters (ℓ, e), and $\mathcal{E}_{\mathrm{s}}$ is semantically secure, then $\mathcal{E}_{\mathrm{RSA}}$ is semantically secure.*

In particular, for any SS adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\mathrm{RSA}}$ as in the random oracle version of Attack Game 11.1, there exist an RSA adversary $\mathcal{B}_{\mathrm{rsa}}$ that breaks the RSA assumption for (ℓ, e) as in Attack Game 10.3, and an SS adversary $\mathcal{B}_{\mathrm{s}}$ that attacks $\mathcal{E}_{\mathrm{s}}$ as in Attack Game 2.1, where $\mathcal{B}_{\mathrm{rsa}}$ and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{SS}^{\mathrm{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\mathrm{RSA}}] \leq \mathrm{RSA}\mathsf{adv}[\mathcal{B}_{\mathrm{rsa}}, \ell, e] + \mathrm{SS}\mathsf{adv}^*[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}].$$

Proof. The proof of Theorem 11.2 carries over, essentially unchanged. □

11.5 ElGamal encryption

In this section we show how to build a public-key encryption scheme from Diffie-Hellman. Security will be based on either the CDH or DDH assumptions from Section 10.5.

The encryption scheme is a variant of a scheme first proposed by ElGamal, and we call it $\mathcal{E}_{\mathrm{EG}}$. It is built out of several components:

- a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$,
- a symmetric cipher $\mathcal{E}_{\mathrm{s}} = (E_{\mathrm{s}}, D_{\mathrm{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a hash function $H : \mathbb{G} \to \mathcal{K}$.

The message space for $\mathcal{E}_{\mathrm{EG}}$ is $\mathcal{M}$, and the ciphertext space is $\mathbb{G} \times \mathcal{C}$. We now describe the key generation, encryption, and decryption algorithms for $\mathcal{E}_{\mathrm{EG}}$.

- the key generation algorithm runs as follows:

$$\begin{aligned} G() := \quad & \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad u \leftarrow g^\alpha \\ & pk \leftarrow u, \quad sk \leftarrow \alpha \\ & \text{output } (pk, sk); \end{aligned}$$

- for a given public key $pk = u \in \mathbb{G}$ and message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad v \leftarrow g^\beta, \quad w \leftarrow u^\beta, \quad k \leftarrow H(w), \quad c \leftarrow E_{\mathrm{s}}(k, m) \\ & \text{output } (v, c); \end{aligned}$$

- for a given secret key $sk = \alpha \in \mathbb{Z}_q$ and a ciphertext $(v, c) \in \mathbb{G} \times \mathcal{C}$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk,\ (v, c)\ ) := \quad & w \leftarrow v^\alpha, \quad k \leftarrow H(w), \quad m \leftarrow D_{\mathrm{s}}(k, c) \\ & \text{output } m. \end{aligned}$$

Thus, $\mathcal{E}_{\mathrm{EG}} = (G, E, D)$, and is defined over $(\mathcal{M}, \mathbb{G} \times \mathcal{C})$.

Note that the description of the group $\mathbb{G}$ and generator $g \in \mathbb{G}$ is considered to be a system parameter, rather than part of the public key.