Chapter 2

Perfectly Secret Encryption

In the previous chapter we presented historical encryption schemes and showed how they can be broken with little computational effort. In this chapter, we look at the other extreme and study encryption schemes that are *provably* secure even against an adversary with unbounded computational power. Such schemes are called *perfectly secret.* Besides rigorously defining the notion, we will explore conditions under which perfect secrecy can be achieved. (Beginning in this chapter, we assume familiarity with basic probability theory. The relevant notions are reviewed in Appendix A.3.)

The material in this chapter belongs, in some sense, more to the world of "classical" cryptography than to the world of "modern" cryptography. Besides the fact that all the material introduced here was developed before the revolution in cryptography that took place in the mid-1970s and 1980s, the constructions we study in this chapter rely only on the first and third principles outlined in Section 1.4. That is, precise mathematical definitions are used and rigorous proofs are given, but it will not be necessary to rely on any unproven computational assumptions. It is clearly advantageous to avoid such assumptions; we will see, however, that doing so has inherent limitations. Thus, in addition to serving as a good basis for understanding the principles underlying modern cryptography, the results of this chapter also justify our later adoption of all three of the aforementioned principles.

Beginning with this chapter, we will define security and analyze schemes using probabilistic experiments involving algorithms making randomized choices; a basic example is given by communicating parties' choosing a random key. Thus, before returning to the subject of cryptography *per se*, we briefly discuss the issue of generating randomness suitable for cryptographic applications.

Generating randomness. Throughout the book, we will simply assume that parties have access to an unlimited supply of independent, unbiased random bits. In practice, where do these random bits come from? In principle, one could generate a small number of random bits by hand, e.g., by flipping a fair coin. But such an approach is not very convenient, nor does it scale.

Modern *random-number generation* proceeds in two steps. First, a "pool" of high-entropy data is collected. (For our purposes a formal definition of entropy is not needed, and it suffices to think of entropy as a measure of unpredictability.) Next, this high-entropy data is processed to yield a sequence of nearly independent and unbiased bits. This second step is necessary since high-entropy data is not necessarily uniform.

For the first step, some source of unpredictable data is needed. There are several ways such data can be acquired. One technique is to rely on external inputs, for example, delays between network events, hard-disk access times, keystrokes or mouse movements made by the user, and so on. Such data is likely to be far from uniform, but if enough measurements are taken the resulting pool of data is expected to have sufficient entropy. More sophisticated approaches—which, by design, incorporate random-number generation more tightly into the system at the hardware level—have also been used. These rely on physical phenomena such as thermal/shot noise or radioactive decay. Intel has recently developed a processor that includes a digital random-number generator on the processor chip and provides a dedicated instruction for accessing the resulting random bits (after they have been suitably processed to yield independent, unbiased bits, as discussed next).

The processing needed to "smooth" the high-entropy data to obtain (nearly) uniform bits is a non-trivial one, and is discussed briefly in Section 5.6.4. Here, we just give a simple example to give an idea of what is done. Imagine that our high-entropy pool results from a sequence of *biased* coin flips, where "heads" occurs with probability p and "tails" with probability $1-p$. (We do assume, however, that the result of any coin flip is *independent* of all other coin flips. In practice this assumption is typically not valid.) The result of 1,000 such coin flips certainly has high entropy, but is not close to uniform. We can obtain a uniform distribution by considering the coin flips in pairs: if we see a head followed by a tail then we output "0," and if we see a tail followed by a head then we output "1." (If we see two heads or two tails in a row, we output nothing, and simply move on to the next pair.) The probability that any pair results in a "0" is $p \cdot (1-p)$, which is exactly equal to the probability that any pair results in a "1," and we thus obtain a uniformly distributed output from our initial high-entropy pool.

Care must be taken in how random bits are produced, and using poor random-number generators can often leave a good cryptosystem vulnerable to attack. One should use a random-number generator that is *designed for cryptographic use*, rather than a "general-purpose" random-number generator, which is not suitable for cryptographic applications. In particular, the `rand()` function in the C stdlib.h library is *not* cryptographically secure, and using it in cryptographic settings can have disastrous consequences.

2.1 Definitions

We begin by recalling and expanding upon the syntax that was introduced in the previous chapter. An encryption scheme is defined by three algorithms Gen, Enc, and Dec, as well as a specification of a (finite) *message space* $\mathcal{M}$

with $|\mathcal{M}| > 1$.[1] The key-generation algorithm Gen is a probabilistic algorithm that outputs a key k chosen according to some distribution. We denote by $\mathcal{K}$ the (finite) *key space*, i.e., the set of all possible keys that can be output by Gen. The encryption algorithm Enc takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext c. We now allow the encryption algorithm to be probabilistic (so $\mathsf{Enc}_k(m)$ might output a different ciphertext when run multiple times), and we write $c \leftarrow \mathsf{Enc}_k(m)$ to denote the possibly probabilistic process by which message m is encrypted using key k to give ciphertext c. (In case Enc is deterministic, we may emphasize this by writing $c := \mathsf{Enc}_k(m)$. Looking ahead, we also sometimes use the notation $x \leftarrow S$ to denote uniform selection of x from a set S.) We let $\mathcal{C}$ denote the set of all possible ciphertexts that can be output by $\mathsf{Enc}_k(m)$, for all possible choices of $k \in \mathcal{K}$ and $m \in \mathcal{M}$ (and for all random choices of Enc in case it is randomized). The decryption algorithm Dec takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$. We assume *perfect correctness*, meaning that for all $k \in \mathcal{K}$, $m \in \mathcal{M}$, and any ciphertext c output by $\mathsf{Enc}_k(m)$, it holds that $\mathsf{Dec}_k(c) = m$ with probability 1. Perfect correctness implies that we may assume Dec is deterministic without loss of generality, since $\mathsf{Dec}_k(c)$ must give the same output every time it is run. We will thus write $m := \mathsf{Dec}_k(c)$ to denote the process of decrypting ciphertext c using key k to yield the message m.

In the definitions and theorems below, we refer to probability distributions over $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$. The distribution over $\mathcal{K}$ is the one defined by running Gen and taking the output. (It is almost always the case that Gen chooses a key uniformly from $\mathcal{K}$ and, in fact, we may assume this without loss of generality; see Exercise 2.1.) We let K be a random variable denoting the value of the key output by Gen; thus, for any $k \in \mathcal{K}$, $\Pr[K = k]$ denotes the probability that the key output by Gen is equal to k. Similarly, we let M be a random variable denoting the message being encrypted, so $\Pr[M = m]$ denotes the probability that the message takes on the value $m \in \mathcal{M}$. The probability distribution of the message is not determined by the encryption scheme itself, but instead reflects the likelihood of different messages being sent by the parties using the scheme, as well as an adversary's uncertainty about what will be sent. As an example, an adversary may know that the message will either be `attack today` or `don't attack`. The adversary may even know (by other means) that with probability 0.7 the message will be a command to attack and with probability 0.3 the message will be a command not to attack. In this case, we have $\Pr[M = \texttt{attack today}] = 0.7$ and $\Pr[M = \texttt{don't attack}] = 0.3$.

K and M are assumed to be independent, i.e., what is being communicated by the parties is independent of the key they happen to share. This makes sense, among other reasons, because the distribution over $\mathcal{K}$ is determined by

[1]If $|\mathcal{M}| = 1$ there is only one message and no point in communicating, let alone encrypting.

the encryption scheme itself (since it is defined by Gen), while the distribution over $\mathcal{M}$ depends on the context in which the encryption scheme is being used.

Fixing an encryption scheme and a distribution over $\mathcal{M}$ determines a distribution over the space of ciphertexts $\mathcal{C}$ given by choosing a key $k \in \mathcal{K}$ (according to Gen) and a message $m \in \mathcal{M}$ (according to the given distribution), and then computing the ciphertext $c \leftarrow \mathsf{Enc}_k(m)$. We let C be the random variable denoting the resulting ciphertext and so, for $c \in \mathcal{C}$, write $\Pr[C = c]$ to denote the probability that the ciphertext is equal to the fixed value c.

Example 2.1

We work through a simple example for the shift cipher (cf. Section 1.3). Here, by definition, we have $\mathcal{K} = \{0, \ldots, 25\}$ with $\Pr[K = k] = 1/26$ for each $k \in \mathcal{K}$.

Say we are given the following distribution over $\mathcal{M}$:

$$\Pr[M = \mathtt{a}] = 0.7 \quad \text{and} \quad \Pr[M = \mathtt{z}] = 0.3.$$

What is the probability that the ciphertext is $\mathtt{B}$? There are only two ways this can occur: either $M = \mathtt{a}$ and $K = 1$, or $M = \mathtt{z}$ and $K = 2$. By independence of M and K, we have

$$\begin{aligned}\Pr[M = \mathtt{a} \wedge K = 1] &= \Pr[M = \mathtt{a}] \cdot \Pr[K = 1] \\ &= 0.7 \cdot \left(\frac{1}{26}\right).\end{aligned}$$

Similarly, $\Pr[M = \mathtt{z} \wedge K = 2] = 0.3 \cdot \left(\frac{1}{26}\right)$. Therefore,

$$\begin{aligned}\Pr[C = \mathtt{B}] &= \Pr[M = \mathtt{a} \wedge K = 1] + \Pr[M = \mathtt{z} \wedge K = 2] \\ &= 0.7 \cdot \left(\frac{1}{26}\right) + 0.3 \cdot \left(\frac{1}{26}\right) \ = \ 1/26.\end{aligned}$$

We can calculate conditional probabilities as well. For example, what is the probability that the message $\mathtt{a}$ was encrypted, given that we observe ciphertext $\mathtt{B}$? Using Bayes' Theorem (Theorem A.8) we have

$$\begin{aligned}\Pr[M = \mathtt{a} \mid C = \mathtt{B}] &= \frac{\Pr[C = \mathtt{B} \mid M = \mathtt{a}] \cdot \Pr[M = \mathtt{a}]}{\Pr[C = \mathtt{B}]} \\ &= \frac{0.7 \cdot \Pr[C = \mathtt{B} \mid M = \mathtt{a}]}{1/26}.\end{aligned}$$

Note that $\Pr[C = \mathtt{B} \mid M = \mathtt{a}] = 1/26$, since if $M = \mathtt{a}$ then the only way $C = \mathtt{B}$ can occur is if $K = 1$ (which occurs with probability 1/26). We conclude that $\Pr[M = \mathtt{a} \mid C = \mathtt{B}] = 0.7$. ◊

Example 2.2

Consider the shift cipher again, but with the following distribution over $\mathcal{M}$:

$$\Pr[M = \mathtt{kim}] = 0.5, \ \Pr[M = \mathtt{ann}] = 0.2, \ \Pr[M = \mathtt{boo}] = 0.3.$$

What is the probability that $C = \texttt{DQQ}$? The only way this ciphertext can occur is if $M = \texttt{ann}$ and $K = 3$, or $M = \texttt{boo}$ and $K = 2$, which happens with probability $0.2 \cdot 1/26 + 0.3 \cdot 1/26 = 1/52$.

So what is the probability that ann was encrypted, conditioned on observing the ciphertext DQQ? A calculation as above using Bayes' Theorem gives $\Pr[M = \texttt{ann} \mid C = \texttt{DQQ}] = 0.4$. $\diamondsuit$

Perfect secrecy. We are now ready to define the notion of *perfect secrecy*. We imagine an adversary who knows the probability distribution over $\mathcal{M}$; that is, the adversary knows the likelihood that different messages will be sent. This adversary also knows the encryption scheme being used; the only thing unknown to the adversary is the key shared by the parties. A message is chosen by one of the honest parties and encrypted, and the resulting ciphertext transmitted to the other party. The adversary can *eavesdrop* on the parties' communication, and thus observe this ciphertext. (That is, this is a ciphertext-only attack, where the attacker gets only a single ciphertext.) For a scheme to be perfectly secret, observing this ciphertext should have *no effect* on the adversary's knowledge regarding the actual message that was sent; in other words, the *a posteriori* probability that some message $m \in \mathcal{M}$ was sent, conditioned on the ciphertext that was observed, should be no different from the *a priori* probability that m would be sent. This means that the ciphertext reveals nothing about the underlying plaintext, and the adversary learns absolutely nothing about the plaintext that was encrypted. Formally:

DEFINITION 2.3 *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *is* perfectly secret *if for every probability distribution over* $\mathcal{M}$*, every message* $m \in \mathcal{M}$*, and every ciphertext* $c \in \mathcal{C}$ *for which* $\Pr[C = c] > 0$*:*

$$\Pr[M = m \mid C = c] = \Pr[M = m].$$

(The requirement that $\Pr[C = c] > 0$ is a technical one needed to prevent conditioning on a zero-probability event.)

We now give an equivalent formulation of perfect secrecy. Informally, this formulation requires that the probability distribution of the ciphertext does not depend on the plaintext, i.e., for any two messages $m, m' \in \mathcal{M}$ the distribution of the ciphertext when m is encrypted should be identical to the distribution of the ciphertext when m' is encrypted. Formally, for every $m, m' \in \mathcal{M}$, and every $c \in \mathcal{C}$,

$$\Pr[\mathsf{Enc}_K(m) = c] = \Pr[\mathsf{Enc}_K(m') = c] \tag{2.1}$$

(where the probabilities are over choice of K and any randomness of Enc). This implies that the ciphertext contains no information about the plaintext, and that it is impossible to distinguish an encryption of m from an encryption of m', since the distributions over the ciphertext are the same in each case.

LEMMA 2.4 *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *is perfectly secret if and only if Equation (2.1) holds for every* $m, m' \in \mathcal{M}$ *and every* $c \in \mathcal{C}$.

PROOF We show that if the stated condition holds, then the scheme is perfectly secret; the converse implication is left to Exercise 2.4. Fix a distribution over $\mathcal{M}$, a message m, and a ciphertext c for which $\Pr[C = c] > 0$. If $\Pr[M = m] = 0$ then we trivially have

$$\Pr[M = m \mid C = c] = 0 = \Pr[M = m].$$

So, assume $\Pr[M = m] > 0$. Notice first that

$$\Pr[C = c \mid M = m] = \Pr[\mathsf{Enc}_K(M) = c \mid M = m] = \Pr[\mathsf{Enc}_K(m) = c],$$

where the first equality is by definition of the random variable C, and the second is because we condition on the event that M is equal to m. Set $\delta_c \stackrel{\text{def}}{=} \Pr[\mathsf{Enc}_K(m) = c] = \Pr[C = c \mid M = m]$. If the condition of the lemma holds, then for *every* $m' \in \mathcal{M}$ we have $\Pr[\mathsf{Enc}_K(m') = c] = \Pr[C = c \mid M = m'] = \delta_c$. Using Bayes' Theorem (see Appendix A.3), we thus have

$$\begin{aligned}
\Pr[M = m \mid C = c] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\
&= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\sum_{m' \in \mathcal{M}} \Pr[C = c \mid M = m'] \cdot \Pr[M = m']} \\
&= \frac{\delta_c \cdot \Pr[M = m]}{\sum_{m' \in \mathcal{M}} \delta_c \cdot \Pr[M = m']} \\
&= \frac{\Pr[M = m]}{\sum_{m' \in \mathcal{M}} \Pr[M = m']} \quad = \quad \Pr[M = m],
\end{aligned}$$

where the summation is over $m' \in \mathcal{M}$ with $\Pr[M = m'] \neq 0$. We conclude that for every $m \in \mathcal{M}$ and $c \in \mathcal{C}$ for which $\Pr[C = c] > 0$, it holds that $\Pr[M = m \mid C = c] = \Pr[M = m]$, and so the scheme is perfectly secret. ∎

Perfect (adversarial) indistinguishability. We conclude this section by presenting another equivalent definition of perfect secrecy. This definition is based on an *experiment* involving an adversary passively observing a ciphertext and then trying to guess which of two possible messages was encrypted. We introduce this notion since it will serve as our starting point for defining computational security in the next chapter. Indeed, throughout the rest of the book we will often use experiments of this sort to define security.

In the present context, we consider the following experiment: an adversary $\mathcal{A}$ first specifies two arbitrary messages $m_0, m_1 \in \mathcal{M}$. One of these two messages is chosen uniformly at random and encrypted using a random key; the resulting ciphertext is given to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a "guess" as to which of the two messages was encrypted; $\mathcal{A}$ *succeeds* if it guesses correctly. An encryption scheme is *perfectly indistinguishable* if *no* adversary $\mathcal{A}$ can succeed with probability better than 1/2. (Note that, for any encryption scheme, $\mathcal{A}$ can succeed with probability 1/2 by outputting a uniform guess; the requirement is simply that no attacker can do any better than this.) We stress that no limitations are placed on the computational power of $\mathcal{A}$.

Formally, let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme with message space $\mathcal{M}$. Let $\mathcal{A}$ be an adversary, which is formally just a (stateful) algorithm. We define an experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ as follows:

The adversarial indistinguishability experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$:

1. *The adversary $\mathcal{A}$ outputs a pair of messages $m_0, m_1 \in \mathcal{M}$.*
2. *A key k is generated using* Gen*, and a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$. We refer to c as the* challenge ciphertext*.*
3. *$\mathcal{A}$ outputs a bit b'.*
4. *The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. We write $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1$ if the output of the experiment is 1 and in this case we say that $\mathcal{A}$* succeeds*.*

As noted earlier, it is trivial for $\mathcal{A}$ to succeed with probability 1/2 by outputting a random guess. Perfect indistinguishability requires that it is impossible for any $\mathcal{A}$ to do better.

DEFINITION 2.5 *Encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ is* perfectly indistinguishable *if for every $\mathcal{A}$ it holds that*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] = \frac{1}{2}\,.$$

The following lemma states that Definition 2.5 is equivalent to Definition 2.3. We leave the proof of the lemma as Exercise 2.5.

LEMMA 2.6 *Encryption scheme Π is perfectly secret if and only if it is perfectly indistinguishable.*

Example 2.7

We show that the Vigenère cipher is *not* perfectly indistinguishable, at least for certain parameters. Concretely, let Π denote the Vigenère cipher for the message space of two-character strings, and where the period is chosen uniformly in $\{1,2\}$. To show that Π is not perfectly indistinguishable, we exhibit an adversary $\mathcal{A}$ for which $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] > \frac{1}{2}$.

Adversary $\mathcal{A}$ does:

1. Output $m_0 = \mathtt{aa}$ and $m_1 = \mathtt{ab}$.
2. Upon receiving the challenge ciphertext $c = c_1 c_2$, do the following: if $c_1 = c_2$ output 0; else output 1.

Computation of $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right]$ is tedious but straightforward.

$$\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] \\
&= \frac{1}{2} \cdot \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1 \mid b = 0\right] + \frac{1}{2} \cdot \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1 \mid b = 1\right] \\
&= \frac{1}{2} \cdot \Pr[\mathcal{A} \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A} \text{ outputs } 1 \mid b = 1], \qquad (2.2)
\end{aligned}$$

where b is the uniform bit determining which message gets encrypted. $\mathcal{A}$ outputs 0 if and only if the two characters of the ciphertext $c = c_1 c_2$ are equal. When $b = 0$ (so $m_0 = \mathtt{aa}$ is encrypted) then $c_1 = c_2$ if either (1) a key of period 1 is chosen, or (2) a key of period 2 is chosen, and both characters of the key are equal. The former occurs with probability $\frac{1}{2}$, and the latter occurs with probability $\frac{1}{2} \cdot \frac{1}{26}$. So

$$\Pr[\mathcal{A} \text{ outputs } 0 \mid b = 0] = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{26} \approx 0.52.$$

When $b = 1$ then $c_1 = c_2$ only if a key of period 2 is chosen and the first character of the key is one more than the second character of the key, which happens with probability $\frac{1}{2} \cdot \frac{1}{26}$. So

$$\Pr[\mathcal{A} \text{ outputs } 1 \mid b = 1] = 1 - \Pr[\mathcal{A} \text{ outputs } 0 \mid b = 1] = 1 - \frac{1}{2} \cdot \frac{1}{26} \approx 0.98.$$

Plugging into Equation (2.2) then gives

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] = \frac{1}{2} \cdot \left(\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{26} + 1 - \frac{1}{2} \cdot \frac{1}{26}\right) = 0.75 > \frac{1}{2},$$

and the scheme is not perfectly indistinguishable. ♢

2.2 The One-Time Pad

In 1917, Vernam patented a perfectly secret encryption scheme now called the *one-time pad*. At the time Vernam proposed the scheme, there was no proof that it was perfectly secret; in fact, there was not yet a notion of what perfect secrecy was. Approximately 25 years later, however, Shannon introduced the definition of perfect secrecy and demonstrated that the one-time pad achieves that level of security.

CONSTRUCTION 2.8

Fix an integer $\ell > 0$. The message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{C}$ are all equal to $\{0,1\}^\ell$ (the set of all binary strings of length ℓ).

- Gen: the key-generation algorithm chooses a key from $\mathcal{K} = \{0,1\}^\ell$ according to the uniform distribution (i.e., each of the 2^ℓ strings in the space is chosen as the key with probability exactly $2^{-\ell}$).
- Enc: given a key $k \in \{0,1\}^\ell$ and a message $m \in \{0,1\}^\ell$, the encryption algorithm outputs the ciphertext $c := k \oplus m$.
- Dec: given a key $k \in \{0,1\}^\ell$ and a ciphertext $c \in \{0,1\}^\ell$, the decryption algorithm outputs the message $m := k \oplus c$.

The one-time pad encryption scheme.

In describing the scheme we let $a \oplus b$ denote the *bitwise exclusive-or* (XOR) of two binary strings a and b (i.e., if $a = a_1 \cdots a_\ell$ and $b = b_1 \cdots b_\ell$ are ℓ-bit strings, then $a \oplus b$ is the ℓ-bit string given by $a_1 \oplus b_1 \cdots a_\ell \oplus b_\ell$). In the one-time pad encryption scheme the key is a uniform string of the same length as the message; the ciphertext is computed by simply XORing the key and the message. A formal definition is given as Construction 2.8. Before discussing security, we first verify correctness: for every key k and every message m it holds that $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = k \oplus k \oplus m = m$, and so the one-time pad constitutes a valid encryption scheme.

One can easily prove perfect secrecy of the one-time pad using Lemma 2.4 and the fact that the ciphertext is uniformly distributed regardless of what message is encrypted. We give a proof based directly on the original definition.

THEOREM 2.9 *The one-time pad encryption scheme is perfectly secret.*

PROOF We first compute $\Pr[C = c \mid M = m']$ for arbitrary $c \in \mathcal{C}$ and $m' \in \mathcal{M}$. For the one-time pad,

$$\begin{aligned}\Pr[C = c \mid M = m'] = \Pr[\mathsf{Enc}_K(m') = c] &= \Pr[m' \oplus K = c] \\ &= \Pr[K = m' \oplus c] \\ &= 2^{-\ell},\end{aligned}$$

where the final equality holds because the key K is a uniform ℓ-bit string. Fix any distribution over $\mathcal{M}$. For any $c \in \mathcal{C}$, we have

$$\begin{aligned}\Pr[C = c] &= \sum_{m' \in \mathcal{M}} \Pr[C = c \mid M = m'] \cdot \Pr[M = m'] \\ &= 2^{-\ell} \cdot \sum_{m' \in \mathcal{M}} \Pr[M = m'] \\ &= 2^{-\ell},\end{aligned}$$

where the sum is over $m' \in \mathcal{M}$ with $\Pr[M = m'] \neq 0$. Bayes' Theorem gives:

$$\begin{aligned}\Pr[M = m \mid C = c] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{2^{-\ell} \cdot \Pr[M = m]}{2^{-\ell}} \\ &= \Pr[M = m].\end{aligned}$$

We conclude that the one-time pad is perfectly secret. ■

The one-time pad was used by several national-intelligence agencies in the mid-20th century to encrypt sensitive traffic. Perhaps most famously, the "red phone" linking the White House and the Kremlin during the Cold War was protected using one-time pad encryption, where the governments of the US and USSR would exchange extremely long keys using trusted couriers carrying briefcases of paper on which random characters were written.

Notwithstanding the above, one-time pad encryption is rarely used any more due to a number of drawbacks it has. Most prominent is that *the key is as long as the message.*[2] This limits the usefulness of the scheme for sending very long messages (as it may be difficult to securely share and store a very long key), and is problematic when the parties cannot predict in advance (an upper bound on) how long the message will be.

Moreover, the one-time pad—as the name indicates—*is only secure if used once* (with the same key). Although we did not yet define a notion of secrecy when multiple messages are encrypted, it is easy to see that encrypting more than one message with the same key leaks a lot of information. In particular, say two messages m, m' are encrypted using the same key k. An adversary who obtains $c = m \oplus k$ and $c' = m' \oplus k$ can compute

$$c \oplus c' = (m \oplus k) \oplus (m' \oplus k) = m \oplus m'$$

and thus learn the exclusive-or of the two messages or, equivalently, exactly where the two messages differ. While this may not seem very significant, it is enough to rule out any claims of perfect secrecy for encrypting two messages using the same key. Moreover, if the messages correspond to natural-language text, then given the exclusive-or of two sufficiently long messages it is possible to perform frequency analysis (as in the previous chapter, though more complex) and recover the messages themselves. An interesting historical example of this is given by the *VENONA project*, as part of which the US and UK were able to decrypt ciphertexts sent by the Soviet Union that were mistakenly encrypted with repeated portions of a one-time pad over several decades.

[2]This does not make the one-time pad useless, since it may be easier for two parties to share a key at some point in time before the message to be communicated is known.

2.3 Limitations of Perfect Secrecy

We ended the previous section by noting some drawbacks of the one-time pad encryption scheme. Here, we show that these drawbacks are not specific to that scheme, but are instead *inherent* limitations of perfect secrecy. Specifically, we prove that *any* perfectly secret encryption scheme must have a key space that is at least as large as the message space. If all keys are the same length, and the message space consists of all strings of some fixed length, this implies that the key is at least as long as the message. In particular, the key length of the one-time pad is optimal. (The other limitation—namely, that the key can be used only once—is also inherent if perfect secrecy is required; see Exercise 2.13.)

THEOREM 2.10 *If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a perfectly secret encryption scheme with message space* $\mathcal{M}$ *and key space* $\mathcal{K}$*, then* $|\mathcal{K}| \geq |\mathcal{M}|$*.*

PROOF We show that if $|\mathcal{K}| < |\mathcal{M}|$ then the scheme cannot be perfectly secret. Assume $|\mathcal{K}| < |\mathcal{M}|$. Consider the uniform distribution over $\mathcal{M}$ and let $c \in \mathcal{C}$ be a ciphertext that occurs with non-zero probability. Let $\mathcal{M}(c)$ be the set of all possible messages that are possible decryptions of c; that is

$$\mathcal{M}(c) \overset{\text{def}}{=} \{m \mid m = \mathsf{Dec}_k(c) \text{ for some } k \in \mathcal{K}\}.$$

Clearly $|\mathcal{M}(c)| \leq |\mathcal{K}|$. (Recall that we may assume Dec is deterministic.) If $|\mathcal{K}| < |\mathcal{M}|$, there is some $m' \in \mathcal{M}$ such that $m' \notin \mathcal{M}(c)$. But then

$$\Pr[M = m' \mid C = c] = 0 \neq \Pr[M = m'],$$

and so the scheme is not perfectly secret. ■

Perfect secrecy with shorter keys? The above theorem shows an inherent limitation of schemes that achieve perfect secrecy. Even so, individuals occasionally claim they have developed a radically new encryption scheme that is "unbreakable" and achieves the security of the one-time pad without using keys as long as what is being encrypted. The above proof demonstrates that such claims *cannot* be true; anyone making such claims either knows very little about cryptography or is blatantly lying.

2.4 *Shannon's Theorem

In his work on perfect secrecy, Shannon also provided a characterization of perfectly secret encryption schemes. This characterization says that, under certain conditions, the key-generation algorithm Gen must choose the key *uniformly* from the set of all possible keys (as in the one-time pad); moreover, for every message m and ciphertext c there is a *unique* key mapping m to c (again, as in the one-time pad). Beyond being interesting in its own right, this theorem is a useful tool for proving (or disproving) perfect secrecy of suggested schemes. We discuss this further after the proof.

The theorem as stated here assumes $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$, meaning that the sets of plaintexts, keys, and ciphertexts all have the same size. We have already seen that for perfect secrecy we must have $|\mathcal{K}| \geq |\mathcal{M}|$. It is easy to see that correct decryption requires $|\mathcal{C}| \geq |\mathcal{M}|$. Therefore, in some sense, encryption schemes with $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$ are "optimal."

THEOREM 2.11 (Shannon's theorem) *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be an encryption scheme with message space* $\mathcal{M}$*, for which* $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$*. The scheme is perfectly secret if and only if:*

1. *Every key* $k \in \mathcal{K}$ *is chosen with (equal) probability* $1/|\mathcal{K}|$ *by algorithm* Gen.
2. *For every* $m \in \mathcal{M}$ *and every* $c \in \mathcal{C}$*, there exists a unique key* $k \in \mathcal{K}$ *such that* $\mathsf{Enc}_k(m)$ *outputs* c.

PROOF The intuition behind the proof is as follows. To see that the stated conditions imply perfect secrecy, note that condition 2 means that any ciphertext c could be the result of encrypting any possible plaintext m, because there is some key k mapping m to c. Since there is a *unique* such key, and each key is chosen with equal probability, perfect secrecy follows as for the one-time pad. For the other direction, perfect secrecy immediately implies that for every m and c there is at least one key mapping m to c. The fact that $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$ means, moreover, that for every m and c there is exactly *one* such key. Given this, each key must be chosen with equal probability or else perfect secrecy would fail to hold. A formal proof follows.

We assume for simplicity that Enc is deterministic. (One can show that this is without loss of generality here.) We first prove that if the encryption scheme satisfies conditions 1 and 2, then it is perfectly secret. The proof is essentially the same as the proof of perfect secrecy for the one-time pad, so we will be relatively brief. Fix arbitrary $c \in \mathcal{C}$ and $m \in \mathcal{M}$. Let k be the unique key, guaranteed by condition 2, for which $\mathsf{Enc}_k(m) = c$. Then,

$$\Pr[C = c \mid M = m] = \Pr[K = k] = 1/|\mathcal{K}|\,,$$

where the final equality holds by condition 1. So

$$\Pr[C = c] = \sum_{m \in \mathcal{M}} \Pr[\mathsf{Enc}_K(m) = c] \cdot \Pr[M = m] = 1/|\mathcal{K}|.$$

This holds for any distribution over $\mathcal{M}$. Thus, for any distribution over $\mathcal{M}$, any $m \in \mathcal{M}$ with $\Pr[M = m] \neq 0$, and any $c \in \mathcal{C}$, we have:

$$\begin{aligned} \Pr[M = m \mid C = c] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{\Pr[\mathsf{Enc}_K(m) = c] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{|\mathcal{K}|^{-1} \cdot \Pr[M = m]}{|\mathcal{K}|^{-1}} = \Pr[M = m], \end{aligned}$$

and the scheme is perfectly secret.

For the second direction, assume the encryption scheme is perfectly secret; we show that conditions 1 and 2 hold. Fix arbitrary $c \in \mathcal{C}$. There must be some message m^* for which $\Pr[\mathsf{Enc}_K(m^*) = c] \neq 0$. Lemma 2.4 then implies that $\Pr[\mathsf{Enc}_K(m) = c] \neq 0$ for every $m \in \mathcal{M}$. In other words, if we let $\mathcal{M} = \{m_1, m_2, \ldots\}$, then for each $m_i \in \mathcal{M}$ we have a nonempty set of keys $\mathcal{K}_i \subset \mathcal{K}$ such that $\mathsf{Enc}_k(m_i) = c$ if and only if $k \in \mathcal{K}_i$. Moreover, when $i \neq j$ then $\mathcal{K}_i$ and $\mathcal{K}_j$ must be disjoint or else correctness fails to hold. Since $|\mathcal{K}| = |\mathcal{M}|$, we see that each $\mathcal{K}_i$ contains only a single key k_i, as required by condition 2. Now, Lemma 2.4 shows that for any $m_i, m_j \in \mathcal{M}$ we have

$$\Pr[K = k_i] = \Pr[\mathsf{Enc}_K(m_i) = c] = \Pr[\mathsf{Enc}_K(m_j) = c] = \Pr[K = k_j].$$

Since this holds for all $1 \leq i, j \leq |\mathcal{M}| = |\mathcal{K}|$, and $k_i \neq k_j$ for $i \neq j$, this means each key is chosen with probability $1/|\mathcal{K}|$, as required by condition 1. ■

Shannon's theorem is useful for deciding whether a given scheme is perfectly secret. Condition 1 is easy to check, and condition 2 can be demonstrated (or contradicted) without having to compute any probabilities (in contrast to working with Definition 2.3 directly). As an example, perfect secrecy of the one-time pad is trivial to prove using Shannon's theorem. We stress, however, that the theorem only applies when $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$.

References and Additional Reading

The one-time pad is popularly credited to Vernam [172], who filed a patent on it, but recent historical research [25] shows that it was invented some

35 years earlier. Analysis of the one-time pad had to await the groundbreaking work of Shannon [154], who introduced the notion of perfect secrecy.

In this chapter we studied perfectly secret *encryption*. Some other cryptographic problems can also be solved with "perfect" security. A notable example is the problem of message authentication where the aim is to prevent an adversary from (undetectably) modifying a message sent from one party to another. We study this problem in depth in Chapter 4, discussing "perfectly secure" message authentication in Section 4.6.

Exercises

2.1 Prove that, by redefining the key space, we may assume that the key-generation algorithm Gen chooses a key uniformly at random from the key space, without changing $\Pr[C = c \mid M = m]$ for any m, c.

Hint: Define the key space to be the set of all possible random tapes for the randomized algorithm Gen.

2.2 Prove that, by redefining the key space, we may assume that Enc is deterministic without changing $\Pr[C = c \mid M = m]$ for any m, c.

2.3 Prove or refute: An encryption scheme with message space $\mathcal{M}$ is perfectly secret if and only if for every probability distribution over $\mathcal{M}$ and every $c_0, c_1 \in \mathcal{C}$ we have $\Pr[C = c_0] = \Pr[C = c_1]$.

2.4 Prove the second direction of Lemma 2.4.

2.5 Prove Lemma 2.6.

2.6 For each of the following encryption schemes, state whether the scheme is perfectly secret. Justify your answer in each case.

(a) The message space is $\mathcal{M} = \{0, \ldots, 4\}$. Algorithm Gen chooses a uniform key from the key space $\{0, \ldots, 5\}$. $\mathsf{Enc}_k(m)$ returns $[k + m \bmod 5]$, and $\mathsf{Dec}_k(c)$ returns $[c - k \bmod 5]$.

(b) The message space is $\mathcal{M} = \{m \in \{0,1\}^\ell \mid \text{the last bit of } m \text{ is } 0\}$. Gen chooses a uniform key from $\{0,1\}^{\ell-1}$. $\mathsf{Enc}_k(m)$ returns ciphertext $m \oplus (k\|0)$, and $\mathsf{Dec}_k(c)$ returns $c \oplus (k\|0)$.

2.7 When using the one-time pad with the key $k = 0^\ell$, we have $\mathsf{Enc}_k(m) = k \oplus m = m$ and the message is sent in the clear! It has therefore been suggested to modify the one-time pad by only encrypting with $k \neq 0^\ell$ (i.e., to have Gen choose k uniformly from the set of *nonzero* keys of length ℓ). Is this modified scheme still perfectly secret? Explain.

2.8 Let Π denote the Vigenère cipher where the message space consists of all 3-character strings (over the English alphabet), and the key is generated by first choosing the period t uniformly from $\{1, 2, 3\}$ and then letting the key be a uniform string of length t.

(a) Define $\mathcal{A}$ as follows: $\mathcal{A}$ outputs $m_0 = \texttt{aab}$ and $m_1 = \texttt{abb}$. When given a ciphertext c, it outputs 0 if the first character of c is the same as the second character of c, and outputs 1 otherwise. Compute $\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1]$.

(b) Construct and analyze an adversary $\mathcal{A}'$ for which $\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi} = 1]$ is greater than your answer from part (a).

2.9 In this exercise, we look at different conditions under which the shift, mono-alphabetic substitution, and Vigenère ciphers are perfectly secret:

(a) Prove that if only a single character is encrypted, then the shift cipher is perfectly secret.

(b) What is the largest message space $\mathcal{M}$ for which the mono-alphabetic substitution cipher provides perfect secrecy?

(c) Prove that the Vigenère cipher using (fixed) period t is perfectly secret when used to encrypt messages of length t.

Reconcile this with the attacks shown in the previous chapter.

2.10 Prove that a scheme satisfying Definition 2.5 must have $|\mathcal{K}| \geq |\mathcal{M}|$ without using Lemma 2.4. Specifically, let Π be an arbitrary encryption scheme with $|\mathcal{K}| < |\mathcal{M}|$. Show an $\mathcal{A}$ for which $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] > \frac{1}{2}$.

Hint: It may be easier to let $\mathcal{A}$ be randomized.

2.11 Assume we require only that an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ satisfy the following: For all $m \in \mathcal{M}$, we have $\Pr[\mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m] \geq 2^{-t}$. (This probability is taken over choice of the key as well as any randomness used during encryption.) Show that perfect secrecy can be achieved with $|\mathcal{K}| < |\mathcal{M}|$ when $t \geq 1$. Prove a lower bound on the size of $\mathcal{K}$ in terms of t.

2.12 Let $\varepsilon \geq 0$ be a constant. Say an encryption scheme is *ε-perfectly secret* if for every adversary $\mathcal{A}$ it holds that

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] \leq \frac{1}{2} + \varepsilon\,.$$

(Compare to Definition 2.5.) Show that ε-perfect secrecy can be achieved with $|\mathcal{K}| < |\mathcal{M}|$ when $\varepsilon > 0$. Prove a lower bound on the size of $\mathcal{K}$ in terms of ε.

2.13 In this problem we consider definitions of perfect secrecy for the encryption of *two* messages (using the same key). Here we consider distributions over *pairs* of messages from the message space $\mathcal{M}$; we let M_1, M_2 be random variables denoting the first and second message, respectively. (We stress that these random variables are not assumed to be independent.) We generate a (single) key k, sample a pair of messages (m_1, m_2) according to the given distribution, and then compute ciphertexts $c_1 \leftarrow \mathsf{Enc}_k(m_1)$ and $c_2 \leftarrow \mathsf{Enc}_k(m_2)$; this induces a distribution over pairs of ciphertexts and we let C_1, C_2 be the corresponding random variables.

(a) Say encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *perfectly secret for two messages* if for all distributions over $\mathcal{M} \times \mathcal{M}$, all $m_1, m_2 \in \mathcal{M}$, and all ciphertexts $c_1, c_2 \in \mathcal{C}$ with $\Pr[C_1 = c_1 \wedge C_2 = c_2] > 0$:

$$\begin{aligned}\Pr\left[M_1 = m_1 \wedge M_2 = m_2 \mid C_1 = c_1 \wedge C_2 = c_2\right]& \\ = \Pr[M_1 = m_1 \wedge M_2 = m_2]&.\end{aligned}$$

Prove that *no* encryption scheme can satisfy this definition.

Hint: Take $c_1 = c_2$.

(b) Say encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *perfectly secret for two distinct messages* if for all distributions over $\mathcal{M} \times \mathcal{M}$ where the first and second messages are guaranteed to be different (i.e., distributions over pairs of *distinct* messages), all $m_1, m_2 \in \mathcal{M}$, and all $c_1, c_2 \in \mathcal{C}$ with $\Pr[C_1 = c_1 \wedge C_2 = c_2] > 0$:

$$\begin{aligned}\Pr[M_1 = m_1 \wedge M_2 = m_2 \mid C_1 = c_1 \wedge C_2 = c_2]& \\ = \Pr[M_1 = m_1 \wedge M_2 = m_2]&.\end{aligned}$$

Show an encryption scheme that provably satisfies this definition.

Hint: The encryption scheme you propose need not be efficient, although an efficient solution is possible.

Part II

Private-Key (Symmetric) Cryptography

Chapter 3

Private-Key Encryption

In the previous chapter we saw some fundamental limitations of perfect secrecy. In this chapter we begin our study of modern cryptography by introducing the weaker (but sufficient) notion of *computational* secrecy. We will then show how this definition can be used to bypass the impossibility results shown previously and, in particular, how a short key (say, 128 bits long) can be used to encrypt many long messages (say, gigabytes in total).

Along the way we will study the fundamental notion of *pseudorandomness*, which captures the idea that something can "look" completely random even though it is not. This powerful concept underlies much of modern cryptography, and has applications and implications beyond the field as well.

3.1 Computational Security

In Chapter 2 we introduced the notion of perfect secrecy. While perfect secrecy is a worthwhile goal, it is also unnecessarily strong. Perfect secrecy requires that *absolutely no information* about an encrypted message is leaked, even to an eavesdropper *with unlimited computational power.* For all practical purposes, however, an encryption scheme would still be considered secure if it leaked only a *tiny* amount of information to eavesdroppers with *bounded computational power.* For example, a scheme that leaks information with probability at most 2^{-60} to eavesdroppers investing up to 200 years of computational effort on the fastest available supercomputer is adequate for any real-world application. Security definitions that take into account computational limits on the attacker, and allow for a small probability of failure, are called *computational*, to distinguish them from notions (like perfect secrecy) that are *information-theoretic* in nature. Computational security is now the *de facto* way in which security is defined for all cryptographic purposes.

We stress that although we give up on obtaining perfect security, this does not mean we do away with the rigorous mathematical approach. Definitions and proofs are still essential, and the only difference is that we now consider weaker (but still meaningful) definitions of security.

Computational security incorporates two relaxations relative to information-theoretic notions of security (in the case of encryption, both these relaxations are necessary in order to go beyond the limitations of perfect secrecy discussed in the previous chapter):

1. *Security is only guaranteed against* efficient *adversaries that run for some feasible amount of time.* This means that given enough time (or sufficient computational resources) an attacker may be able to violate security. If we can make the resources required to break the scheme larger than those available to any realistic attacker, then for all practical purposes the scheme is unbreakable.

2. *Adversaries can potentially succeed (i.e., security can potentially fail) with some* very small probability. If we can make this probability sufficiently small, we need not worry about it.

To obtain a meaningful theory, we need to precisely define the above relaxations. There are two general approaches for doing so: the *concrete approach* and the *asymptotic approach.* These are described next.

3.1.1 The Concrete Approach

The concrete approach to computational security quantifies the security of a cryptographic scheme by explicitly bounding the maximum success probability of any (randomized) adversary running for some specified amount of time or, more precisely, investing some specific amount of computational effort. Thus, a concrete definition of security takes roughly the following form:

> *A scheme is* (t, ε)-secure *if any adversary running for time at most* t *succeeds in breaking the scheme with probability at most* ε.

(Of course, the above serves only as a general template, and for the above statement to make sense we need to define exactly what it means to "break" the scheme in question.) As an example, one might have a scheme with the guarantee that no adversary running for at most 200 years using the fastest available supercomputer can succeed in breaking the scheme with probability better than 2^{-60}. Or, it may be more convenient to measure running time in terms of CPU cycles, and to construct a scheme such that no adversary using at most 2^{80} cycles can break the scheme with probability better than 2^{-60}.

It is instructive to get a feel for the large values of t and the small values of ε that are typical of modern cryptographic schemes.

Example 3.1

Modern private-key encryption schemes are generally assumed to give almost optimal security in the following sense: when the key has length n—and so the key space has size 2^n—an adversary running for time t (measured in, say, computer cycles) succeeds in breaking the scheme with probability at most

$ct/2^n$ for some fixed constant c. (This simply corresponds to a brute-force search of the key space, and assumes no preprocessing has been done.)

Assuming $c = 1$ for simplicity, a key of length $n = 60$ provides adequate security against an adversary using a desktop computer. Indeed, on a 4 GHz processor (that executes 4×10^9 cycles per second) 2^{60} CPU cycles require $2^{60}/(4 \times 10^9)$ seconds, or about 9 years. However, the fastest supercomputer at the time of this writing can execute roughly 2×10^{16} floating point operations per second, and 2^{60} such operations require only about 1 minute on such a machine. Taking $n = 80$ would be a more prudent choice; even the computer just mentioned would take about 2 years to carry out 2^{80} operations.

(The above numbers are for illustrative purposes only; in practice $c > 1$, and several other factors—such as the time required for memory access and the possibility of parallel computation on a network of computers—significantly affect the performance of brute-force attacks.)

Today, however, a recommended key length might be $n = 128$. The difference between 2^{80} and 2^{128} is a *multiplicative factor* of 2^{48}. To get a feeling for how big this is, note that according to physicists' estimates the number of seconds since the Big Bang is on the order of 2^{58}.

If the probability that an attacker can successfully recover an encrypted message in one year is at most 2^{-60}, then it is much more likely that the sender and receiver will both be hit by lightning in that same period of time. An event that occurs once every hundred years can be roughly estimated to occur with probability 2^{-30} in any given second. Something that occurs with probability 2^{-60} in any given second is 2^{30} times *less* likely, and might be expected to occur roughly once every 100 billion years. ◇

The concrete approach is important in practice, since concrete guarantees are what users of a cryptographic scheme are ultimately interested in. However, precise concrete guarantees are difficult to provide. Furthermore, one must be careful in interpreting concrete security claims. For example, a claim that no adversary running for 5 years can break a given scheme with probability better than ε begs the questions: what type of computing power (e.g., desktop PC, supercomputer, network of hundreds of computers) does this assume? Does this take into account future advances in computing power (which, by Moore's Law, roughly doubles every 18 months)? Does the estimate assume the use of "off-the-shelf" algorithms, or dedicated software implementations optimized for the attack? Furthermore, such a guarantee says little about the success probability of an adversary running for 2 years (other than the fact that it can be at most ε) and says nothing about the success probability of an adversary running for 10 years.

3.1.2 The Asymptotic Approach

As partly noted above, there are some technical and theoretical difficulties in using the concrete-security approach. These issues must be dealt with in practice, but when concrete security is not an immediate concern it is convenient instead to use an *asymptotic* approach to security; this is the approach taken in this book. This approach, rooted in complexity theory, introduces an integer-valued *security parameter* (denoted by n) that parameterizes both cryptographic schemes as well as all involved parties (namely, the honest parties as well as the attacker). When honest parties initialize a scheme (i.e., when they generate keys), they choose some value n for the security parameter; for the purposes of this discussion, one can think of the security parameter as corresponding to the length of the key. The security parameter is assumed to be known to any adversary attacking the scheme, and we now view the running time of the adversary, as well as its success probability, as functions of the security parameter rather than as concrete numbers. Then:

1. We equate "efficient adversaries" with randomized (i.e., probabilistic) algorithms running in time *polynomial in n*. This means there is some polynomial p such that the adversary runs for time at most $p(n)$ when the security parameter is n. We also require—for real-world efficiency—that honest parties run in polynomial time, although we stress that the adversary may be much more powerful (and run much longer than) the honest parties.

2. We equate the notion of "small probabilities of success" with success probabilities *smaller than any inverse polynomial in n* (see Definition 3.4). Such probabilities are called *negligible*.

Let PPT stand for "probabilistic polynomial-time." A definition of asymptotic security then takes the following general form:

> *A scheme is* secure *if any* PPT *adversary succeeds in breaking the scheme with at most negligible probability.*

This notion of security is *asymptotic* since security depends on the behavior of the scheme for sufficiently large values of n. The following example makes this clear.

Example 3.2

Say we have a scheme that is asymptotically secure. Then it may be the case that an adversary running for n^3 minutes can succeed in "breaking the scheme" with probability $2^{40} \cdot 2^{-n}$ (which is a negligible function of n). When $n \leq 40$ this means that an adversary running for 40^3 minutes (about 6 weeks) can break the scheme with probability 1, so such values of n are not very useful. Even for $n = 50$ an adversary running for 50^3 minutes (about 3 months) can break the scheme with probability roughly $1/1000$, which may not be acceptable. On the other hand, when $n = 500$ an adversary running for 200 years breaks the scheme only with probability roughly 2^{-500}. $\diamondsuit$

As indicated by the previous example, we can view the security parameter as a mechanism that allows the honest parties to "tune" the security of a scheme to some desired level. (Increasing the security parameter also increases the time required to run the scheme, as well as the length of the key, so the honest parties will want to set the security parameter as small as possible subject to defending against the class of attacks they are concerned about.) Viewing the security parameter as the key length, this corresponds roughly to the fact that the time required for an exhaustive-search attack grows exponentially in the length of the key. The ability to "increase security" by increasing the security parameter has important practical ramifications, since it enables honest parties to defend against increases in computing power. The following example gives a sense of how this might play out in practice.

Example 3.3
Let us see the effect that the availability of faster computers might have on security in practice. Say we have a cryptographic scheme in which the honest parties run for $10^6 \cdot n^2$ cycles, and for which an adversary running for $10^8 \cdot n^4$ cycles can succeed in "breaking" the scheme with probability at most $2^{-n/2}$. (The numbers are intended to make calculations easier, and are not meant to correspond to any existing cryptographic scheme.)

Say all parties are using 2 GHz computers and the honest parties set $n = 80$. Then the honest parties run for $10^6 \cdot 6400$ cycles, or 3.2 seconds, and an adversary running for $10^8 \cdot (80)^4$ cycles, or roughly 3 weeks, can break the scheme with probability only 2^{-40}.

Say 8 GHz computers become available, and all parties upgrade. Honest parties can increase n to 160 (which requires generating a fresh key) and maintain a running time of 3.2 seconds (i.e., $10^6 \cdot 160^2$ cycles at $8 \cdot 10^9$ cycles/second). In contrast, the adversary now has to run for over 8 million seconds, or more than 13 weeks, to achieve a success probability of 2^{-80}. The effect of a faster computer has been to make the adversary's job *harder*. $\diamondsuit$

Even when using the asymptotic approach it is important to remember that, ultimately, when a cryptosystem is deployed in practice a concrete security guarantee will be needed. (After all, one must decide on some value of n.) As the above examples indicate, however, it is generally the case that an asymptotic security claim can be translated into a concrete security bound for any desired value of n.

The Asymptotic Approach in Detail

We now discuss more formally the notions of "polynomial-time algorithms" and "negligible success probabilities."

Efficient algorithms. We have defined an algorithm to be efficient if it runs in polynomial time. An algorithm A runs in polynomial time if there exists a polynomial p such that, for every input $x \in \{0,1\}^*$, the computation of $A(x)$ terminates within at most $p(|x|)$ steps. (Here, $|x|$ denotes the length of the string x.) As mentioned earlier, we are only interested in adversaries whose running time is polynomial in the security parameter n. Since we measure the running time of an algorithm in terms of the length of its input, we sometimes provide algorithms with the security parameter written in unary (i.e., as 1^n, or a string of n ones) as input. Parties (or, more precisely, the algorithms they run) may take other inputs besides the security parameter—for example, a message to be encrypted—and we allow their running time to be polynomial in the (total) length of their inputs.

By default, we allow all algorithms to be probabilistic (or randomized). Any such algorithm may "toss a coin" at each step of its execution; this is a metaphorical way of saying that the algorithm can access an unbiased random bit at each step. Equivalently, we can view a randomized algorithm as one that, in addition to its input, is given a uniformly distributed *random tape* of sufficient length[1] whose bits it can use, as needed, throughout its execution.

We consider randomized algorithms by default for two reasons. First, randomness is essential to cryptography (e.g., in order to choose random keys and so on) and so honest parties must be probabilistic; given this, it is natural to allow adversaries to be probabilistic as well. Second, randomization is practical and—as far as we know—gives attackers additional power. Since our goal is to model *all* realistic attacks, we prefer a more liberal definition of efficient computation.

Negligible success probability. A negligible function is one that is asymptotically smaller than any inverse polynomial function. Formally:

DEFINITION 3.4 *A function f from the natural numbers to the non-negative real numbers is* negligible *if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.*

For shorthand, the above is also stated as follows: for every polynomial p and *all sufficiently large values of n* it holds that $f(n) < \frac{1}{p(n)}$. An equivalent formulation of the above is to require that for all constants c there exists an N such that for all $n > N$ it holds that $f(n) < n^{-c}$. We typically denote an arbitrary negligible function by negl.

Example 3.5

The functions $2^{-n}, 2^{-\sqrt{n}}$, and $n^{-\log n}$ are all negligible. However, they approach zero at very different rates. For example, we can look at the minimum value of n for which each function is smaller than $1/n^5$:

[1]If the algorithm in question runs for $p(n)$ steps on inputs of length n, then a random tape of length $p(n)$ is sufficient since the attacker can read at most one random bit per time step.

1. Solving $2^{-n} < n^{-5}$ we get $n > 5 \log n$. The smallest integer value of n for which this holds is $n = 23$.
2. Solving $2^{-\sqrt{n}} < n^{-5}$ we get $n > 25 \log^2 n$. The smallest integer value of n for which this holds is $n \approx 3500$.
3. Solving $n^{-\log n} < n^{-5}$ we get $\log n > 5$. The smallest integer value of n for which this holds is $n = 33$.

From the above you may have the impression that $n^{-\log n}$ approaches zero more quickly than $2^{-\sqrt{n}}$. However, this is incorrect; for all $n > 65536$ it holds that $2^{-\sqrt{n}} < n^{-\log n}$. Nevertheless, this does show that for values of n in the hundreds or thousands, an adversarial success probability of $n^{-\log n}$ is preferable to an adversarial success probability of $2^{-\sqrt{n}}$. $\diamondsuit$

A technical advantage of working with negligible success probabilities is that they obey certain closure properties. The following is an easy exercise.

PROPOSITION 3.6 *Let* negl_1 *and* negl_2 *be negligible functions. Then,*

1. *The function* negl_3 *defined by* $\mathsf{negl}_3(n) = \mathsf{negl}_1(n) + \mathsf{negl}_2(n)$ *is negligible.*
2. *For any positive polynomial* p*, the function* negl_4 *defined by* $\mathsf{negl}_4(n) = p(n) \cdot \mathsf{negl}_1(n)$ *is negligible.*

The second part of the above proposition implies that if a certain event occurs with only negligible probability in a certain experiment, then the event occurs with negligible probability even if the experiment is repeated polynomially many times. (This relies on the union bound; see Proposition A.7.) For example, the probability that n fair coin flips all come up "heads" is negligible. This means that even if we repeat the experiment of flipping n coins polynomially many times, the probability that *any* of those experiments result in n heads is still negligible.

A corollary of the second part of the above proposition is that if a function g is *not* negligible, then neither is the function $f(n) \overset{\text{def}}{=} g(n)/p(n)$ for any positive polynomial p.

Asymptotic Security: A Summary

Any security definition consists of two parts: a definition of what is considered a "break" of the scheme, and a specification of the power of the adversary. The power of the adversary can relate to many issues (e.g., in the case of encryption, whether we assume a ciphertext-only attack or a chosen-plaintext attack). However, when it comes to the *computational* power of the adversary, we will from now on model the adversary as efficient and thus only consider adversarial strategies that can be implemented in probabilistic polynomial time. Definitions will also always be formulated so that a break that occurs with negligible probability is not considered significant. Thus, the general framework of any security definition will be as follows:

> A scheme is *secure* if for every *probabilistic polynomial-time* adversary $\mathcal{A}$ carrying out an attack of some formally specified type, the probability that $\mathcal{A}$ succeeds in the attack (where success is also formally specified) is *negligible.*

Such a definition is *asymptotic* because it is possible that for small values of n an adversary can succeed with high probability. In order to see this in more detail, we expand the term "negligible" in the above statement:

> A scheme is *secure* if for every PPT adversary $\mathcal{A}$ carrying out an attack of some formally specified type, and for every positive polynomial p, there exists an integer N such that when $n > N$ the probability that $\mathcal{A}$ succeeds in the attack is less than $\frac{1}{p(n)}$.

Note that nothing is guaranteed for values $n \leq N$.

On the Choices Made in Defining Asymptotic Security

In defining the general notion of asymptotic security, we have made two choices: we have identified efficient adversarial strategies with the class of *probabilistic, polynomial-time algorithms*, and have equated small chances of success with *negligible probabilities.* Both of these choices are—to some extent—arbitrary, and one could build a perfectly reasonable theory by defining, say, efficient strategies as those running in quadratic time, or small success probabilities as those bounded by 2^{-n}. Nevertheless, we briefly justify the choices we have made (which are the standard ones).

Those familiar with complexity theory or algorithms will recognize that the idea of equating efficient computation with (probabilistic) polynomial-time algorithms is not unique to cryptography. One advantage of using (probabilistic) polynomial time as our measure of efficiency is that this frees us from having to precisely specify our model of computation, since the extended Church–Turing thesis states that all "reasonable" models of computation are polynomially equivalent. Thus, we need not specify whether we use Turing machines, boolean circuits, or random-access machines; we can present algorithms in high-level pseudocode and be confident that if our analysis shows that these algorithms run in polynomial time, then any reasonable implementation will also.

Another advantage of (probabilistic) polynomial-time algorithms is that they satisfy desirable closure properties: in particular, an algorithm that makes polynomially many calls to a polynomial-time subroutine (and does only polynomial computation in addition) will itself run in polynomial time.

The most important feature of negligible probabilities is the closure property we have already seen in Proposition 3.6(2): any polynomial times a negligible function is still negligible. This means, in particular, that if an algorithm makes polynomially many calls to some subroutine that "fails" with negligible probability each time it is called, then the probability that any of the calls to that subroutine fail is still negligible.

Necessity of the Relaxations

Computational secrecy introduces two relaxations of perfect secrecy: first, security is guaranteed only against efficient adversaries; second, a small probability of success is allowed. Both these relaxations are essential for achieving practical encryption schemes, and in particular for bypassing the negative results for perfectly secret encryption. We informally discuss why this is the case. Assume we have an encryption scheme where the size of the key space $\mathcal{K}$ is much smaller than the size of the message space $\mathcal{M}$. (As shown in the previous chapter, this means the scheme cannot be perfectly secret.) Two attacks apply regardless of how the encryption scheme is constructed:

- Given a ciphertext c, an adversary can decrypt c using all keys $k \in \mathcal{K}$. This gives a list of all the messages to which c can possibly correspond. Since this list cannot contain all of $\mathcal{M}$ (because $|\mathcal{K}| < |\mathcal{M}|$), this attack leaks *some* information about the message that was encrypted.

  Moreover, say the adversary carries out a known-plaintext attack and learns that ciphertexts $c_1, \ldots, c_\ell$ correspond to the messages $m_1, \ldots, m_\ell$, respectively. The adversary can again try decrypting each of these ciphertexts with all possible keys until it finds a key k for which $\mathsf{Dec}_k(c_i) = m_i$ for all i. Later, given a ciphertext c that is the encryption of an unknown message m, it is almost surely the case that $\mathsf{Dec}_k(c) = m$.

  Exhaustive-search attacks like the above allow an adversary to succeed with probability essentially 1 in time linear in $|\mathcal{K}|$.

- Consider again the case where the adversary learns that ciphertexts $c_1, \ldots, c_\ell$ correspond to messages $m_1, \ldots, m_\ell$. The adversary can *guess* a uniform key $k \in \mathcal{K}$ and check to see whether $\mathsf{Dec}_k(c_i) = m_i$ for all i. If so, then, as above, the attacker can use k to decrypt anything subsequently encrypted by the honest parties.

  Here the adversary runs in essentially constant time and succeeds with nonzero (though very small) probability $1/|\mathcal{K}|$.

It follows that if we wish to encrypt many messages using a single short key, security can only be achieved if we limit the running time of the adversary (so the adversary does not have sufficient time to carry out a brute-force search) and are willing to allow a very small probability of success (so the second "attack" is ruled out).

3.2 Defining Computationally Secure Encryption

Given the background of the previous section, we are ready to present a definition of computational security for private-key encryption. First, we redefine the *syntax* of private-key encryption; this will be essentially the same as the syntax introduced in Chapter 2 except that we now explicitly take into account the security parameter n. We also allow the decryption algorithm to output an error message in case it is presented with an invalid ciphertext. Finally, by default, we let the message space be the set $\{0,1\}^*$ of all (finite-length) binary strings.

DEFINITION 3.7 *A* private-key encryption scheme *is a tuple of probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

1. *The* key-generation algorithm Gen *takes as input* 1^n *(i.e., the security parameter written in unary) and outputs a key k; we write* $k \leftarrow \mathsf{Gen}(1^n)$ *(emphasizing that* Gen *is a randomized algorithm). We assume without loss of generality that any key k output by* $\mathsf{Gen}(1^n)$ *satisfies* $|k| \geq n$.

2. *The* encryption algorithm Enc *takes as input a key k and a plaintext message* $m \in \{0,1\}^*$*, and outputs a ciphertext c. Since* Enc *may be randomized, we write this as* $c \leftarrow \mathsf{Enc}_k(m)$.

3. *The* decryption algorithm Dec *takes as input a key k and a ciphertext c, and outputs a message m or an error. We assume that* Dec *is deterministic, and so write* $m := \mathsf{Dec}_k(c)$ *(assuming here that* Dec *does not return an error). We denote a generic error by the symbol* $\perp$.

It is required that for every n, every key k output by $\mathsf{Gen}(1^n)$*, and every* $m \in \{0,1\}^*$*, it holds that* $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$.

If $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is such that for k output by* $\mathsf{Gen}(1^n)$*, algorithm* Enc_k *is only defined for messages* $m \in \{0,1\}^{\ell(n)}$*, then we say that* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a* fixed-length private-key encryption scheme for messages of length $\ell(n)$.

Almost always, $\mathsf{Gen}(1^n)$ simply outputs a uniform n-bit string as the key. When this is the case, we will omit Gen and simply define a private-key encryption scheme by a *pair* of algorithms $(\mathsf{Enc}, \mathsf{Dec})$.

The above definition considers *stateless* schemes, in which each invocation of Enc (and Dec) is independent of all prior invocations. Later in the chapter, we will occasionally discuss *stateful* schemes in which the sender (and possibly the receiver) is required to maintain state across invocations. Unless explicitly noted otherwise, all our results assume stateless encryption/decryption.

3.2.1 The Basic Definition of Security

We begin by presenting the most basic notion of security for private-key encryption: security against a ciphertext-only attack where the adversary observes only a *single* ciphertext or, equivalently, security when a given key is used to encrypt just a *single* message. We consider stronger definitions of security later in the chapter.

Motivating the definition. As we have already discussed, any definition of security consists of two distinct components: a threat model (i.e., a specification of the assumed power of the adversary) and a security goal (usually specified by describing what constitutes a "break" of the scheme). We begin our definitional treatment by considering the simplest threat model, where we have an *eavesdropping adversary* who observes the encryption of a single message. This is exactly the threat model that was considered in the previous chapter with the exception that, as explained in the previous section, we are now interested only in adversaries that are computationally bounded and so limited to running in polynomial time.

Although we have made two assumptions about the adversary's capabilities (namely, that it only eavesdrops, and that it runs in polynomial time), we make no assumptions whatsoever about the adversary's *strategy* in trying to decipher the ciphertext it observes. This is crucial for obtaining meaningful notions of security; the definition ensures protection against *any* computationally bounded adversary, regardless of the algorithm it uses.

Correctly defining the security goal for encryption is not trivial, but we have already discussed this issue at length in Section 1.4.1 and in the previous chapter. We therefore just recall that the idea behind the definition is that the adversary should be unable to learn *any partial information* about the plaintext from the ciphertext. The definition of *semantic security* (cf. Section 3.2.2) exactly formalizes this notion, and was the first definition of computationally secure encryption to be proposed. Semantic security is complex and difficult to work with. Fortunately, there is an equivalent definition called *indistinguishability* that is much simpler.

The definition of indistinguishability is patterned on the alternative definition of perfect secrecy given as Definition 2.5. (This serves as further motivation that the definition of indistinguishability is a good one.) Recall that Definition 2.5 considers an experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ in which an adversary $\mathcal{A}$ outputs two messages m_0 and m_1, and is then given an encryption of one of those messages using a uniform key. The definition states that a scheme Π is secure if no adversary $\mathcal{A}$ can determine which of the messages m_0, m_1 was encrypted with probability any different from $1/2$, which is the probability that $\mathcal{A}$ is correct if it just makes a random guess.

Here, we keep the experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ almost exactly the same (except for some technical differences discussed below), but introduce two key modifications in the definition itself:

1. We now consider only adversaries running in *polynomial time*, whereas Definition 2.5 considered even adversaries with unbounded running time.
2. We now concede that the adversary might determine the encrypted message with probability *negligibly better than* $1/2$.

As discussed extensively in the previous section, the above relaxations constitute the core elements of computational security.

As for the other differences, the most prominent is that we now parameterize the experiment by a security parameter n. We then measure both the running time of the adversary $\mathcal{A}$ as well as its success probability as functions of n. We write $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ to denote the experiment being run with security parameter n, and write

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \tag{3.1}$$

to denote the probability that the output of experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ is 1. Note that with $\mathcal{A}, \Pi$ fixed, Equation (3.1) is a function of n.

A second difference in experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ is that we now explicitly require the adversary to output two messages m_0, m_1 *of equal length.* (In Definition 2.5 this requirement is implicit if the message space $\mathcal{M}$ only contains messages of some fixed length, as is the case for the one-time pad encryption scheme.) This means that, by default, we do not require a secure encryption scheme to hide the length of the plaintext. We revisit this point at the end of this section; see also Exercises 3.2 and 3.3.

Indistinguishability in the presence of an eavesdropper. We now give the formal definition, beginning with the experiment outlined above. The experiment is defined for any private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, any adversary $\mathcal{A}$, and any value n for the security parameter:

The adversarial indistinguishability experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

1. *The adversary $\mathcal{A}$ is given input 1^n, and outputs a pair of messages m_0, m_1 with $|m_0| = |m_1|$.*
2. *A key k is generated by running $\mathsf{Gen}(1^n)$, and a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$. We refer to c as the* challenge ciphertext.
3. *$\mathcal{A}$ outputs a bit b'.*
4. *The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. If $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1$, we say that $\mathcal{A}$* succeeds.

There is no limitation on the lengths of m_0 and m_1, as long as they are the same. (Of course, if $\mathcal{A}$ runs in polynomial time, then m_0 and m_1 have length polynomial in n.) If Π is a fixed-length scheme for messages of length $\ell(n)$, the above experiment is modified by requiring $m_0, m_1 \in \{0,1\}^{\ell(n)}$.

The fact that the adversary can only eavesdrop is implicit in the fact that its input is limited to a (single) ciphertext, and the adversary does not have any further interaction with the sender or the receiver. (As we will see later, allowing additional interaction makes the adversary significantly stronger.)

The definition of indistinguishability states that an encryption scheme is secure if no PPT adversary $\mathcal{A}$ succeeds in guessing which message was encrypted in the above experiment with probability significantly better than random guessing (which is correct with probability 1/2):

DEFINITION 3.8 *A private-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions in the presence of an eavesdropper*, or is* EAV-secure*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that, for all* n*,*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the randomness used by $\mathcal{A}$ *and the randomness used in the experiment (for choosing the key and the bit* b*, as well as any randomness used by* Enc*).*

Note: unless otherwise qualified, when we write "$f(n) \leq g(n)$" we mean that inequality holds for all n.

It should be clear that Definition 3.8 is *weaker* than Definition 2.5, which is equivalent to perfect secrecy. Thus, any perfectly secret encryption scheme has indistinguishable encryptions in the presence of an eavesdropper. Our goal, therefore, will be to show that there exist encryption schemes satisfying the above in which the key is shorter than the message. That is, we will show schemes that satisfy Definition 3.8 but cannot satisfy Definition 2.5.

An equivalent formulation. Definition 3.8 requires that no PPT adversary can determine which of two messages was encrypted, with probability significantly better than 1/2. An equivalent formulation is that every PPT adversary *behaves the same* whether it sees an encryption of m_0 or of m_1. Since $\mathcal{A}$ outputs a single bit, "behaving the same" means it outputs 1 with almost the same probability in each case. To formalize this, define $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n, b)$ as above except that the fixed bit b is used (rather than being chosen at random). Let $\mathsf{out}_{\mathcal{A}}(\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n, b))$ denote the output bit b' of $\mathcal{A}$ in the experiment. The following essentially states that no $\mathcal{A}$ can determine whether it is running in experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n, 0)$ or experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n, 1)$.

DEFINITION 3.9 *A private-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions in the presence of an eavesdropper *if for all* PPT *adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\left|\Pr[\mathsf{out}_{\mathcal{A}}(\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n, 0)) = 1] - \Pr[\mathsf{out}_{\mathcal{A}}(\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n, 1)) = 1]\right| \leq \mathsf{negl}(n).$$

The fact that this is equivalent to Definition 3.8 is left as an exercise.

Encryption and Plaintext Length

The default notion of secure encryption does not require the encryption scheme to hide the plaintext length and, in fact, all commonly used encryption schemes reveal the plaintext length (or a close approximation thereof). The main reason for this is that it is *impossible* to support arbitrary-length messages while hiding all information about the plaintext length (cf. Exercise 3.2). In many cases this is inconsequential since the plaintext length is already public or is not sensitive. This is not always the case, however, and sometimes leaking the plaintext length is problematic. As examples:

- *Simple numeric/text data:* Say the encryption scheme being used reveals the plaintext length exactly. Then encrypted salary information would reveal whether someone makes a 5-figure or a 6-figure salary. Similarly, encryption of "yes"/"no" responses would leak the answer exactly.

- *Auto-suggestions:* Websites often include an "auto-complete" or "auto-suggestion" functionality by which the webserver suggests a list of potential words or phrases based on partial information the user has already typed. The *size* of this list can reveal information about the letters the user has typed so far. (For example, the number of auto-completions returned for "`th`" is far greater than the number for "`zo`.")

- *Database searches:* Consider a user querying a database for all records matching some search term. The *number of records returned* can reveal a lot of information about what the user was searching for. This can be particularly damaging if the user is searching for medical information and the query reveals information about a disease the user has.

- *Compressed data:* If the plaintext is compressed before being encrypted, then information about the plaintext might be revealed even if only fixed-length data is ever encrypted. (Such an encryption scheme would therefore not satisfy Definition 3.8.) For example, a short compressed plaintext would indicate that the original (uncompressed) plaintext has a lot of redundancy. If an adversary can control a portion of what gets encrypted, this vulnerability can enable an adversary to learn additional information about the plaintext; it has been shown possible to use an attack of exactly this sort (the *CRIME attack*) against encrypted HTTP traffic to reveal secret session cookies.

When using encryption one should determine whether leaking the plaintext length is a concern and, if so, take steps to mitigate or prevent such leakage by padding all messages to some pre-determined length before encrypting them.

3.2.2 *Semantic Security

We motivated the definition of secure encryption by saying that it should be infeasible for an adversary to learn any partial information about the plaintext from the ciphertext. However, the definition of indistinguishability looks very different. As we have mentioned, Definition 3.8 is equivalent to a definition called *semantic security* that formalizes the notion that partial information cannot be learned. We build up to that definition by discussing two weaker notions and showing that they are implied by indistinguishability.

We begin by showing that indistinguishability means that ciphertexts leak no information about individual bits of the plaintext. Formally, say encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is EAV-secure (recall then when Gen is omitted, the key is a uniform n-bit string), and $m \in \{0,1\}^\ell$ is uniform. Then we show that for any index i, it is infeasible to guess m^i from $\mathsf{Enc}_k(m)$ (where, in this section, m^i denotes the ith bit of m) with probability much better than 1/2.

THEOREM 3.10 *Let* $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ *be a fixed-length private-key encryption scheme for messages of length* ℓ *that has indistinguishable encryptions in the presence of an eavesdropper. Then for all* PPT *adversaries* $\mathcal{A}$ *and any* $i \in \{1, \ldots, \ell\}$*, there is a negligible function* negl *such that*

$$\Pr\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m)) = m^i\right] \le \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over uniform $m \in \{0,1\}^\ell$ *and* $k \in \{0,1\}^n$*, the randomness of* $\mathcal{A}$*, and the randomness of* Enc*.*

PROOF The idea behind the proof of this theorem is that if it were possible to guess the ith bit of m from $\mathsf{Enc}_k(m)$, then it would also be possible to distinguish between encryptions of messages m_0 and m_1 whose ith bits differ. We formalize this via a *proof by reduction*, in which we show how to use any efficient adversary $\mathcal{A}$ to construct an efficient adversary $\mathcal{A}'$ such that if $\mathcal{A}$ violates the security notion of the theorem for Π, then $\mathcal{A}'$ violates the definition of indistinguishability for Π. (See Section 3.3.2.) Since Π has indistinguishable encryptions, it must also be secure in the sense of the theorem.

Fix an arbitrary PPT adversary $\mathcal{A}$ and $i \in \{1, \ldots, \ell\}$. Let $I_0 \subset \{0,1\}^\ell$ be the set of all strings whose ith bit is 0, and let $I_1 \subset \{0,1\}^\ell$ be the set of all strings whose ith bit is 1. We have

$$\begin{aligned}&\Pr\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m)) = m^i\right] \\ &= \frac{1}{2} \cdot \Pr_{m_0 \leftarrow I_0}\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m_0)) = 0\right] + \frac{1}{2} \cdot \Pr_{m_1 \leftarrow I_1}\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m_1)) = 1\right].\end{aligned}$$

Construct the following eavesdropping adversary $\mathcal{A}'$:

Adversary $\mathcal{A}'$:
1. Choose uniform $m_0 \in I_0$ and $m_1 \in I_1$. Output m_0, m_1.
2. Upon observing a ciphertext c, invoke $\mathcal{A}(1^n, c)$. If $\mathcal{A}$ outputs 0, output $b' = 0$; otherwise, output $b' = 1$.

$\mathcal{A}'$ runs in polynomial time since $\mathcal{A}$ does.

By the definition of experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)$, we have that $\mathcal{A}'$ succeeds if and only if $\mathcal{A}$ outputs b upon receiving $\mathsf{Enc}_k(m_b)$. So

$$\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)=1\right]\\
&\ = \Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_b))=b\right]\\
&\ = \frac{1}{2}\cdot \Pr_{m_0\leftarrow I_0}\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_0))=0\right] + \frac{1}{2}\cdot \Pr_{m_1\leftarrow I_1}\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_1))=1\right]\\
&\ = \Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=m^i\right].
\end{aligned}$$

By the assumption that $(\mathsf{Enc},\mathsf{Dec})$ has indistinguishable encryptions in the presence of an eavesdropper, there is a negligible function negl such that $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)=1\right]\le\frac{1}{2}+\mathsf{negl}(n)$. We conclude that

$$\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=m^i\right]\le\frac{1}{2}+\mathsf{negl}(n),$$

completing the proof. ∎

We next claim, roughly, that indistinguishability means that no PPT adversary can learn *any* function of the plaintext given the ciphertext, regardless of the distribution of the message being sent. This is intended to capture the idea that no information about a plaintext is leaked by the resulting ciphertext. This requirement is, however, non-trivial to define formally. To see why, note that even for the case considered above, it is easy to compute the ith bit of m if m is chosen, say, uniformly from the set of all strings whose ith bit is 0 (rather than uniformly from $\{0,1\}^\ell$). Thus, what we actually want to say is that if there exists any adversary who correctly computes $f(m)$ with some probability when given $\mathsf{Enc}_k(m)$, then there exists an adversary that can correctly compute $f(m)$ with the same probability *without* being given the ciphertext at all (and only knowing the distribution of m). In what follows we focus on the case when m is chosen uniformly from some set $S\subseteq\{0,1\}^\ell$.

THEOREM 3.11 *Let* $(\mathsf{Enc},\mathsf{Dec})$ *be a fixed-length private-key encryption scheme for messages of length* ℓ *that has indistinguishable encryptions in the presence of an eavesdropper. Then for any* PPT *algorithm* $\mathcal{A}$ *there is a* PPT *algorithm* $\mathcal{A}'$ *such that for any* $S\subseteq\{0,1\}^\ell$ *and any function* $f:\{0,1\}^\ell\to\{0,1\}$*, there is a negligible function* negl *such that:*

$$\left|\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=f(m)\right]-\Pr\left[\mathcal{A}'(1^n)=f(m)\right]\right|\le\mathsf{negl}(n),$$

where the first probability is taken over uniform choice of $k\in\{0,1\}^n$ *and* $m\in S$*, the randomness of* $\mathcal{A}$*, and the randomness of* Enc*, and the second probability is taken over uniform choice of* $m\in S$ *and the randomness of* $\mathcal{A}'$*.*

PROOF (Sketch) The fact that $(\mathsf{Enc}, \mathsf{Dec})$ is EAV-secure means that, for any $S \subseteq \{0,1\}^\ell$, no PPT adversary can distinguish between $\mathsf{Enc}_k(m)$ (for uniform $m \in S$) and $\mathsf{Enc}_k(1^\ell)$. Consider now the probability that $\mathcal{A}$ successfully computes $f(m)$ given $\mathsf{Enc}_k(m)$. We claim that $\mathcal{A}$ should successfully compute $f(m)$ given $\mathsf{Enc}_k(1^\ell)$ with almost the same probability; otherwise, $\mathcal{A}$ could be used to distinguish between $\mathsf{Enc}_k(m)$ and $\mathsf{Enc}_k(1^\ell)$. The distinguisher is easily constructed: choose uniform $m \in S$, and output $m_0 = m$, $m_1 = 1^\ell$. When given a ciphertext c that is an encryption of either m_0 or m_1, invoke $\mathcal{A}(1^n, c)$ and output 0 if and only if $\mathcal{A}$ outputs $f(m)$. If $\mathcal{A}$ outputs $f(m)$ when given an encryption of m with probability that is significantly different from the probability that it outputs $f(m)$ when given an encryption of 1^ℓ, then the described distinguisher violates Definition 3.9.

The above suggests the following algorithm $\mathcal{A}'$ that does not receive $c = \mathsf{Enc}_k(m)$, yet computes $f(m)$ almost as well as $\mathcal{A}$ does: $\mathcal{A}'(1^n)$ chooses a uniform key $k \in \{0,1\}^n$, invokes $\mathcal{A}$ on $c \leftarrow \mathsf{Enc}_k(1^\ell)$, and outputs whatever $\mathcal{A}$ does. By the above, we have that $\mathcal{A}$ outputs $f(m)$ when run as a subroutine by $\mathcal{A}'$ with almost the same probability as when it receives $\mathsf{Enc}_k(m)$. Thus, $\mathcal{A}'$ fulfills the property required by the claim. ■

Semantic security. The full definition of semantic security guarantees considerably more than the property considered in Theorem 3.11. The definition allows the length of the plaintext to depend on the security parameter, and allows for essentially arbitrary distributions over plaintexts. (Actually, we allow only *efficiently sampleable* distributions. This means that there is some probabilistic polynomial-time algorithm Samp such that $\mathsf{Samp}(1^n)$ outputs messages according to the distribution.) The definition also takes into account arbitrary "external" information $h(m)$ about the plaintext that may be leaked to the adversary through other means (e.g., because the same message m is used for some other purpose as well).

DEFINITION 3.12 *A private-key encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$ *is* semantically secure in the presence of an eavesdropper *if for every* PPT *algorithm* $\mathcal{A}$ *there exists a* PPT *algorithm* $\mathcal{A}'$ *such that for any* PPT *algorithm* Samp *and polynomial-time computable functions* f *and* h*, the following is negligible:*

$$\left| \Pr[\mathcal{A}(1^n, \mathsf{Enc}_k(m), h(m)) = f(m)] - \Pr[\mathcal{A}'(1^n, |m|, h(m)) = f(m)] \right|,$$

where the first probability is taken over uniform $k \in \{0,1\}^n$*,* m *output by* $\mathsf{Samp}(1^n)$*, the randomness of* $\mathcal{A}$*, and the randomness of* Enc*, and the second probability is taken over* m *output by* $\mathsf{Samp}(1^n)$ *and the randomness of* $\mathcal{A}'$*.*

The adversary $\mathcal{A}$ is given the ciphertext $\mathsf{Enc}_k(m)$ as well as the external information $h(m)$, and attempts to guess the value of $f(m)$. Algorithm $\mathcal{A}'$ also attempts to guess the value of $f(m)$, but is given *only* $h(m)$ and the length of m. The security requirement states that $\mathcal{A}$'s probability of correctly guessing $f(m)$ is about the same as that of $\mathcal{A}'$. Intuitively, then, the ciphertext $\mathsf{Enc}_k(m)$ does not reveal any additional information about the value of $f(m)$.

Definition 3.12 constitutes a very strong and convincing formulation of the security guarantees that should be provided by an encryption scheme. However, it is easier to work with the definition of indistinguishability (Definition 3.8). Fortunately, the definitions are *equivalent*:

THEOREM 3.13 *A private-key encryption scheme has indistinguishable encryptions in the presence of an eavesdropper if and only if it is semantically secure in the presence of an eavesdropper.*

Looking ahead, a similar equivalence between semantic security and indistinguishability is known for all the definitions that we present in this chapter as well as those in Chapter 11. We can therefore use indistinguishability as our working definition, while being assured that the guarantees achieved are those of semantic security.

3.3 Constructing Secure Encryption Schemes

Having defined what it means for an encryption scheme to be secure, the reader may expect us to turn immediately to constructions of secure encryption schemes. Before doing so, however, we need to introduce the notions of *pseudorandom generators* (PRGs) and *stream ciphers*, important building blocks for private-key encryption. These, in turn, will lead to a discussion of *pseudorandomness*, which plays a fundamental role in cryptography in general and private-key encryption in particular.

3.3.1 Pseudorandom Generators and Stream Ciphers

A pseudorandom generator G is an efficient, deterministic algorithm for transforming a short, uniform string called the *seed* into a longer, "uniform-looking" (or "pseudorandom") output string. Stated differently, a pseudorandom generator uses a small amount of true randomness in order to generate a large amount of pseudorandomness. This is useful whenever a large number of random(-looking) bits are needed, since generating true random bits is difficult and slow. (See the discussion at the beginning of Chapter 2.) Indeed, pseudorandom generators have been studied since at least the 1940s when they were proposed for running statistical simulations. In that context, researchers proposed various statistical tests that a pseudorandom generator should pass in order to be considered "good." As a simple example, the first bit of the output of a pseudorandom generator should be equal to 1 with probability very close to 1/2 (where the probability is taken over uniform choice of the seed), since the first bit of a uniform string is equal to 1 with probability exactly 1/2. In fact, the parity of any fixed subset of the output bits should also be 1 with probability very close to 1/2. More complex statistical tests can also be considered.

This historical approach to determining the quality of some candidate pseudorandom generator is ad hoc, and it is not clear when passing some set of statistical tests is sufficient to guarantee the soundness of using a candidate pseudorandom generator for some application. (In particular, there may be another statistical test that *does* successfully distinguish the output of the generator from true random bits.) The historical approach is even more problematic when using pseudorandom generators for cryptographic applications; in that setting, security may be compromised if an attacker is able to distinguish the output of a generator from uniform, and we do not know in advance what strategy an attacker might use.

The above considerations motivated a cryptographic approach to defining pseudorandom generators in the 1980s. The basic realization was that a good pseudorandom generator should pass *all* (efficient) statistical tests. That is, for *any* efficient statistical test (or *distinguisher*) D, the probability that D returns 1 when given the output of the pseudorandom generator should be close to the probability that D returns 1 when given a uniform string of the same length. Informally, then, the output of a pseudorandom generator should "look like" a uniform string to *any* efficient observer.

(We stress that, formally speaking, it does not make sense to say that any fixed string is "pseudorandom," in the same way that it is meaningless to refer to any fixed string as "random." Rather, pseudorandomness is a property of a *distribution* on strings. Nevertheless, we sometimes informally call a string sampled according to the uniform distribution a "uniform string," and a string output by a pseudorandom generator a "pseudorandom string.")

Another perspective is obtained by defining what it means for a distribution to be pseudorandom. Let Dist be a distribution on ℓ-bit strings. (This means that Dist assigns some probability to every string in $\{0,1\}^\ell$; sampling from Dist means that we choose an ℓ-bit string according to this probability distribution.) Informally, Dist is *pseudorandom* if the experiment in which a string is sampled from Dist is indistinguishable from the experiment in which a uniform string of length ℓ is sampled. (Strictly speaking, since we are in an asymptotic setting we need to speak of the pseudorandomness of a *sequence* of distributions $\mathsf{Dist} = \{\mathsf{Dist}_n\}$, where distribution Dist_n is used for security parameter n. We ignore this point in our current discussion.) More precisely, it should be infeasible for any polynomial-time algorithm to tell (better than guessing) whether it is given a string sampled according to Dist, or whether it is given a uniform ℓ-bit string. This means that *a pseudorandom string is just as good as a uniform string*, as long as we consider only polynomial-time observers. Just as indistinguishability is a computational relaxation of perfect secrecy, pseudorandomness is a computational relaxation of true randomness. (We will generalize this perspective when we discuss the notion of indistinguishability in Chapter 7.)

Now let $G : \{0,1\}^n \to \{0,1\}^\ell$ be a function, and define Dist to be the distribution on ℓ-bit strings obtained by choosing a uniform $s \in \{0,1\}^n$ and outputting $G(s)$. Then G is a pseudorandom generator if and only if the distribution Dist is pseudorandom.

The formal definition. As discussed above, G is a pseudorandom generator if no efficient distinguisher can detect whether it is given a string output by G or a string chosen uniformly at random. As in Definition 3.9, this is formalized by requiring that every efficient algorithm outputs 1 with almost the same probability when given $G(s)$ (for uniform seed s) or a uniform string. (For an equivalent definition analogous to Definition 3.8, see Exercise 3.5.) We obtain a definition in the asymptotic setting by letting the security parameter n determine the length of the seed. We then insist that G be computable by an efficient algorithm. As a technicality, we also require that G's output be longer than its input; otherwise, G is not very useful or interesting.

DEFINITION 3.14 *Let ℓ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the result $G(s)$ is a string of length $\ell(n)$. We say that G is a* pseudorandom generator *if the following conditions hold:*

1. **(Expansion:)** *For every n it holds that $\ell(n) > n$.*
2. **(Pseudorandomness:)** *For any* PPT *algorithm D, there is a negligible function* negl *such that*

$$\left|\Pr[D(G(s)) = 1] - \Pr[D(r) = 1]\right| \leq \mathsf{negl}(n),$$

where the first probability is taken over uniform choice of $s \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $r \in \{0,1\}^{\ell(n)}$ and the randomness of D.

We call ℓ the expansion factor *of G.*

We give an example of an insecure pseudorandom generator to gain familiarity with the definition.

Example 3.15

Define $G(s)$ to output s followed by $\oplus_{i=1}^{n} s_i$, so the expansion factor of G is $\ell(n) = n + 1$. The output of G can easily be distinguished from uniform. Consider the following efficient distinguisher D: on input a string w, output 1 if and only if the final bit of w is equal to the XOR of all the preceding bits of w. Since this property holds for all strings output by G, we have

$\Pr[D(G(s)) = 1] = 1$. On the other hand, if w is uniform, the final bit of w is uniform and so $\Pr[D(w) = 1] = \frac{1}{2}$. The quantity $|\frac{1}{2} - 1|$ is constant, not negligible, and so this G is not a pseudorandom generator. (Note that D is not always "correct," since it sometimes outputs 1 even when given a uniform string. This does not change the fact that D is a good distinguisher.) $\diamondsuit$

Discussion. The distribution on the output of a pseudorandom generator G is far from uniform. To see this, consider the case that $\ell(n) = 2n$ and so G doubles the length of its input. Under the uniform distribution on $\{0,1\}^{2n}$, each of the 2^{2n} possible strings is chosen with probability exactly 2^{-2n}. In contrast, consider the distribution of the output of G (when G is run on a uniform seed). When G receives an input of length n, the number of different strings in the range of G is at most 2^n. The fraction of strings of length $2n$ that are in the range of G is thus at most $2^n/2^{2n} = 2^{-n}$, and we see that the vast majority of strings of length $2n$ do not occur as outputs of G.

This in particular means that it is trivial to distinguish between a random string and a pseudorandom string *given an unlimited amount of time.* Let G be as above and consider the exponential-time distinguisher D that works as follows: $D(w)$ outputs 1 if and only if there exists an $s \in \{0,1\}^n$ such that $G(s) = w$. (This computation is carried out in exponential time by exhaustively computing $G(s)$ for every $s \in \{0,1\}^n$. Recall that by Kerckhoffs' principle, the specification of G is known to D.) Now, if w were output by G, then D outputs 1 with probability 1. In contrast, if w is uniformly distributed in $\{0,1\}^{2n}$, then the probability that there exists an s with $G(s) = w$ is at most 2^{-n}, and so D outputs 1 in this case with probability at most 2^{-n}. So

$$\big|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]\big| \geq 1 - 2^{-n},$$

which is large. This is just another example of a *brute-force attack*, and does not contradict the pseudorandomness of G since the attack is not efficient.

The seed and its length. The seed for a pseudorandom generator is analogous to the cryptographic key used by an encryption scheme, and the seed must be chosen uniformly and be kept secret from any adversary. Another important point, evident from the above discussion of brute-force attacks, is that s must be long enough so that it is not feasible to enumerate all possible seeds. In an asymptotic sense this is taken care of by setting the length of the seed equal to the security parameter, so that exhaustive search over all possible seeds requires exponential time. In practice, the seed must be long enough so that it is impossible to try all possible seeds within some specified time bound.

On the existence of pseudorandom generators. Do pseudorandom generators exist? They certainly seem difficult to construct, and one may rightly ask whether any algorithm satisfying Definition 3.14 exists. Although we do not know how to unconditionally prove the existence of pseudorandom generators, we have strong reasons to believe they exist. For one, they can be constructed under the rather weak assumption that *one-way functions* exist (which is true if certain problems like factoring large numbers are hard); this will be discussed in detail in Chapter 7. We also have several practical constructions of candidate pseudorandom generators called *stream ciphers* for which no efficient distinguishers are known. (Later, we will introduce even stronger primitives called *block ciphers*.) We give a high-level overview of stream ciphers next, and discuss concrete stream ciphers in Chapter 6.

Stream Ciphers

Our definition of a pseudorandom generator is limited in two ways: the expansion factor is fixed, and the generator produces its entire output in "one shot." *Stream ciphers*, used in practice to instantiate pseudorandom generators, work somewhat differently. The pseudorandom output bits of a stream cipher are produced gradually and on demand, so that an application can request exactly as many pseudorandom bits as needed. This improves efficiency (since an application can request fewer bits, if sufficient) and flexibility (since there is no upper bound on the number of bits that can be requested).

Formally, we view a stream cipher[2] as a pair of deterministic algorithms $(\mathsf{Init}, \mathsf{GetBits})$ where:

- Init takes as input a seed s and an optional *initialization vector* IV, and outputs an initial state st_0.

- $\mathsf{GetBits}$ takes as input state information st_i, and outputs a bit y and updated state st_{i+1}. (In practice, y is a *block* of several bits; we treat y as a single bit here for generality and simplicity.)

Given a stream cipher and any desired expansion factor ℓ, we can define an algorithm G_ℓ mapping inputs of length n to outputs of length $\ell(n)$. The algorithm simply runs Init, and then repeatedly runs $\mathsf{GetBits}$ a total of ℓ times.

ALGORITHM 3.16
Constructing G_ℓ **from** $(\mathsf{Init}, \mathsf{GetBits})$

Input: Seed s and optional initialization vector IV
Output: $y_1, \ldots, y_\ell$

```
$\mathsf{st}_0 := \mathsf{Init}(s, IV)$
for $i = 1$ to $\ell$:
    $(y_i, \mathsf{st}_i) := \mathsf{GetBits}(\mathsf{st}_{i-1})$
return $y_1, \ldots, y_\ell$
```

[2]The terminology here is not completely standard, and beware that "stream cipher" is used by different people in different (but related) ways. For example, some use it to refer to G_ℓ (see below), while some use it to refer to Construction 3.17 when instantiated with G_ℓ.

A stream cipher is *secure* in the basic sense if it takes no IV and for any polynomial ℓ with $\ell(n) > n$, the function G_ℓ constructed above is a pseudorandom generator with expansion factor ℓ. We briefly discuss one possible security notion for stream ciphers that use an IV in Section 3.6.1.

3.3.2 Proofs by Reduction

If we wish to prove that a given construction is computationally secure, then we must rely on unproven assumptions[3] (unless the scheme is information-theoretically secure). Our strategy will be to assume that some mathematical problem is hard, or that some *low-level* cryptographic primitive is secure, and then to *prove* that a given construction based on this problem/primitive is secure under this assumption. In Section 1.4.2 we have already explained in great detail why this approach is preferable so we do not repeat those arguments here.

The proof that a cryptographic construction is secure as long as some underlying problem is hard generally proceeds by presenting an explicit *reduction* showing how to transform any efficient adversary $\mathcal{A}$ that succeeds in "breaking" the construction into an efficient algorithm $\mathcal{A}'$ that solves the problem that was assumed to be hard. Since this is so important, we walk through a high-level outline of the steps of such a proof in detail. (We will see numerous concrete examples through the book, beginning with the proof of Theorem 3.18.) We begin with an assumption that some problem X cannot be solved (in some precisely defined sense) by any polynomial-time algorithm, except with negligible probability. We want to prove that some cryptographic construction Π is secure (again, in some sense that is precisely defined). A proof proceeds via the following steps (see also Figure 3.1):

1. Fix some efficient (i.e., probabilistic polynomial-time) adversary $\mathcal{A}$ attacking Π. Denote this adversary's success probability by $\varepsilon(n)$.
2. Construct an efficient algorithm $\mathcal{A}'$, called the "reduction," that attempts to solve problem X using adversary $\mathcal{A}$ as a subroutine. An important point here is that $\mathcal{A}'$ knows nothing about how $\mathcal{A}$ works; the only thing $\mathcal{A}'$ knows is that $\mathcal{A}$ is expecting to attack Π. So, given some input instance x of problem X, our algorithm $\mathcal{A}'$ will *simulate* for $\mathcal{A}$ an instance of Π such that:
   (a) As far as $\mathcal{A}$ can tell, it is interacting with Π. That is, the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ should be distributed identically to (or at least close to) the view of $\mathcal{A}$ when it interacts with Π itself.
   (b) If $\mathcal{A}$ succeeds in "breaking" the instance of Π that is being simulated by $\mathcal{A}'$, this should allow $\mathcal{A}'$ to solve the instance x it was given, at least with inverse polynomial probability $1/p(n)$.

[3]In particular, most of cryptography requires the unproven assumption that $\mathcal{P} \neq \mathcal{NP}$.

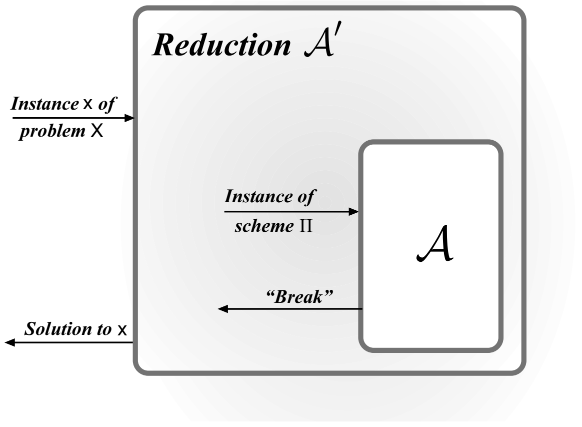

FIGURE 3.1: A high-level overview of a security proof by reduction.

3. Taken together, 2(a) and 2(b) imply that $\mathcal{A}'$ solves X with probability $\varepsilon(n)/p(n)$. If $\varepsilon(n)$ is not negligible, then neither is $\varepsilon(n)/p(n)$. Moreover, if $\mathcal{A}$ is efficient then we obtain an efficient algorithm $\mathcal{A}'$ solving X with non-negligible probability, contradicting the initial assumption.

4. Given our assumption regarding X, we conclude that *no* efficient adversary $\mathcal{A}$ can succeed in breaking Π with non-negligible probability. Stated differently, Π is computationally secure.

In the following section we will illustrate exactly the above idea: we will show how to use any pseudorandom generator G to construct an encryption scheme; we prove the encryption scheme secure by showing that any attacker who can "break" the encryption scheme can be used to distinguish the output of G from a uniform string. Under the assumption that G is a pseudorandom generator, then, the encryption scheme is secure.

3.3.3 A Secure Fixed-Length Encryption Scheme

A pseudorandom generator provides a natural way to construct a secure, fixed-length encryption scheme with a key shorter than the message. Recall that in the one-time pad (see Section 2.2), encryption is done by XORing a random pad with the message. The insight is that we can use a *pseudorandom* pad instead. Rather than sharing this long, pseudorandom pad, however, the sender and receiver can instead share a *seed* which is used to generate that pad when needed (see Figure 3.2); this seed will be shorter than the pad and hence shorter than the message. As for security, the intuition is that a pseudorandom string "looks random" to any polynomial-time adversary and so a computationally bounded eavesdropper cannot distinguish between a message encrypted using the one-time pad or a message encrypted using this "pseudo-"one-time pad encryption scheme.

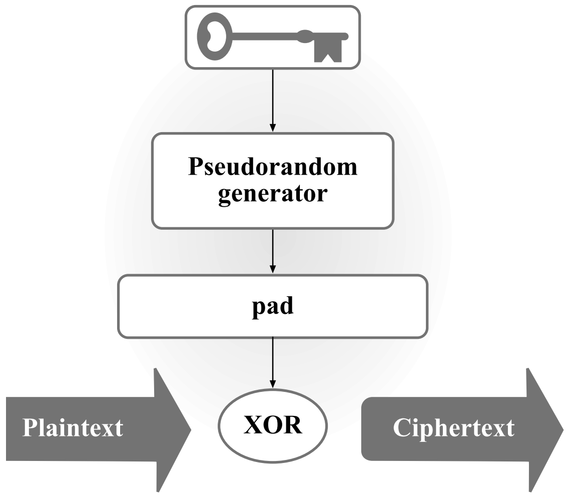

FIGURE 3.2: Encryption with a pseudorandom generator.

The encryption scheme. Fix some message length ℓ and let G be a pseudorandom generator with expansion factor ℓ (that is, $|G(s)| = \ell(|s|)$). Recall that an encryption scheme is defined by three algorithms: a key-generation algorithm Gen, an encryption algorithm Enc, and a decryption algorithm Dec. The key-generation algorithm is the trivial one: $\mathsf{Gen}(1^n)$ simply outputs a uniform key k of length n. Encryption works by applying G to the key (which serves as a seed) in order to obtain a pad that is then XORed with the plaintext. Decryption applies G to the key and XORs the resulting pad with the ciphertext to recover the message. The scheme is described formally in Construction 3.17. In Section 3.6.1, we describe how stream ciphers are used to implement a variant of this scheme in practice.

CONSTRUCTION 3.17

Let G be a pseudorandom generator with expansion factor ℓ. Define a private-key encryption scheme for messages of length ℓ as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext
$$c := G(k) \oplus m.$$
- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^{\ell(n)}$, output the message
$$m := G(k) \oplus c.$$

A private-key encryption scheme based on any pseudorandom generator.

THEOREM 3.18 *If G is a pseudorandom generator, then Construction 3.17 is a fixed-length private-key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper.*

PROOF Let Π denote Construction 3.17. We show that Π satisfies Definition 3.8. Namely, we show that for any probabilistic polynomial-time adversary $\mathcal{A}$ there is a negligible function negl such that

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n). \tag{3.2}$$

The intuition is that if Π used a uniform pad in place of the pseudorandom pad $G(k)$, then the resulting scheme would be identical to the one-time pad encryption scheme and $\mathcal{A}$ would be unable to correctly guess which message was encrypted with probability any better than 1/2. Thus, if Equation (3.2) does *not* hold then $\mathcal{A}$ must implicitly be distinguishing the output of G from a random string. We make this explicit by showing a *reduction*; namely, by showing how to use $\mathcal{A}$ to construct an efficient distinguisher D, with the property that D's ability to distinguish the output of G from a uniform string is directly related to $\mathcal{A}$'s ability to determine which message was encrypted by Π. Security of G then implies security of Π.

Let $\mathcal{A}$ be an arbitrary PPT adversary. We construct a distinguisher D that takes a string w as input, and whose goal is to determine whether w was chosen uniformly (i.e., w is a "random string") or whether w was generated by choosing a uniform k and computing $w := G(k)$ (i.e., w is a "pseudorandom string"). We construct D so that it emulates the eavesdropping experiment for $\mathcal{A}$, as described below, and observes whether $\mathcal{A}$ succeeds or not. If $\mathcal{A}$ succeeds then D guesses that w must be a pseudorandom string, while if $\mathcal{A}$ does not succeed then D guesses that w is a random string. In detail:

Distinguisher D:
D is given as input a string $w \in \{0,1\}^{\ell(n)}$. (We assume that n can be determined from $\ell(n)$.)

1. Run $\mathcal{A}(1^n)$ to obtain a pair of messages $m_0, m_1 \in \{0,1\}^{\ell(n)}$.
2. Choose a uniform bit $b \in \{0,1\}$. Set $c := w \oplus m_b$.
3. Give c to $\mathcal{A}$ and obtain output b'. Output 1 if $b' = b$, and output 0 otherwise.

D clearly runs in polynomial time (assuming $\mathcal{A}$ does).

Before analyzing the behavior of D, we define a modified encryption scheme $\widetilde{\Pi} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ that is exactly the one-time pad encryption scheme, except that we now incorporate a security parameter that determines the length of the message to be encrypted. That is, $\widetilde{\mathsf{Gen}}(1^n)$ outputs a uniform key k of length $\ell(n)$, and the encryption of message $m \in 2^{\ell(n)}$ using key $k \in \{0,1\}^{\ell(n)}$

is the ciphertext $c := k \oplus m$. (Decryption can be performed as usual, but is inessential to what follows.) Perfect secrecy of the one-time pad implies

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] = \frac{1}{2}. \tag{3.3}$$

To analyze the behavior of D, the main observations are:

1. If w is chosen uniformly from $\{0,1\}^{\ell(n)}$, then the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)$. This is because when $\mathcal{A}$ is run as a subroutine by $D(w)$ in this case, $\mathcal{A}$ is given a ciphertext $c = w \oplus m_b$ where $w \in \{0,1\}^{\ell(n)}$ is uniform. Since D outputs 1 exactly when $\mathcal{A}$ succeeds in its eavesdropping experiment, we therefore have (cf. Equation (3.3))

$$\Pr_{w\leftarrow\{0,1\}^{\ell(n)}}[D(w) = 1] = \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] = \frac{1}{2}. \tag{3.4}$$

   (The subscript on the first probability just makes explicit that w is chosen uniformly from $\{0,1\}^{\ell(n)}$ there.)

2. If w is instead generated by choosing uniform $k \in \{0,1\}^n$ and then setting $w := G(k)$, the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$. This is because $\mathcal{A}$, when run as a subroutine by D, is now given a ciphertext $c = w \oplus m_b$ where $w = G(k)$ for a uniform $k \in \{0,1\}^n$. Thus,

$$\Pr_{k\leftarrow\{0,1\}^n}[D(G(k)) = 1] = \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right]. \tag{3.5}$$

Since G is a pseudorandom generator (and since D runs in polynomial time), we know there is a negligible function negl such that

$$\left|\Pr_{w\leftarrow\{0,1\}^{\ell(n)}}[D(w) = 1] - \Pr_{k\leftarrow\{0,1\}^n}[D(G(k)) = 1]\right| \le \mathsf{negl}(n).$$

Using Equations (3.4) and (3.5), we thus see that

$$\left|\frac{1}{2} - \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right]\right| \le \mathsf{negl}(n),$$

which implies $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \le \frac{1}{2} + \mathsf{negl}(n)$. Since $\mathcal{A}$ was an arbitrary PPT adversary, this completes the proof that Π has indistinguishable encryptions in the presence of an eavesdropper. ■

It is easy to get lost in the details of the proof and wonder whether anything has been gained as compared to the one-time pad; after all, the one-time pad also encrypts an ℓ-bit message by XORing it with an ℓ-bit string! The point of the construction, of course, is that the ℓ-bit string $G(k)$ can be *much* *longer* than the shared key k. In particular, using the above scheme it is possible to securely encrypt a 1 Mb file using only a 128-bit key. By relying on computational secrecy we have thus circumvented the impossibility result of Theorem 2.10, which states that any *perfectly* secret encryption scheme must use a key at least as long as the message.

Reductions—a discussion. We do *not* prove unconditionally that Construction 3.17 is secure. Rather, we prove that it is secure *under the assumption* that G is a pseudorandom generator. This approach of *reducing* the security of a higher-level construction to a lower-level primitive has a number of advantages (as discussed in Section 1.4.2). One of these advantages is that, in general, it is easier to design a lower-level primitive than a higher-level one; it is also easier, in general, to directly analyze an algorithm G with respect to a lower-level definition than to analyze a more complex scheme Π with respect to a higher-level definition. This does not mean that constructing a pseudorandom generator is "easy," only that it is easier than constructing an encryption scheme from scratch. (In the present case the encryption scheme does nothing except XOR the output of a pseudorandom generator with the message and so this isn't really true. However, we will see more complex constructions and in those cases the ability to reduce the task to a simpler one is of great importance.) Another advantage is that once an appropriate G has been constructed, it can be used as a component of various other schemes.

Concrete security. Although Theorem 3.18 and its proof are in an asymptotic setting, we can readily adapt the proof to bound the *concrete* security of the encryption scheme in terms of the concrete security of G. Fix some value of n for the remainder of this discussion, and let Π now denote Construction 3.17 using this value of n. Assume G is (t, ε)-pseudorandom (for the given value of n), in the sense that for all distinguishers D running in time at most t we have

$$\big|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]\big| \leq \varepsilon. \tag{3.6}$$

(Think of $t \approx 2^{80}$ and $\varepsilon \approx 2^{-60}$, though precise values are irrelevant for our discussion.) We claim that Π is $(t - c,\, \varepsilon)$-secure for some (small) constant c, in the sense that for all $\mathcal{A}$ running in time at most $t - c$ we have

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1\right] \leq \frac{1}{2} + \varepsilon. \tag{3.7}$$

(Note that the above are now fixed numbers, not functions of n, since we are not in an asymptotic setting here.) To see this, let $\mathcal{A}$ be an arbitrary adversary running in time at most $t - c$. Distinguisher D, as constructed in the proof of Theorem 3.18, has very little overhead besides running $\mathcal{A}$; setting c appropriately ensures that D runs in time at most t. Our assumption on the concrete security of G then implies Equation (3.6); proceeding exactly as in the proof of Theorem 3.18, we obtain Equation (3.7).

3.4 Stronger Security Notions

Until now we have considered a relatively weak definition of security in which the adversary only passively eavesdrops on a single ciphertext sent between the honest parties. In this section, we consider two stronger security notions. Recall that a security definition specifies a security goal and an attack model. In defining the first new security notion, we modify the security goal; for the second we strengthen the attack model.

3.4.1 Security for Multiple Encryptions

Definition 3.8 deals with the case where the communicating parties transmit a single ciphertext that is observed by an eavesdropper. It would be convenient, however, if the communicating parties could send multiple ciphertexts to each other—all generated using the same key—even if an eavesdropper might observe all of them. For such applications we need an encryption scheme secure for the encryption of multiple messages.

We begin with an appropriate definition of security for this setting. As in the case of Definition 3.8, we first introduce an appropriate experiment defined for any encryption scheme Π, adversary $\mathcal{A}$, and security parameter n:

The multiple-message eavesdropping experiment $\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(n)$:

1. *The adversary $\mathcal{A}$ is given input 1^n, and outputs a pair of equal-length* lists *of messages $\vec{M}_0 = (m_{0,1}, \ldots, m_{0,t})$ and $\vec{M}_1 = (m_{1,1}, \ldots, m_{1,t})$, with $|m_{0,i}| = |m_{1,i}|$ for all i.*
2. *A key k is generated by running $\mathsf{Gen}(1^n)$, and a uniform bit $b \in \{0,1\}$ is chosen. For all i, the ciphertext $c_i \leftarrow \mathsf{Enc}_k(m_{b,i})$ is computed and the list $\vec{C} = (c_1, \ldots, c_t)$ is given to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a bit b'.*
4. *The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.*

The definition of security is the same as before, except that it now refers to the above experiment.

DEFINITION 3.19 *A private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has* indistinguishable multiple encryptions in the presence of an eavesdropper *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function* negl *such that*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the randomness used by $\mathcal{A}$ and the randomness used in the experiment.

Any scheme that has indistinguishable multiple encryptions in the presence of an eavesdropper clearly also satisfies Definition 3.8, since experiment $\mathsf{PrivK}^{\mathsf{eav}}$ corresponds to the special case of $\mathsf{PrivK}^{\mathsf{mult}}$ where the adversary outputs two lists containing only a single message each. In fact, our new definition is strictly stronger than Definition 3.8, as the following shows.

PROPOSITION 3.20 *There is a private-key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper, but not indistinguishable* multiple *encryptions in the presence of an eavesdropper.*

PROOF We do not have to look far to find an example of an encryption scheme satisfying the proposition. The one-time pad is perfectly secret, and so also has indistinguishable encryptions in the presence of an eavesdropper. We show that it is not secure in the sense of Definition 3.19. (We have discussed this attack in Chapter 2 already; here, we merely analyze the attack with respect to Definition 3.19.)

Concretely, consider the following adversary $\mathcal{A}$ attacking the scheme (in the sense defined by experiment $\mathsf{PrivK}^{\mathsf{mult}}$): $\mathcal{A}$ outputs $\vec{M}_0 = (0^\ell, 0^\ell)$ and $\vec{M}_1 = (0^\ell, 1^\ell)$. (The first contains the same plaintext twice, while the second contains two different messages.) Let $\vec{C} = (c_1, c_2)$ be the list of ciphertexts that $\mathcal{A}$ receives. If $c_1 = c_2$, then $\mathcal{A}$ outputs $b' = 0$; otherwise, $\mathcal{A}$ outputs $b' = 1$.

We now analyze the probability that $b' = b$. The crucial point is that the one-time pad is *deterministic*, so encrypting the same message twice (using the same key) yields the same ciphertext. Thus, if $b = 0$ then we must have $c_1 = c_2$ and $\mathcal{A}$ outputs 0 in this case. On the other hand, if $b = 1$ then a different message is encrypted each time; hence $c_1 \neq c_2$ and $\mathcal{A}$ outputs 1. We conclude that $\mathcal{A}$ correctly outputs $b' = b$ with probability 1, and so the encryption scheme is not secure with respect to Definition 3.19. ■

Necessity of probabilistic encryption. The above might appear to show that Definition 3.19 is impossible to achieve using *any* encryption scheme. But in fact this is true only if the encryption scheme is *deterministic* and so encrypting the same message multiple times (using the same key) always yields the same result. This is important enough to state as a theorem.

THEOREM 3.21 *If* Π *is a (stateless*[4]*) encryption scheme in which* Enc *is a deterministic function of the key and the message, then* Π *cannot have indistinguishable multiple encryptions in the presence of an eavesdropper.*

This should not be taken to mean that Definition 3.19 is too strong. Indeed,

[4]We will see in Section 3.6.1 that if the encryption scheme is stateful, then it is possible to securely encrypt multiple messages even if encryption is deterministic.

leaking to an eavesdropper the fact that two encrypted messages are the same can be a significant security breach. (Consider, e.g., a scenario in which a student encrypts a series of true/false answers!)

To construct a scheme secure for encrypting multiple messages, we must design a scheme in which encryption is *randomized* so that when the same message is encrypted multiple times, different ciphertexts can be produced. This may seem impossible since decryption must always be able to recover the message. However, we will soon see how to achieve it.

3.4.2 Chosen-Plaintext Attacks and CPA-Security

Chosen-plaintext attacks capture the ability of an adversary to exercise (partial) control over what the honest parties encrypt. We imagine a scenario in which two honest parties share a key k, and the attacker can influence these parties to encrypt messages $m_1, m_2, \ldots$ (using k) and send the resulting ciphertexts over a channel that the attacker can observe. At some later point in time, the attacker observes a ciphertext corresponding to some *unknown* message m encrypted using the same key k; let us even assume that the attacker knows that m is one of two possibilities m_0, m_1. Security against chosen-plaintext attacks means that even in this case the attacker cannot tell which of these two messages was encrypted with probability significantly better than random guessing. (For now we revert back to the case where the eavesdropper is given only a single encryption of an unknown message. Shortly, we will return to consideration of the multiple-message case.)

Chosen-plaintext attacks in the real world. Are chosen-plaintext attacks a realistic concern? For starters, note that chosen-plaintext attacks also encompass *known-plaintext attacks*—in which the attacker knows what messages are being encrypted, even if it does not get to choose them—as a special case. Moreover, there are several real-world scenarios in which an adversary might have significant influence over what messages get encrypted. A simple example is given by an attacker typing on a terminal, which in turn encrypts and sends everything the adversary types using a key shared with a remote server (and unknown to the attacker). Here the attacker exactly controls what gets encrypted, but the encryption scheme should remain secure when it is used—with the same key— to encrypt data for another user.

Interestingly, chosen-plaintext attacks have also been used successfully as part of historical efforts to break military encryption schemes. For example, during World War II the British placed mines at certain locations, knowing that the Germans—when finding those mines—would encrypt the locations and send them back to headquarters. These encrypted messages were used by cryptanalysts at Bletchley Park to break the German encryption scheme.

Another example is given by the famous story involving the Battle of Midway. In May 1942, US Navy cryptanalysts intercepted an encrypted message from the Japanese which they were able to partially decode. The result indicated that the Japanese were planning an attack on `AF`, where `AF` was a ciphertext fragment that the US was unable to decode. For other reasons, the US believed that Midway Island was the target. Unfortunately, their attempts to convince Washington planners that this was the case were futile; the general belief was that Midway could not possibly be the target. The Navy cryptanalysts devised the following plan: They instructed US forces at Midway to send a fake message that their freshwater supplies were low. The Japanese intercepted this message and immediately reported to their superiors that "`AF` is low on water." The Navy cryptanalysts now had their proof that `AF` corresponded to Midway, and the US dispatched three aircraft carriers to that location. The result was that Midway was saved, and the Japanese incurred significant losses. This battle was a turning point in the war between the US and Japan in the Pacific.

The Navy cryptanalysts here carried out a chosen-plaintext attack, as they were able to influence the Japanese (albeit in a roundabout way) to encrypt the word "Midway." If the Japanese encryption scheme had been secure against chosen-plaintext attacks, this strategy by the US cryptanalysts would not have worked (and history may have turned out very differently)!

CPA-security. In the formal definition we model chosen-plaintext attacks by giving the adversary $\mathcal{A}$ access to an *encryption oracle*, viewed as a "black box" that encrypts messages of $\mathcal{A}$'s choice using a key k that is unknown to $\mathcal{A}$. That is, we imagine $\mathcal{A}$ has access to an "oracle" $\mathsf{Enc}_k(\cdot)$; when $\mathcal{A}$ *queries* this oracle by providing it with a message m as input, the oracle returns a ciphertext $c \leftarrow \mathsf{Enc}_k(m)$ as the reply. (When Enc is randomized, the oracle uses fresh randomness each time it answers a query.) The adversary is allowed to interact with the encryption oracle adaptively, as many times as it likes.

Consider the following experiment defined for any encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, adversary $\mathcal{A}$, and value n for the security parameter:

The CPA indistinguishability experiment $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$:

1. *A key k is generated by running $\mathsf{Gen}(1^n)$.*
2. *The adversary $\mathcal{A}$ is given input 1^n and oracle access to $\mathsf{Enc}_k(\cdot)$, and outputs a pair of messages m_0, m_1 of the same length.*
3. *A uniform bit $b \in \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$.*
4. *The adversary $\mathcal{A}$ continues to have oracle access to $\mathsf{Enc}_k(\cdot)$, and outputs a bit b'.*
5. *The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. In the former case, we say that $\mathcal{A}$ succeeds.*

DEFINITION 3.22 *A private-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions under a chosen-plaintext attack*, or is* CPA-secure*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the randomness used by $\mathcal{A}$*, as well as the randomness used in the experiment.*

CPA-Security for Multiple Encryptions

Definition 3.22 can be extended to the case of multiple encryptions in the same way that Definition 3.8 is extended to give Definition 3.19, i.e., by using lists of plaintexts. Here, we take a different approach that is somewhat simpler and has the advantage of modeling attackers that can *adaptively* choose plaintexts to be encrypted, even after observing previous ciphertexts. In the present definition, we give the attacker access to a "left-or-right" oracle $\mathsf{LR}_{k,b}$ that, on input a pair of equal-length messages m_0, m_1, computes the ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ and returns c. That is, if $b = 0$ then the adversary receives an encryption of the "left" plaintext, and if $b = 1$ then it receives an encryption of the "right" plaintext. Here, b is a random bit chosen at the beginning of the experiment, and as in previous definitions the goal of the attacker is to guess b. This generalizes the previous definition of multiple-message security (Definition 3.19) because instead of outputting the lists $(m_{0,1}, \ldots, m_{0,t})$ and $(m_{1,1}, \ldots, m_{1,t})$, one of whose messages will be encrypted, the attacker can now sequentially query $\mathsf{LR}_{k,b}(m_{0,1}, m_{1,1}), \ldots, \mathsf{LR}_{k,b}(m_{0,t}, m_{1,t})$. This also encompasses the attacker's access to an encryption oracle, since the attacker can simply query $\mathsf{LR}_{k,b}(m, m)$ to obtain $\mathsf{Enc}_k(m)$.

We now formally define this experiment, called the LR-oracle experiment. Let Π be an encryption scheme, $\mathcal{A}$ an adversary, and n the security parameter:

The LR-oracle experiment $\mathsf{PrivK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. *A key k is generated by running* $\mathsf{Gen}(1^n)$*.*
2. *A uniform bit $b \in \{0, 1\}$ is chosen.*
3. *The adversary $\mathcal{A}$ is given input 1^n and oracle access to* $\mathsf{LR}_{k,b}(\cdot, \cdot)$*, as defined above.*
4. *The adversary $\mathcal{A}$ outputs a bit b'.*
5. *The output of the experiment is defined to be* 1 *if $b' = b$, and* 0 *otherwise. In the former case, we say that $\mathcal{A}$* succeeds.

DEFINITION 3.23 *Private-key encryption scheme* Π *has* indistinguishable multiple encryptions under a chosen-plaintext attack*, or is* CPA-secure for multiple encryptions*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the randomness used by $\mathcal{A}$ *and the randomness used in the experiment.*

Our earlier discussion shows that CPA-security for multiple encryptions is at least as strong as all our previous definitions. In particular, if a private-key encryption scheme is CPA-secure for multiple encryptions then it is clearly CPA-secure as well. Importantly, the converse also holds; that is, CPA-security implies CPA-security for multiple encryptions. (This stands in contrast to the case of eavesdropping adversaries; see Proposition 3.20.) We state the following theorem here without proof; a similar result in the public-key setting is proved in Section 11.2.2.

THEOREM 3.24 *Any private-key encryption scheme that is CPA-secure is also CPA-secure for* multiple *encryptions.*

This is a significant technical advantage of CPA-security: It suffices to prove that a scheme is CPA-secure (for a single encryption), and we then obtain "for free" that it is CPA-secure for multiple encryptions as well.

Security against chosen-plaintext attacks is nowadays the minimal notion of security an encryption scheme should satisfy, though it is becoming more common to require even the stronger security properties discussed in Section 4.5.

Fixed-length vs. arbitrary-length messages. Another advantage of working with the definition of CPA-security is that it allows us to treat fixed-length encryption schemes without loss of generality. In particular, given any CPA-secure *fixed-length* encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, it is possible to construct a CPA-secure encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ for *arbitrary-length* messages quite easily. For simplicity, say Π encrypts messages that are 1-bit long (though everything we say extends in the natural way regardless of the message length supported by Π). Leave Gen' the same as Gen. Define Enc'_k for any message m (having some arbitrary length ℓ) as $\mathsf{Enc}'_k(m) = \mathsf{Enc}_k(m_1), \ldots, \mathsf{Enc}_k(m_\ell)$, where m_i denotes the ith bit of m. Decryption is done in the natural way. Π' is CPA-secure if Π is; a proof follows from Theorem 3.24.

There are more efficient ways to encrypt messages of arbitrary length than by adapting a fixed-length encryption scheme in the above manner. We explore this further in Section 3.6.