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. $\diamondsuit$

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 re-define 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*

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

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

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

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:*

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

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 $G : \{0,1\}^n \rightarrow \{0,1\}^\ell$ 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.) ♢

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 $\mathcal{P} \neq \mathcal{NP}$.

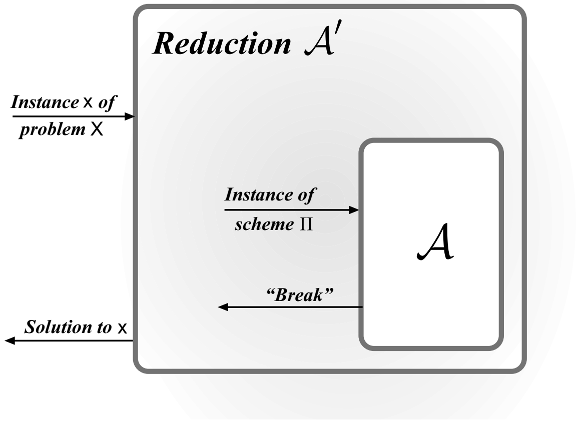

FIGURE 3.1: A high-level overview of a security proof by reduction.

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.

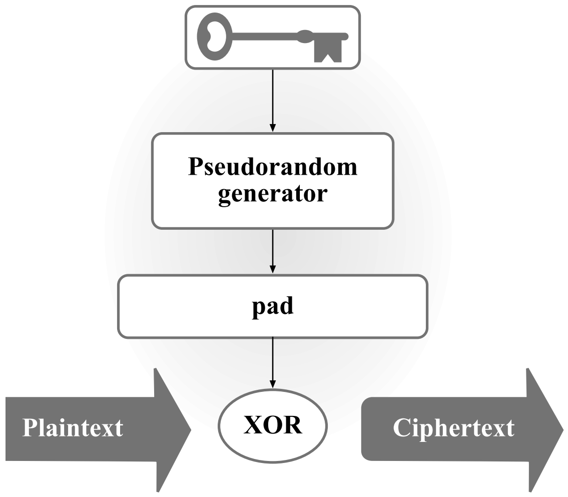

FIGURE 3.2: Encryption with a pseudorandom generator.

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

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

(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.

3.5 Constructing CPA-Secure Encryption Schemes

Before constructing encryption schemes secure against chosen-plaintext attacks, we first introduce the important notion of *pseudorandom functions*.

3.5.1 Pseudorandom Functions and Block Ciphers

Pseudorandom functions (PRFs) generalize the notion of pseudorandom generators. Now, instead of considering "random-looking" *strings* we consider "random-looking" *functions*. As in our earlier discussion of pseudorandomness, it does not make much sense to say that any *fixed* function $f : \{0,1\}^* \to \{0,1\}^*$ is pseudorandom (in the same way that it makes little sense to say that any fixed function is random). Thus, we must instead refer to the pseudorandomness of a *distribution* on functions. Such a distribution is induced naturally by considering *keyed functions*, defined next.

A keyed function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is a two-input function, where the first input is called the *key* and denoted k. We say F is *efficient* if there is a polynomial-time algorithm that computes $F(k,x)$ given k and x. (We will only be interested in efficient keyed functions.) In typical usage a key k is chosen and fixed, and we are then interested in the single-input function $F_k : \{0,1\}^* \to \{0,1\}^*$ defined by $F_k(x) = F(k,x)$. The security parameter n dictates the key length, input length, and output length. That is, we associate with F three functions ℓ_{key}, ℓ_{in}, and ℓ_{out}; for any key $k \in \{0,1\}^{\ell_{key}(n)}$, the function F_k is only defined for inputs $x \in \{0,1\}^{\ell_{in}(n)}$, in which case $F_k(x) \in \{0,1\}^{\ell_{out}(n)}$. Unless stated otherwise, we assume for simplicity that F is *length-preserving*, meaning $\ell_{key}(n) = \ell_{in}(n) = \ell_{out}(n) = n$. That is, by fixing a key $k \in \{0,1\}^n$ we obtain a function F_k mapping n-bit input strings to n-bit output strings.

A keyed function F induces a natural distribution on functions given by choosing a uniform key $k \in \{0,1\}^n$ and then considering the resulting single-input function F_k. We call F *pseudorandom* if the function F_k (for a uniform key k) is indistinguishable from a function chosen uniformly at random from the set of all functions having the same domain and range; that is, if no efficient adversary can distinguish—in a sense we more carefully define below—whether it is interacting with F_k (for uniform k) or f (where f is chosen uniformly from the set of all functions mapping n-bit inputs to n-bit outputs).

Since choosing a function at random is less intuitive than choosing a string at random, it is worth spending a bit more time on this idea. Consider the set Func_n of all functions mapping n-bit strings to n-bit strings. This set is finite, and selecting a uniform function mapping n-bit strings to n-bit strings means choosing an element uniformly from this set. How large is Func_n? A function f is specified by giving its value on each point in its domain. We can view any function (over a finite domain) as a large look-up table that stores

$f(x)$ in the row of the table labeled by x. For $f \in \mathsf{Func}_n$, the look-up table for f has 2^n rows (one for each point of the domain $\{0,1\}^n$), with each row containing an n-bit string (since the range of f is $\{0,1\}^n$). Concatenating all the entries of the table, we see that any function in Func_n can be represented by a string of length $2^n \cdot n$. Moreover, this correspondence is one-to-one, as each string of length $2^n \cdot n$ (i.e., each table containing 2^n entries of length n) defines a unique function in Func_n. Thus, the size of Func_n is exactly the number of strings of length $n \cdot 2^n$, or $|\mathsf{Func}_n| = 2^{n \cdot 2^n}$.

Viewing a function as a look-up table provides another useful way to think about selecting a uniform function $f \in \mathsf{Func}_n$: It is exactly equivalent to choosing each row in the look-up table of f uniformly. This means, in particular, that the values $f(x)$ and $f(y)$ (for any two inputs $x \neq y$) are uniform and independent. We can view this look-up table being populated by random entries in advance, before f is evaluated on any input, or we can view entries of the table being chosen uniformly "on-the-fly," as needed, whenever f is evaluated on a new input on which f has not been evaluated before.

Coming back to our discussion of pseudorandom functions, recall that a pseudorandom function is a keyed function F such that F_k (for $k \in \{0,1\}^n$ chosen uniformly at random) is indistinguishable from f (for $f \in \mathsf{Func}_n$ chosen uniformly at random). The former is chosen from a distribution over (at most) 2^n distinct functions, whereas the latter is chosen from all $2^{n \cdot 2^n}$ functions in Func_n. Despite this, the "behavior" of these functions must look the same to any polynomial-time distinguisher.

A first attempt at formalizing the notion of a pseudorandom function would be to proceed in the same way as in Definition 3.14. That is, we could require that every polynomial-time distinguisher D that receives a description of the pseudorandom function F_k outputs 1 with "almost" the same probability as when it receives a description of a random function f. However, this definition is inappropriate since the description of a random function has *exponential length* (given by its look-up table of length $n \cdot 2^n$), while D is limited to running in polynomial time. So, D would not even have sufficient time to examine its entire input.

The definition therefore gives D access to an *oracle* $\mathcal{O}$ which is either equal to F_k (for uniform k) or f (for a uniform function f). The distinguisher D may query its oracle at any point x, in response to which the oracle returns $\mathcal{O}(x)$. We treat the oracle as a black box in the same way as when we provided the adversary with oracle access to the encryption algorithm in the definition of a chosen-plaintext attack. Here, however, the oracle computes a deterministic function and so returns the same result if queried twice on the same input. (For this reason, we may assume without loss of generality that D never queries the oracle twice on the same input.) D may interact freely with its oracle, choosing its queries adaptively based on all previous outputs. Since D runs in polynomial time, however, it can ask only polynomially many queries.

We now present the formal definition. (The definition assumes F is length-preserving for simplicity only.)

DEFINITION 3.25 *Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, length-preserving, keyed function. F is a* pseudorandom function *if for all probabilistic polynomial-time distinguishers D, there is a negligible function* negl *such that:*

$$\left| \Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \mathsf{negl}(n),$$

where the first probability is taken over uniform choice of $k \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $f \in \mathsf{Func}_n$ and the randomness of D.

An important point is that D is *not* given the key k. It is meaningless to require that F_k be pseudorandom if k is known, since given k it is trivial to distinguish an oracle for F_k from an oracle for f. (All the distinguisher has to do is query the oracle at any point x to obtain the answer y, and compare this to the result $y' := F_k(x)$ that it computes itself using the known value k. An oracle for F_k will return $y = y'$, while an oracle for a random function will have $y = y'$ with probability only 2^{-n}.) This means that if k is revealed, any claims about the pseudorandomness of F_k no longer hold. To take a concrete example, if F is a pseudorandom function, then given oracle access to F_k (for uniform k) it must be hard to find an input x for which $F_k(x) = 0^n$ (since it would be hard to find such an input for a truly random function f). But if k is known, finding such an input may be easy.

Example 3.26

As usual, we can gain familiarity with the definition by looking at an insecure example. Define the keyed, length-preserving function F by $F(k, x) = k \oplus x$. For any input x, the value of $F_k(x)$ is uniformly distributed (when k is uniform). Nevertheless, F is not pseudorandom since its values on any *two* points are correlated. Concretely, consider the distinguisher D that queries its oracle $\mathcal{O}$ on arbitrary, distinct points x_1, x_2 to obtain values $y_1 = \mathcal{O}(x_1)$ and $y_2 = \mathcal{O}(x_2)$, and outputs 1 if and only if $y_1 \oplus y_2 = x_1 \oplus x_2$. If $\mathcal{O} = F_k$, for any k, then D outputs 1. On the other hand, if $\mathcal{O} = f$ for f chosen uniformly from Func_n, then the probability that $f(x_1) \oplus f(x_2) = x_1 \oplus x_2$ is exactly the probability that $f(x_2) = x_1 \oplus x_2 \oplus f(x_1)$, or 2^{-n}, and D outputs 1 with this probability. The difference is $|1 - 2^{-n}|$, which is not negligible. $\diamondsuit$

Pseudorandom Permutations/Block Ciphers

Let Perm_n be the set of all permutations (i.e., bijections) on $\{0,1\}^n$. Viewing any $f \in \mathsf{Perm}_n$ as a look-up table as before, we now have the added constraint that the entries in any two distinct rows must be different. We have 2^n different choices for the entry in the first row of the table; once we fix this entry, we are left with only $2^n - 1$ choices for the second row, and so on. We thus see that the size of Perm_n is $(2^n)!$.

Let F be a keyed function. We call F a *keyed permutation* if $\ell_{in} = \ell_{out}$, and furthermore for all $k \in \ell_{key}(n)$ the function $F_k : \{0,1\}^{\ell_{in}(n)} \to \{0,1\}^{\ell_{in}(n)}$ is one-to-one (i.e., F_k is a permutation). We call ℓ_{in} the *block length* of F. As before, unless stated otherwise we assume F is *length-preserving* and so $\ell_{key}(n) = \ell_{in}(n) = n$. A keyed permutation is *efficient* if there is a polynomial-time algorithm for computing $F_k(x)$ given k and x, as well as a polynomial-time algorithm for computing $F_k^{-1}(y)$ given k and y. That is, F_k should be both efficiently computable and *efficiently invertible* given k.

The definition of what it means for an efficient, keyed permutation F to be a *pseudorandom permutation* is exactly analogous to Definition 3.25, with the only difference being that now we require F_k to be indistinguishable from a uniform *permutation* rather than a uniform function. That is, we require that no efficient algorithm can distinguish between access to F_k (for uniform key k) and access to f (for uniform $f \in \mathsf{Perm}_n$). It turns out that this is merely an aesthetic choice since, whenever the block length is sufficiently long, a random permutation is itself indistinguishable from a random function. Intuitively this is due to the fact that a uniform function f looks identical to a uniform permutation unless distinct values x and y are found for which $f(x) = f(y)$, since in such a case the function cannot be a permutation. However, the probability of finding such values x, y using a polynomial number of queries is negligible. (This follows from the results of Appendix A.4.)

PROPOSITION 3.27 *If F is a pseudorandom permutation and additionally $\ell_{in}(n) \geq n$, then F is also a pseudorandom function.*

If F is a keyed permutation then cryptographic schemes based on F might require the honest parties to compute the inverse F_k^{-1} in addition to computing F_k itself. This potentially introduces new security concerns. In particular, it may now be necessary to impose the stronger requirement that F_k be indistinguishable from a uniform permutation *even if the distinguisher is additionally given oracle access to the inverse of the permutation.* If F has this property, we call it a *strong* pseudorandom permutation.

DEFINITION 3.28 *Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, length-preserving, keyed permutation. F is a* strong pseudorandom permutation *if for all probabilistic polynomial-time distinguishers D, there exists a negligible function* negl *such that:*

$$\left| \Pr[D^{F_k(\cdot), F_k^{-1}(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot), f^{-1}(\cdot)}(1^n) = 1] \right| \leq \mathsf{negl}(n),$$

where the first probability is taken over uniform choice of $k \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $f \in \mathsf{Perm}_n$ and the randomness of D.

Of course, any strong pseudorandom permutation is also a pseudorandom permutation.

Block ciphers. In practice, *block ciphers* are designed to be secure instantiations of (strong) pseudorandom permutations with some *fixed* key length and block length. We discuss approaches for building block ciphers, and some popular candidate block ciphers, in Chapter 6. For the purposes of the present chapter the details of these constructions are unimportant, and for now we simply assume that (strong) pseudorandom permutations exist.

Pseudorandom functions and pseudorandom generators. As one might expect, there is a close relationship between pseudorandom functions and pseudorandom generators. It is fairly easy to construct a pseudorandom generator G from a pseudorandom function F by simply evaluating F on a series of different inputs; e.g., we can define $G(s) \stackrel{\text{def}}{=} F_s(1)\|F_s(2)\|\cdots\|F_s(\ell)$ for any desired ℓ. If F_s were replaced by a uniform function f, the output of G would be uniform; thus, when using F instead, the output is pseudorandom. You are asked to prove this formally in Exercise 3.14.

More generally, we can use the above idea to construct a stream cipher $(\mathsf{Init}, \mathsf{GetBits})$ that accepts an initialization vector IV. (See Section 3.3.1.) The only difference is that instead of evaluating F_s on the fixed input sequence $1, 2, 3, \ldots$, we evaluate F on the inputs $IV + 1$, $IV + 2$, $\ldots$.

CONSTRUCTION 3.29

Let F be a pseudorandom function. Define a stream cipher $(\mathsf{Init}, \mathsf{GetBits})$, where each call to $\mathsf{GetBits}$ outputs n bits, as follows:

- Init: on input $s \in \{0,1\}^n$ and $IV \in \{0,1\}^n$, set $\mathsf{st}_0 := (s, IV)$.
- $\mathsf{GetBits}$: on input $\mathsf{st}_i = (s, IV)$, compute $IV' := IV + 1$ and set $y := F_s(IV')$ and $\mathsf{st}_{i+1} := (s, IV')$. Output (y, st_{i+1}).

A stream cipher from any pseudorandom function/block cipher.

Although stream ciphers can be constructed from block ciphers, dedicated stream ciphers used in practice typically have better performance, especially in resource-constrained environments. On the other hand, stream ciphers appear to be less well understood (in practice) than block ciphers, and confidence in their security is lower. It is therefore recommended to use block ciphers (possibly by converting them to stream ciphers first) whenever possible.

Considering the other direction, a pseudorandom generator G immediately gives a pseudorandom function F *with small block length.* Specifically, say G has expansion factor $n \cdot 2^{t(n)}$. We can define the keyed function $F : \{0,1\}^n \times \{0,1\}^{t(n)} \to \{0,1\}^n$ as follows: to compute $F_k(i)$, first compute $G(k)$ and interpret the result as a look-up table with $2^{t(n)}$ rows each containing n bits; output the ith row. This runs in polynomial time only if $t(n) = \mathcal{O}(\log n)$. It is possible, though more difficult, to construct pseudorandom functions *with large block length* from pseudorandom generators; this is shown in Section 7.5. Pseudorandom generators, in turn, can be constructed based on certain mathematical problems conjectured to be hard. The existence of pseudorandom functions based on these hard mathematical problems represents one of the amazing contributions of modern cryptography.

3.5.2 CPA-Secure Encryption from Pseudorandom Functions

We focus here on constructing a CPA-secure, fixed-length encryption scheme. By what we have said at the end of Section 3.4.2, this implies the existence of a CPA-secure encryption scheme for arbitrary-length messages. In Section 3.6 we will discuss more efficient ways of encrypting messages of arbitrary length.

A naive attempt at constructing a secure encryption scheme from a pseudorandom permutation is to define $\mathsf{Enc}_k(m) = F_k(m)$. Although we expect that this "reveals no information about m" (since, if f is a uniform function, then $f(m)$ is simply a uniform n-bit string), this method of encryption is *deterministic* and so cannot possibly be CPA-secure. In particular, encrypting the same plaintext twice will yield the same ciphertext.

Our secure construction is *randomized*. Specifically, we encrypt by applying the pseudorandom function to a *random value* r (rather than the message) and XORing the result with the plaintext. (See Figure 3.3 and Construction 3.30.) This can again be viewed as an instance of XORing a pseudorandom pad with the plaintext, with the major difference being the fact that a *fresh* pseudorandom pad is used each time. (In fact, the pseudorandom pad is only "fresh" if the pseudorandom function is applied to a "fresh" value on which it has never been applied before. While it is possible that a random r will be equal to some r-value chosen previously, this happens with only negligible probability.)

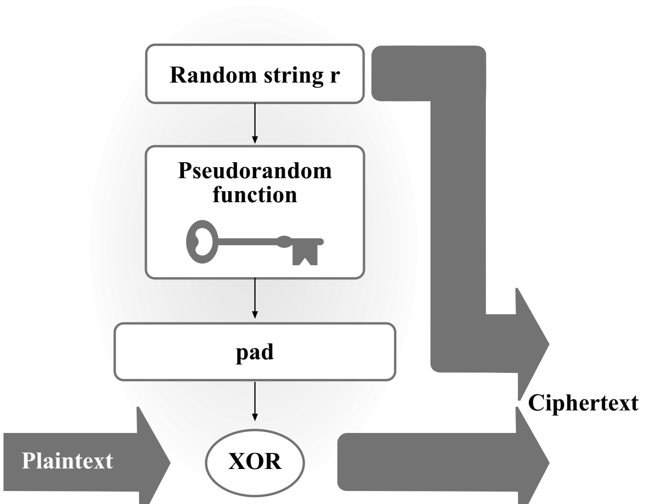

FIGURE 3.3: Encryption with a pseudorandom function.

Proofs of security based on pseudorandom functions. Before turning to the proof that the above construction is CPA-secure, we highlight a common template that is used by most proofs of security (even outside the context of encryption) for constructions based on pseudorandom functions. The first step of such proofs is to consider a hypothetical version of the construction in which the pseudorandom function is replaced with a random function. It is then argued—using a proof by reduction—that this modification does not significantly affect the attacker's success probability. We are then left with analyzing a scheme that uses a completely random function. At this point the rest of the proof typically relies on probabilistic analysis and does not rely on any computational assumptions. We will utilize this proof template several times in this and the next chapter.

CONSTRUCTION 3.30

Let F be a pseudorandom function. Define a private-key encryption scheme for messages of length n as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, choose uniform $r \in \{0,1\}^n$ and output the ciphertext

$$c := \langle r,\, F_k(r) \oplus m \rangle.$$

- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the plaintext message

$$m := F_k(r) \oplus s.$$

A CPA-secure encryption scheme from any pseudorandom function.

THEOREM 3.31 *If F is a pseudorandom function, then Construction 3.30 is a CPA-secure private-key encryption scheme for messages of length n.*

PROOF Let $\widetilde{\Pi} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ be an encryption scheme that is exactly the same as $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ from Construction 3.30, except that a truly random function f is used in place of F_k. That is, $\widetilde{\mathsf{Gen}}(1^n)$ chooses a uniform function $f \in \mathsf{Func}_n$, and $\widetilde{\mathsf{Enc}}$ encrypts just like Enc except that f is used instead of F_k. (This modified encryption scheme is not efficient. But we can still define it as a hypothetical encryption scheme for the sake of the proof.)

Fix an arbitrary PPT adversary $\mathcal{A}$, and let $q(n)$ be an upper bound on the number of queries that $\mathcal{A}(1^n)$ makes to its encryption oracle. (Note that q must be upper-bounded by some polynomial.) As the first step of the proof, we show that there is a negligible function negl such that

$$\left| \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1 \right] - \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1 \right] \right| \leq \mathsf{negl}(n). \tag{3.8}$$

We prove this by reduction. We use $\mathcal{A}$ to construct a distinguisher D for the pseudorandom function F. The distinguisher D is given oracle access to some function $\mathcal{O}$, and its goal is to determine whether this function is "pseudorandom" (i.e., equal to F_k for uniform $k \in \{0,1\}^n$) or "random" (i.e., equal to f for uniform $f \in \mathsf{Func}_n$). To do this, D emulates experiment $\mathsf{PrivK}^{\mathsf{cpa}}$ for $\mathcal{A}$ in the manner described below, and observes whether $\mathcal{A}$ succeeds or not. If $\mathcal{A}$ succeeds then D guesses that its oracle must be a pseudorandom function, whereas if $\mathcal{A}$ does not succeed then D guesses that its oracle must be a random function. In detail:

Distinguisher D:
D is given input 1^n and access to an oracle $\mathcal{O} : \{0,1\}^n \to \{0,1\}^n$.

1. Run $\mathcal{A}(1^n)$. Whenever $\mathcal{A}$ queries its encryption oracle on a message $m \in \{0,1\}^n$, answer this query in the following way:
   (a) Choose uniform $r \in \{0,1\}^n$.
   (b) Query $\mathcal{O}(r)$ and obtain response y.
   (c) Return the ciphertext $\langle r, y \oplus m \rangle$ to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs messages $m_0, m_1 \in \{0,1\}^n$, choose a uniform bit $b \in \{0,1\}$ and then:
   (a) Choose uniform $r \in \{0,1\}^n$.
   (b) Query $\mathcal{O}(r)$ and obtain response y.
   (c) Return the challenge ciphertext $\langle r, y \oplus m_b \rangle$ to $\mathcal{A}$.
3. Continue answering encryption-oracle queries of $\mathcal{A}$ as before until $\mathcal{A}$ outputs a bit b'. Output 1 if $b' = b$, and 0 otherwise.

D runs in polynomial time since $\mathcal{A}$ does. The key points are as follows:

1. If D's oracle is a pseudorandom function, then the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$. This is because, in this case, a key k is chosen uniformly at random and then every encryption is carried out by choosing a uniform r, computing $y := F_k(r)$, and setting the ciphertext equal to $\langle r, y \oplus m \rangle$, exactly as in Construction 3.30. Thus,

$$\Pr_{k \leftarrow \{0,1\}^n}\left[D^{F_k(\cdot)}(1^n) = 1\right] = \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1\right], \tag{3.9}$$

   where we emphasize that k is chosen uniformly on the left-hand side.

2. If D's oracle is a random function, then the view of $\mathcal{A}$ when run as a subroutine by D is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)$. This can be seen exactly as above, with the only difference being that a uniform function $f \in \mathsf{Func}_n$ is used instead of F_k. Thus,

$$\Pr_{f \leftarrow \mathsf{Func}_n}\left[D^{f(\cdot)}(1^n) = 1\right] = \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right], \tag{3.10}$$

   where f is chosen uniformly from Func_n on the left-hand side.

By the assumption that F is a pseudorandom function (and since D is efficient), there exists a negligible function negl for which

$$\left|\Pr\left[D^{F_k(\cdot)}(1^n)=1\right]-\Pr\left[D^{f(\cdot)}(1^n)=1\right]\right|\le \mathsf{negl}(n).$$

Combining the above with Equations (3.9) and (3.10) gives Equation (3.8).

For the second part of the proof, we show that

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1\right]\le \frac{1}{2}+\frac{q(n)}{2^n}. \tag{3.11}$$

(Recall that $q(n)$ is a bound on the number of encryption queries made by $\mathcal{A}$. The above holds even if we place no computational restrictions on $\mathcal{A}$.) To see that Equation (3.11) holds, observe that every time a message m is encrypted in $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)$ (either by the encryption oracle or when the challenge ciphertext is computed), a uniform $r\in\{0,1\}^n$ is chosen and the ciphertext is set equal to $\langle r,\, f(r)\oplus m\rangle$. Let r^* denote the random string used when generating the challenge ciphertext $\langle r^*, f(r^*)\oplus m_b\rangle$. There are two possibilities:

1. *The value r^* is never used when answering any of $\mathcal{A}$'s encryption-oracle queries:* In this case, $\mathcal{A}$ learns nothing about $f(r^*)$ from its interaction with the encryption oracle (since f is a truly random function). This means that, as far as $\mathcal{A}$ is concerned, the value $f(r^*)$ that is XORed with m_b is uniformly distributed and independent of the rest of the experiment, and so the probability that $\mathcal{A}$ outputs $b'=b$ in this case is exactly $1/2$ (as in the case of the one-time pad).

2. *The value r^* is used when answering at least one of $\mathcal{A}$'s encryption-oracle queries:* In this case, $\mathcal{A}$ may easily determine whether m_0 or m_1 was encrypted. This is so because if the encryption oracle ever returns a ciphertext $\langle r^*, s\rangle$ in response to a request to encrypt the message m, the adversary learns that $f(r^*)=s\oplus m$.

   However, since $\mathcal{A}$ makes at most $q(n)$ queries to its encryption oracle (and thus at most $q(n)$ values of r are used when answering $\mathcal{A}$'s encryption-oracle queries), and since r^* is chosen uniformly from $\{0,1\}^n$, the probability of this event is at most $q(n)/2^n$.

Let Repeat denote the event that r^* is used by the encryption oracle when answering at least one of $\mathcal{A}$'s queries. As just discussed, the probability of Repeat is at most $q(n)/2^n$, and the probability that $\mathcal{A}$ succeeds in $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}$ if Repeat does not occur is exactly $1/2$. Therefore:

$$\begin{aligned}
&\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1]\\
&\quad=\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1\wedge \mathsf{Repeat}]+\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1\wedge \overline{\mathsf{Repeat}}]\\
&\quad\le \Pr[\mathsf{Repeat}]+\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1\mid \overline{\mathsf{Repeat}}]\ \le\ \frac{q(n)}{2^n}+\frac{1}{2}.
\end{aligned}$$

Combining the above with Equation (3.8), we see that there is a negligible function negl such that $\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \frac{q(n)}{2^n} + \mathsf{negl}(n)$. Since q is polynomial, $\frac{q(n)}{2^n}$ is negligible. In addition, the sum of two negligible functions is negligible, and thus there exists a negligible function negl' such that $\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}'(n)$, completing the proof. ■

Concrete security. The above proof shows that

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

for some negligible function negl. The final term depends on the security of F as a pseudorandom function; it is a bound on the distinguishing probability of algorithm D (which has roughly the same running time as the adversary $\mathcal{A}$). The term $\frac{q(n)}{2^n}$ represents a bound on the probability that the value r^* used to encrypt the challenge ciphertext was used to encrypt some other message.

3.6 Modes of Operation

Modes of operation provide a way to securely (and efficiently) encrypt long messages using stream or block ciphers.

3.6.1 Stream-Cipher Modes of Operation

Construction 3.17 provides a way to construct an encryption scheme using a pseudorandom generator. That scheme has two main drawbacks. First, as presented, the length of the message to be encrypted must be fixed and known in advance. Second, the scheme is only EAV-secure, not CPA-secure.

Stream ciphers, which can be viewed as flexible pseudorandom generators, can be used to address these drawbacks. In practice, stream ciphers are used for encryption in two ways: *synchronized mode* and *unsynchronized mode.*

Synchronized mode. In this stateful encryption scheme, the sender and receiver must be *synchronized* in the sense that they know how much plaintext has been encrypted (resp., decrypted) so far. Synchronized mode is typically used in a single communication session between parties (see Section 4.5.3), where statefulness is acceptable and messages are received in order without being lost. The intuition here is that a long pseudorandom stream is generated, and a different part of it is used to encrypt each message. Synchronization is needed to ensure correct decryption (i.e., so the receiver knows which part of the stream was used to encrypt the next message), and to ensure that no portion of the stream is re-used. We describe this mode in detail next.

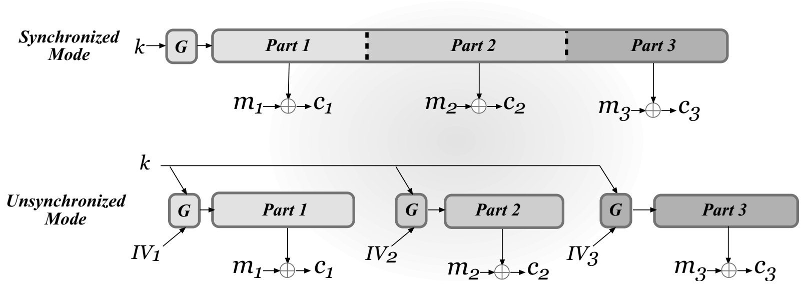

FIGURE 3.4: Synchronized mode and unsynchronized mode.

We have seen in Algorithm 3.16 that a stream cipher can be used to construct a pseudorandom generator G_ℓ with any desired expansion factor ℓ. We can easily modify that algorithm to obtain a pseudorandom generator G_∞ with *variable* output length. G_∞ takes two inputs: a seed s and a desired output length 1^ℓ (we specify this in unary since G_∞ will run in time polynomial in ℓ). As in Algorithm 3.16, $G_\infty(s, 1^\ell)$ runs $\mathsf{Init}(s)$ and then repeatedly runs $\mathsf{GetBits}$ a total of ℓ times.

We can use G_∞ in Construction 3.17 to handle encryption of arbitrary-length messages: encryption of a message m using the key k is done by computing the ciphertext $c := G_\infty(k, 1^{|m|}) \oplus m$; decryption of a ciphertext c using the key k is carried out by computing the message $m := G_\infty(k, 1^{|c|}) \oplus c$. A minor modification of the proof of Theorem 3.18 shows that if the stream cipher is secure then this encryption scheme has indistinguishable encryptions in the presence of an eavesdropper.

A little thought shows that if the communicating parties are willing to maintain state, then they can use the same key to encrypt *multiple* messages. (See Figure 3.4.) The conceptual insight is that the parties can treat multiple messages $m_1, m_2, \ldots$ as a single, long message; furthermore, Construction 3.17 (as well as the modified version in the previous paragraph) has the property that initial portions of a message can be encrypted and transmitted even if the rest of the message is not yet known. Concretely, the parties share a key k and both begin by computing $\mathsf{st}_0 := \mathsf{Init}(k)$. To encrypt the first message m_1 of length ℓ_1, the sender repeatedly runs $\mathsf{GetBits}$ a total of ℓ_1 times, beginning at st_0, to obtain a stream of bits $\mathsf{pad}_1 \overset{\text{def}}{=} y_1, \ldots, y_{\ell_1}$ along with updated state st_{ℓ_1}; it then sends $c_1 := \mathsf{pad}_1 \oplus m_1$. Upon receiving c_1, the other party repeatedly runs $\mathsf{GetBits}$ a total of ℓ_1 times to obtain the same values pad_1 and st_{ℓ_1}; it uses pad_1 to recover $m_1 := \mathsf{pad}_1 \oplus c_1$. Later, to encrypt a second message m_2 of length ℓ_2, the sender will repeatedly run $\mathsf{GetBits}$ a total of ℓ_2 times, beginning at st_{ℓ_1}, to obtain $\mathsf{pad}_2 \overset{\text{def}}{=} y_{\ell_1+1}, \ldots, y_{\ell_1+\ell_2}$ and updated state $\mathsf{st}_{\ell_1+\ell_2}$, and then compute the ciphertext $c_2 := \mathsf{pad}_2 \oplus m_2$, and so on. This can continue indefinitely, allowing the parties to send an unlimited number of messages of arbitrary length. We remark that in this mode, the stream cipher does not need to use an IV.

This method of encrypting multiple messages requires the communicating parties to maintain synchronized state, explaining the terminology "synchronized mode." For that reason, the method is appropriate when two parties are communicating within a single "session," but it does not work well for sporadic communication or when a party might, over time, communicate from different devices. (It is relatively easy to keep copies of a fixed key in different locations; it is harder to maintain synchronized state across multiple locations.) Furthermore, if the parties ever get out of sync (e.g., because one of the transmissions between the parties is dropped), decryption will return an incorrect result. Resynchronization is possible, but adds additional overhead.

Unsynchronized mode. For stream ciphers whose Init function accepts an initialization vector as input, we can achieve *stateless* CPA-secure encryption for messages of arbitrary length. Here we modify G_∞ to accept three inputs: a seed s, an initialization vector IV, and a desired output length 1^ℓ. Now this algorithm first computes $\mathsf{st}_0 := \mathsf{Init}(s, IV)$ before repeatedly running $\mathsf{GetBits}$ a total of ℓ times. Encryption can then be carried out using a variant of Construction 3.30: the encryption of a message m using the key k is done by choosing a uniform initialization vector $IV \in \{0,1\}^n$ and computing the ciphertext $\langle IV, G_\infty(s, IV, 1^{|m|}) \oplus m\rangle$; decryption is performed in the natural way. (See Figure 3.4.) This scheme is CPA-secure if the stream cipher now has the stronger property that, for any polynomial ℓ, the function F defined by $F_k(IV) \overset{\text{def}}{=} G_\infty(k, IV, 1^\ell)$ is a pseudorandom function. (In fact, F need only be pseudorandom when evaluated on *uniform* inputs. Keyed functions with this weaker property are called *weak* pseudorandom functions.)

3.6.2 Block-Cipher Modes of Operation

We have already seen a construction of a CPA-secure encryption scheme based on pseudorandom functions/block ciphers. But Construction 3.30 (and its extension to arbitrary-length messages as discussed at the end of Section 3.4.2) have the drawback that the length of the ciphertext is *double* the length of the plaintext. Block-cipher modes of operation provide a way of encrypting arbitrary-length messages using shorter ciphertexts.

In this section, let F be a block cipher with block length n. We assume here that all messages m being encrypted have length a multiple of n, and write $m = m_1, m_2, \ldots, m_\ell$ where each $m_i \in \{0,1\}^n$ represents a block of the plaintext. Messages that are not a multiple of n can always be unambiguously padded to have length a multiple of n by appending a 1 followed by sufficiently many 0s, and so this assumption is without much loss of generality.

Several block-cipher modes of operation are known; we present four of the most common ones and discuss their security.

Electronic Code Book (ECB) mode. This is a naive mode of operation in which the ciphertext is obtained by direct application of the block cipher to each plaintext block. That is, $c := \langle F_k(m_1), F_k(m_2), \ldots, F_k(m_\ell)\rangle$; see

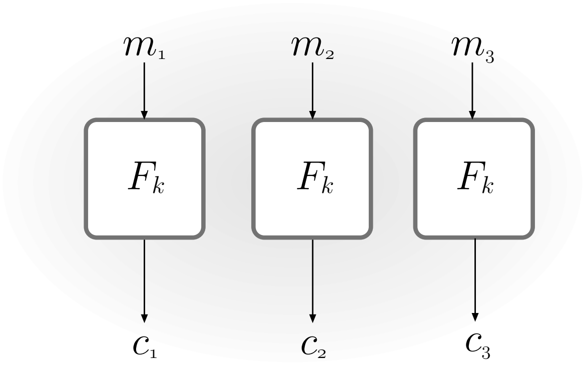

FIGURE 3.5: Electronic Code Book (ECB) mode.

Figure 3.5. Decryption is done in the obvious way, using the fact that F_k^{-1} is efficiently computable.

ECB mode is *deterministic* and therefore cannot be CPA-secure. Worse, ECB-mode encryption does not even have indistinguishable encryptions in the presence of an eavesdropper. This is because if a block is repeated in the plaintext, it will result in a repeating block in the ciphertext. Thus, it is easy to distinguish an encryption of a plaintext that consists of two identical blocks from an encryption of a plaintext that consists of two different blocks. This is not just a theoretical problem. Consider encrypting an image in which small groups of pixels correspond to a plaintext block. Encrypting using ECB mode may reveal a significant amount of information about patterns in the image, something that should not happen when using a secure encryption scheme. Figure 3.6 demonstrates this.

For these reasons, ECB mode should never be used. (We include it only because of its historical significance.)

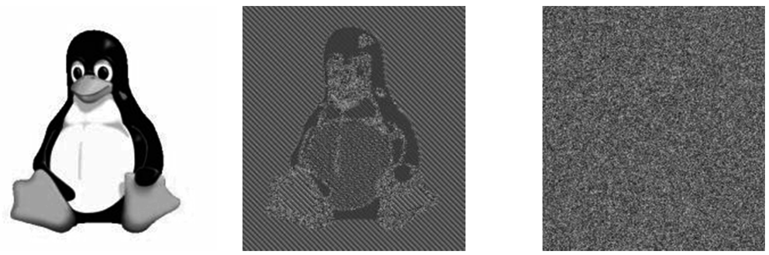

FIGURE 3.6: An illustration of the dangers of using ECB mode. The middle figure is an encryption of the image on the left using ECB mode; the figure on the right is an encryption of the same image using a secure mode. (Taken from `http://en.wikipedia.org` and derived from images created by Larry Ewing (`lewing@isc.tamu.edu`) using The GIMP.)

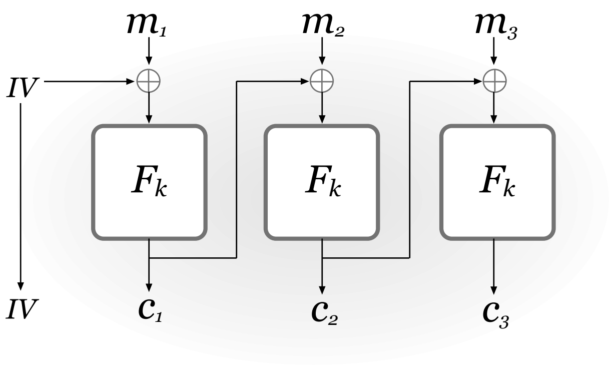

FIGURE 3.7: Cipher Block Chaining (CBC) mode.

Cipher Block Chaining (CBC) mode. To encrypt using this mode, a uniform initialization vector (IV) of length n is first chosen. Then, ciphertext blocks are generated by applying the block cipher to the XOR of the current plaintext block and the previous ciphertext block. That is, set $c_0 := IV$ and then, for $i = 1$ to ℓ, set $c_i := F_k(c_{i-1} \oplus m_i)$. The final ciphertext is $\langle c_0, c_1, \ldots, c_\ell \rangle$. (See Figure 3.7.) Decryption of a ciphertext $c_0, \ldots, c_\ell$ is done by computing $m_i := F_k^{-1}(c_i) \oplus c_{i-1}$ for $i = 1, \ldots, \ell$. We stress that the IV is included in the ciphertext; this is crucial so decryption can be carried out.

Importantly, encryption in CBC mode is probabilistic and it has been proven that if F is a pseudorandom permutation then CBC-mode encryption is CPA-secure. The main drawback of this mode is that encryption must be carried out sequentially because the ciphertext block c_{i-1} is needed in order to encrypt the plaintext block m_i. Thus, if parallel processing is available, CBC-mode encryption may not be the most efficient choice.

One may be tempted to think that it suffices to use a *distinct* IV (rather than a random IV) for every encryption, e.g., to first use $IV = 1$ and then increment the IV by one each time a message is encrypted. In Exercise 3.20, we ask you to show that this variant of CBC-mode encryption is not secure.

One might also consider a stateful variant of CBC-mode encryption—called *chained CBC mode*—in which the last block of the previous ciphertext is used as the IV when encrypting the next message. This reduces the bandwidth, as the IV need not be sent each time. See Figure 3.8, where an initial message m_1, m_2, m_3 is encrypted using a random IV, and then subsequently a second message m_4, m_5 is encrypted using c_3 as the IV. (In contrast, encryption using stateless CBC mode would generate a fresh IV when encrypting the second message.) Chained CBC mode is used in SSL 3.0 and TLS 1.0.

It may appear that chained CBC mode is as secure as CBC mode, since the chained-CBC encryption of m_1, m_2, m_3 followed by encryption of m_4, m_5 yields the same ciphertext blocks as CBC-mode encryption of the (single) message m_1, m_2, m_3, m_4, m_5. Nevertheless, chained CBC mode is vulnerable to a chosen-plaintext attack. The basis of the attack is that the adversary knows in

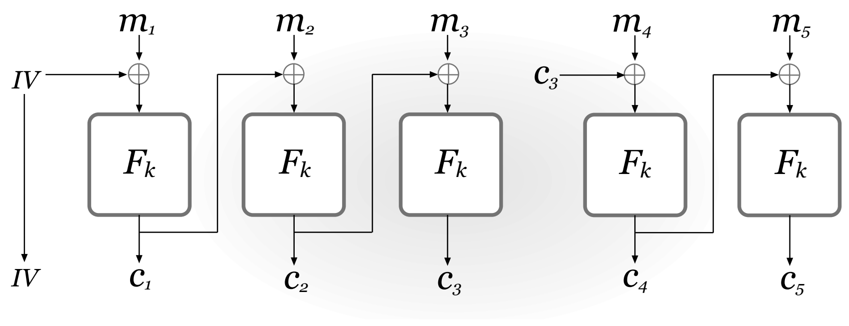

FIGURE 3.8: Chained CBC.

advance the "initialization vector" that will be used for the second encrypted message. We describe the attack informally, based on Figure 3.8. Assume the attacker knows that $m_1 \in \{m_1^0, m_1^1\}$, and observes the first ciphertext IV, c_1, c_2, c_3. The attacker then requests an encryption of a second message m_4, m_5 with $m_4 = IV \oplus m_1^0 \oplus c_3$, and observes a second ciphertext c_4, c_5. One can verify that $m_1 = m_1^0$ if and only if $c_4 = c_1$. This example should serve as a strong warning against making any modifications to cryptographic schemes, even if those modifications seem benign.

Output Feedback (OFB) mode. The third mode we present can be viewed as an unsynchronized stream-cipher mode, where the stream cipher is constructed in a specific way from the underlying block cipher. We describe the mode directly. First, a uniform $IV \in \{0,1\}^n$ is chosen. Then, a pseudorandom stream is generated from IV in the following way: Define $y_0 := IV$, and set the ith block y_i of the stream to be $y_i := F_k(y_{i-1})$. Each block of the plaintext is encrypted by XORing it with the appropriate block of the stream; that is, $c_i := y_i \oplus m_i$. (See Figure 3.9.) As in CBC mode, the IV is included as part of the ciphertext to enable decryption. However, in contrast to CBC mode, here it is not required that F be invertible. (In fact, it need

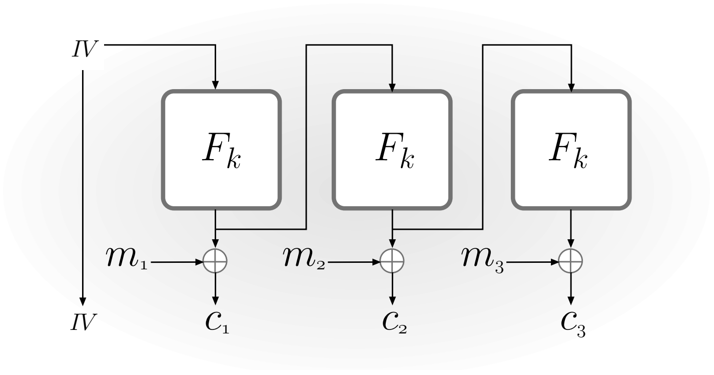

FIGURE 3.9: Output Feedback (OFB) mode.

not even be a permutation.) Furthermore, as in stream-cipher modes of operation, here it is not necessary for the plaintext length to be a multiple of the block length. Instead, the generated stream can be truncated to exactly the plaintext length. Another advantage of OFB mode is that its stateful variant (in which the final value y_ℓ used to encrypt some message is used as the IV for encrypting the next message) *is* secure. This stateful variant is equivalent to a *synchronized* stream-cipher mode, with the stream cipher constructed from the block cipher in the specific manner just described.

OFB mode can be shown to be CPA-secure if F is a pseudorandom function. Although both encryption and decryption must be carried out sequentially, this mode has the advantage relative to CBC mode that the bulk of the computation (namely, computation of the pseudorandom stream) can be done independently of the actual message to be encrypted. So, it is possible to generate a pseudorandom stream ahead of time using preprocessing, after which point encryption of the plaintext (once it is known) is incredibly fast.

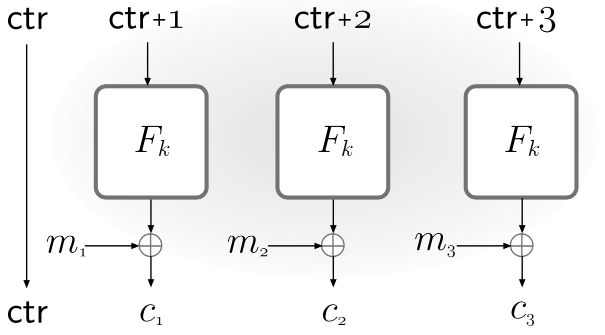

FIGURE 3.10: Counter (CTR) mode.

Counter (CTR) mode. Counter mode can also be viewed as an unsynchronized stream-cipher mode, where the stream cipher is constructed from the block cipher as in Construction 3.29. We give a self-contained description here. To encrypt using CTR mode, a uniform value $\mathsf{ctr} \in \{0,1\}^n$ is first chosen. Then, a pseudorandom stream is generated by computing $y_i := F_k(\mathsf{ctr} + i)$, where ctr and i are viewed as integers and addition is done modulo 2^n. The ith ciphertext block is $c_i := y_i \oplus m_i$, and the IV is again sent as part of the ciphertext; see Figure 3.10. Note again that decryption does not require F to be invertible, or even a permutation. As with OFB mode, another "stream-cipher" mode, the generated stream can be truncated to exactly the plaintext length, preprocessing can be used to generate the pseudorandom stream before the message is known, and the stateful variant of CTR mode is secure.

In contrast to all the secure modes discussed previously, CTR mode has the advantage that encryption and decryption can be fully *parallelized*, since all the blocks of the pseudorandom stream can be computed independently of each other. In contrast to OFB, it is also possible to decrypt the ith block of the ciphertext using only a single evaluation of F. These features make CTR mode an attractive choice.

CTR mode is also fairly simple to analyze:

THEOREM 3.32 *If F is a pseudorandom function, then CTR mode is CPA-secure.*

PROOF We follow the same template as in the proof of Theorem 3.31: we first replace F with a random function and then analyze the resulting scheme.

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be the (stateless) CTR-mode encryption scheme, and let $\widetilde{\Pi} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ be the encryption scheme that is identical to Π except that a truly random function is used in place of F_k. That is, $\widetilde{\mathsf{Gen}}(1^n)$ chooses a uniform function $f \in \mathsf{Func}_n$, and $\widetilde{\mathsf{Enc}}$ encrypts just like Enc except that f is used instead of F_k. (Once again, neither $\widetilde{\mathsf{Gen}}$ nor $\widetilde{\mathsf{Enc}}$ is efficient but this does not matter for the purposes of defining an experiment involving $\widetilde{\Pi}$.)

Fix an arbitrary PPT adversary $\mathcal{A}$, and let $q(n)$ be a polynomial upper-bound on the number of encryption-oracle queries made by $\mathcal{A}(1^n)$ as well as the maximum number of blocks in any such query and the maximum number of blocks in m_0 and m_1. As the first step of the proof, we claim that there is a negligible function negl such that

$$\left|\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1\right] - \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right]\right| \leq \mathsf{negl}(n). \tag{3.12}$$

This is proved by reduction in a way similar to the analogous step in the proof of Theorem 3.31, and is left as an exercise for the reader.

We next claim that

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] < \frac{1}{2} + \frac{2q(n)^2}{2^n}. \tag{3.13}$$

Combined with Equation (3.12) this means that

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

Since q is polynomial, $\frac{2q(n)^2}{2^n}$ is negligible and this completes the proof.

We now prove Equation (3.13). Fix some value n for the security parameter. Let $\ell^* \leq q(n)$ denote the length (in blocks) of the messages m_0, m_1 output by $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)$, and let ctr^* denote the initial value used when generating the challenge ciphertext. Similarly, let $\ell_i \leq q(n)$ be the length (in blocks) of the ith encryption-oracle query made by $\mathcal{A}$, and let ctr_i denote the initial value used when answering this query. When the ith encryption-oracle query is answered, f is applied to the values $\mathsf{ctr}_i + 1, \ldots, \mathsf{ctr}_i + \ell_i$. When the challenge ciphertext is encrypted, f is applied to $\mathsf{ctr}^* + 1, \ldots, \mathsf{ctr}^* + \ell^*$, and the ith ciphertext block is computed by XORing $f(\mathsf{ctr}^* + i)$ with the ith message block. There are two cases:

1. *There do not exist any $i, j, j^* \geq 1$ (with $j \leq \ell_i$ and $j^* \leq \ell^*$) for which* $\mathsf{ctr}_i + j = \mathsf{ctr}^* + j^*$*:* In this case, the values $f(\mathsf{ctr}^* + 1), \ldots, f(\mathsf{ctr}^* + \ell^*)$ used when encrypting the challenge ciphertext are uniformly distributed and independent of the rest of the experiment since f was not applied to any of these inputs when encrypting the adversary's oracle queries. This means that the challenge ciphertext is computed by XORing a stream of uniform bits with the message m_b, and so the probability that $\mathcal{A}$ outputs $b' = b$ is exactly $1/2$ (as in the case of the one-time pad).

2. *There exist $i, j, j^* \geq 1$ (with $j \leq \ell_i$ and $j^* \leq \ell^*$) for which* $\mathsf{ctr}_i + j = \mathsf{ctr}^* + j^*$*:* We denote this event by $\mathsf{Overlap}$. In this case $\mathcal{A}$ may potentially determine which of its messages was encrypted to give the challenge ciphertext, since $\mathcal{A}$ can deduce the value of $f(\mathsf{ctr}_i + j) = f(\mathsf{ctr}^* + j^*)$ from the answer to its ith oracle query.

Let us analyze the probability that $\mathsf{Overlap}$ occurs. The probability is maximized if ℓ^* and each ℓ_i are as large as possible, so assume $\ell^* = \ell_i = q(n)$ for all i. Let $\mathsf{Overlap}_i$ denote the event that the sequence $\mathsf{ctr}_i + 1, \ldots, \mathsf{ctr}_i + q(n)$ overlaps the sequence $\mathsf{ctr}^* + 1, \ldots, \mathsf{ctr}^* + q(n)$; then $\mathsf{Overlap}$ is the event that $\mathsf{Overlap}_i$ occurs for some i. Since there are at most $q(n)$ oracle queries, a union bound (cf. Proposition A.7) gives

$$\Pr[\mathsf{Overlap}] \leq \sum_{i=1}^{q(n)} \Pr[\mathsf{Overlap}_i]. \tag{3.14}$$

Fixing ctr^*, event $\mathsf{Overlap}_i$ occurs exactly when ctr_i satisfies

$$\mathsf{ctr}^* - q(n) + 1 \leq \mathsf{ctr}_i \leq \mathsf{ctr}^* + q(n) - 1.$$

Since there are $2q(n) - 1$ values of ctr_i for which $\mathsf{Overlap}_i$ occurs, and ctr_i is chosen uniformly from $\{0,1\}^n$, we see that

$$\Pr[\mathsf{Overlap}_i] = \frac{2q(n) - 1}{2^n} < \frac{2q(n)}{2^n}.$$

Combined with Equation (3.14), this gives $\Pr[\mathsf{Overlap}] < 2q(n)^2/2^n$.

Given the above, we can easily bound the success probability of $\mathcal{A}$:

$$\begin{aligned}
\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1] &= \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1 \wedge \overline{\mathsf{Overlap}}] \\
&\quad + \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1 \wedge \mathsf{Overlap}] \\
&\leq \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1 \mid \overline{\mathsf{Overlap}}] + \Pr[\mathsf{Overlap}] \\
&< \frac{1}{2} + \frac{2q(n)^2}{2^n}.
\end{aligned}$$

This proves Equation (3.13) and completes the proof. ■

Modes of operation and message tampering. In many texts, modes of operation are also compared based on how well they protect against adversarial modification of the ciphertext. We do *not* include such a comparison here because the issue of *message integrity* or *message authentication* should be dealt with separately from encryption, and we do so in the next chapter. None of the above modes achieves message integrity in the sense we will define there. We defer further discussion to the next chapter.

With regard to the behavior of different modes in the presence of "benign" (i.e., non-adversarial) transmission errors, see Exercises 3.21 and 3.22. We remark, however, that in general such errors can be addressed using standard techniques (e.g., error correction or re-transmission).

Block length and concrete security. CBC, OFB, and CTR modes all use a random IV. This has the effect of randomizing the encryption process, and ensures that (with high probability) the underlying block cipher is always evaluated on *fresh* (i.e., new) inputs. This is important because, as we have seen in the proofs of Theorem 3.31 and Theorem 3.32, if an input to the block cipher is used more than once then security can be violated.

The block length of a block cipher thus has a significant impact on the concrete security of encryption schemes based on that cipher. Consider, e.g., CTR mode using a block cipher F with block length ℓ. The IV is then a uniform ℓ-bit string, and we expect an IV to repeat after encrypting about $2^{\ell/2}$ messages (see Appendix A.4). If ℓ is too short, then—even if F is secure as a pseudorandom permutation—the resulting concrete-security bound will be too weak for practical applications. Concretely, if $\ell = 64$ as is the case for DES (a block cipher we will study in Chapter 6), then after $2^{32} \approx 4{,}300{,}000{,}000$ encryptions—or roughly 34 gigabytes of plaintext—a repeated IV is expected to occur. Although this may seem like a lot of data, it is smaller than the capacity of modern hard drives.

IV misuse. In our description and discussion of the various (secure) modes, we have assumed a random IV is chosen each time a message encrypted. What if this assumption goes wrong due, e.g., to poor randomness generation or a mistaken implementation? Certainly we can no longer guarantee security in the sense of Definition 3.22. From a practical point of view, though, the "stream-cipher modes" (OFB and CTR) are much worse than CBC mode. If an IV repeats when using the former, an attacker can XOR the two resulting ciphertexts and learn a lot of information about the *entire* contents of both encrypted messages (as we have seen previously in the context of the one-time pad if the key is re-used). With CBC mode, however, it is likely that after only a few blocks the inputs to the block cipher will "diverge" and the attacker will be unable to learn any information beyond the first few message blocks.

One way to address potential IV misuse is to use stateful encryption, as discussed in the context of OFB and CTR modes. If stateful encryption is not possible, and there is concern about potential IV misuse, then CBC mode is recommended for the reasons described above.

3.7 Chosen-Ciphertext Attacks

3.7.1 Defining CCA-Security

Until now we have defined security against two types of attacks: passive eavesdropping and chosen-plaintext attacks. A *chosen-ciphertext attack* is even more powerful. In a chosen-ciphertext attack, the adversary has the ability not only to obtain encryptions of messages of its choice (as in a chosen-plaintext attack), but also to obtain the *decryption* of ciphertexts of its choice (with one exception discussed later). Formally, we give the adversary access to a *decryption oracle* in addition to an encryption oracle. We present the formal definition and defer further discussion.

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

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

1. *A key k is generated by running $\mathsf{Gen}(1^n)$.*
2. *Adversary $\mathcal{A}$ is given input 1^n and oracle access to $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$. It 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 $\mathcal{A}$. We call c the* challenge ciphertext.
4. *The adversary $\mathcal{A}$ continues to have oracle access to $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$, but is not allowed to query the latter on the challenge ciphertext itself. Eventually, $\mathcal{A}$ outputs a bit b'.*
5. *The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. If the output of the experiment is 1, we say that $\mathcal{A}$* succeeds.

DEFINITION 3.33 *A private-key encryption scheme Π has* indistinguishable encryptions under a chosen-ciphertext attack, *or is* CCA-secure, *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function* negl *such that:*

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

where the probability is taken over all randomness used in the experiment.

For completeness, we remark that the natural analogue of Theorem 3.24 holds for CCA-security as well. (Namely, if a scheme has indistinguishable encryptions under a chosen-ciphertext attack then it has indistinguishable *multiple* encryptions under a chosen-ciphertext attack, defined appropriately.)

In the experiment above, the adversary's access to the decryption oracle is unlimited *except* for the restriction that the adversary may not request decryption of the challenge ciphertext itself. This restriction is necessary or else there is clearly no hope for any encryption scheme to satisfy the definition.

At this point you may be wondering if chosen-ciphertext attacks realistically model any real-world attack. As in the case of chosen-plaintext attacks, we do not expect honest parties to decrypt arbitrary ciphertexts of an adversary's choice. Nevertheless, there may be scenarios where an adversary might be able to *influence* what gets decrypted and learn some partial information about the result. For example:

1. In the Midway example from Section 3.4.2, it is conceivable that the US cryptanalysts might also have tried to send encrypted messages to the Japanese and then monitor their behavior. Such behavior (e.g., movement of forces and the like) could have provided important information about the underlying plaintext.

2. Imagine a user sending encrypted messages to their bank. An adversary may be able to send ciphertexts on behalf of that user; the bank will decrypt those ciphertexts and the adversary may learn something about the result. For example, if a ciphertext corresponds to an *ill-formed* plaintext (e.g., an unintelligible message, or simply one that is not formatted correctly), the adversary may be able to deduce this from the bank's reaction (i.e., the pattern of subsequent communication). A practical attack of this type is presented in Section 3.7.2 below.

3. Encryption is often used in higher-level protocols; e.g., an encryption scheme might be used as part of an authentication protocol where one party sends a ciphertext to the other, who decrypts it and returns the result. In this case, one of the honest parties behaves exactly like a decryption oracle.

Insecurity of the schemes we have studied. None of the encryption schemes we have seen thus far is CCA-secure. We demonstrate this for Construction 3.30, where encryption is computed as $\mathsf{Enc}_k(m) = \langle r, F_k(r) \oplus m \rangle$. Consider an adversary $\mathcal{A}$ running in the CCA indistinguishability experiment who chooses $m_0 = 0^n$ and $m_1 = 1^n$. Then, upon receiving a ciphertext $c = \langle r, s \rangle$, the adversary can flip the first bit of s and ask for a decryption of the resulting ciphertext c'. Since $c' \neq c$, this query is allowed and the decryption oracle answers with either 10^{n-1} (in which case it is clear that $b = 0$) or 01^{n-1} (in which case $b = 1$). This example demonstrates that CCA-security is quite stringent. Any encryption scheme that allows ciphertexts to be "manipulated" in any controlled way cannot be CCA-secure. Thus, CCA-security implies a very important property called *non-malleability*. Loosely speaking, a non-malleable encryption scheme has the property that if the adversary tries to modify a given ciphertext, the result is either an invalid ciphertext or one that decrypts to a plaintext having no relation to the original one. This is a very useful property for schemes used in complex cryptographic protocols.

Constructing a CCA-secure encryption scheme. We show how to construct a CCA-secure encryption scheme in Section 4.5.4. The construction is presented there because it uses tools that we develop in Chapter 4.

3.7.2 Padding-Oracle Attacks

The chosen-ciphertext attack on Construction 3.30 described in the previous section is a bit contrived, since it assumes the attacker can obtain the complete decryption of a modified ciphertext. While this sort of attack is allowed under Definition 3.33, it is not clear that it represents a serious concern in practice. Here we show a chosen-ciphertext attack on a natural and widely used encryption scheme; moreover, the attack only requires the ability of an attacker to determine whether or not a modified ciphertext decrypts correctly. Such information is frequently very easy to obtain since, for example, a server might request a retransmission or terminate a session if it ever receives a ciphertext that does not decrypt correctly, and either of these events would generate a noticeable change in the observed traffic. The attack has been shown to work in practice on various deployed protocols; we give one concrete example at the end of this section.

As mentioned previously, when using CBC mode the length of the plaintext must be a multiple of the block length; if a plaintext does not satisfy this property, it must be padded before being encrypted. We refer to the original plaintext as the *message*, and the result after padding as *encoded data*. The padding scheme we use must allow the receiver to unambiguously determine from the encoded data where the message ends. One popular and standardized approach is to use PKCS #5 padding. Assume the original message has an integral number of bytes, and let L denote the block length (in bytes) of the block cipher being used. Let b denote the number of bytes that need to be appended to the message in order to make the total length of the resulting encoded data a multiple of the block length. Here, b is an integer between 1 and L, inclusive. (We cannot have $b = 0$ since this would lead to ambiguous padding. Thus, if the message length is already a multiple of the block length, L bytes of padding are appended.) Then we append to the message the string containing the integer b (represented in 1 byte, or 2 hexadecimal digits) repeated b times. That is, if 1 byte of padding is needed then the string 0x01 (written in hexadecimal) is appended; if 4 bytes of padding are needed then the hexadecimal string 0x04040404 is appended; etc. The encoded data is then encrypted using regular CBC-mode encryption.

When decrypting, the receiver first applies CBC-mode decryption as usual to recover the encoded data, and then checks that the encoded data is correctly padded. (This is easily done: simply read the value b of the final byte and then verify that the final b bytes of the result all have value b.) If so, then the padding is stripped off and the original message returned. Otherwise, the standard procedure is to return a "bad padding" error (e.g., in Java the standard exception is called `javax.crypto.BadPaddingException`). The presence of such an error message provides an adversary with a *partial* decryption oracle. That is, an adversary can send any ciphertext to the server and learn (based on whether a "bad padding" error is returned or not) whether the underlying encoded data is padded in the correct format. Although this may seem like meaningless information, we show that it enables an adversary to completely recover the original message for any ciphertext of its choice.

We describe the attack on a 3-block ciphertext for simplicity. Let IV, c_1, c_2 be a ciphertext observed by the attacker, and let m_1, m_2 be the underlying encoded data (unknown to the attacker) which corresponds to the padded message, as discussed above. (Each block is L bytes long.) Note that

$$m_2 = F_k^{-1}(c_2) \oplus c_1, \tag{3.15}$$

where k is the key (which is, of course, not known to the attacker) being used by the honest parties. The second block m_2 ends in $\underbrace{0\text{x}b \cdots 0\text{x}b}_{b \text{ times}}$, where we let $0\text{x}b$ denote the 1-byte representation of the integer b. The key property used in the attack is that certain changes to the ciphertext yield predictable changes in the underlying encoded data after CBC-mode decryption. Specifically, let c_1' be identical to c_1 except for a modification in the final byte. Consider decryption of the modified ciphertext IV, c_1', c_2. This will result in encoded data m_1', m_2' where $m_2' = F_k^{-1}(c_2) \oplus c_1'$. Comparing to Equation (3.15) we see that m_2' will be identical to m_2 except for a modification in the final byte. (The value of m_1' is unpredictable, but this will not adversely affect the attack.) Similarly, if c_1' is the same as c_1 except for a change in its ith byte, then decryption of IV, c_1', c_2 will result in m_1', m_2' where m_2' is the same as m_2 except for a change in its ith byte. More generally, if $c_1' = c_1 \oplus \Delta$ for any string Δ, then decryption of IV, c_1', c_2 yields m_1', m_2' where $m_2' = m_2 \oplus \Delta$. The upshot is that the attacker can exercise significant control over the final block of the encoded data.

As a warmup, let us see how the adversary can exploit this to learn b, the amount of padding. (This reveals the length of the original message.) Recall that upon decryption, the receiver looks at the value b of the final byte of the second block of the encoded data, and then verifies that the final b bytes all have the same value. The attacker begins by modifying the first byte of c_1 and sending the resulting ciphertext IV, c_1', c_2 to the receiver. If decryption fails (i.e., the receiver returns an error) then it must be the case that the receiver is checking all L bytes of m_2', and therefore $b = L$! Otherwise, the attacker learns that $b < L$, and it can then repeat the process with the second byte, and so on. The left-most modified byte for which decryption fails reveals exactly the left-most byte being checked by the receiver, and so reveals exactly b.

With b known, the attacker can proceed to learn the bytes of the message one-by-one. We illustrate the idea for the final byte of the message, which we denote by B. The attacker knows that m_2 ends in $0\text{x}B0\text{x}b \cdots 0\text{x}b$ (with $0\text{x}b$ repeated b times) and wishes to learn B. Define

$$\Delta_i \overset{\text{def}}{=} 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}i\ \overbrace{0\text{x}(b+1) \cdots 0\text{x}(b+1)}^{b \text{ times}}$$
$$\oplus\ 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}00\ \overbrace{0\text{x}b \cdots 0\text{x}b}^{b \text{ times}}$$

for $0 \leq i < 2^8$; i.e., the final $b+1$ byes of Δ_i contain the integer i (represented in hexadecimal) followed by the value $(b+1) \oplus b$ (in hexadecimal) repeated b times. If the attacker submits the ciphertext $IV, c_1 \oplus \Delta_i, c_2$ to the receiver then, after CBC-mode decryption, the resulting encoded data will equal $0\text{x}(B \oplus i)0\text{x}(b+1) \cdots 0\text{x}(b+1)$ (with $0\text{x}(b+1)$ repeated b times). Decryption will fail unless $0\text{x}(B \oplus i) = 0\text{x}(b+1)$. The attacker simply needs to try at most 2^8 values $\Delta_0, \ldots, \Delta_{2^8-1}$ until decryption succeeds for some Δ_i, at which point it can deduce that $B = 0\text{x}(b+1) \oplus i$. We leave a full description of how to extend this attack so as to learn the entire plaintext—and not just the final block—as an exercise.

A padding-oracle attack on CAPTCHAs. A CAPTCHA is a distorted image of, say, an English word that is easy for humans to read, but hard for a computer to process. CAPTCHAs are used in order to ensure that a human user—and not some automated software—is interacting with a webpage. CAPTCHAs are used, e.g., by online email services to ensure that humans open accounts; this is important to prevent spammers from automatically opening thousands of accounts and using them to send spam.

One way that CAPTCHAs can be configured is as a separate service run on an independent server. In order to see how this works, we denote a webserver by $\mathcal{S}_W$, the CAPTCHA server by $\mathcal{S}_C$, and the user by $\mathcal{U}$. When the user $\mathcal{U}$ loads a webpage served by $\mathcal{S}_W$, the following events occur: $\mathcal{S}_W$ encrypts a random English word w using a key k that was initially shared between $\mathcal{S}_W$ and $\mathcal{S}_C$, and sends the resulting ciphertext to the user (along with the webpage). $\mathcal{U}$ forwards the ciphertext to $\mathcal{S}_C$, who decrypts it, obtains w, and renders a distorted image of w to $\mathcal{U}$. Finally, $\mathcal{U}$ sends the word w back to $\mathcal{S}_W$ for verification. The interesting observation here is that $\mathcal{S}_C$ decrypts any ciphertext it receives from $\mathcal{U}$ and will issue a "bad padding" error message if decryption fails as described earlier. This presents $\mathcal{U}$ with an opportunity to carry out a padding-oracle attack as described above, and thus to solve the CAPTCHA (i.e., to determine w) automatically *without any human involvement*, rendering the CAPTCHA ineffective. While ad hoc measures can be used (e.g., having $\mathcal{S}_C$ return a random image instead of a decryption error), what is really needed is to use an encryption scheme that is CCA-secure.

References and Additional Reading

The modern computational approach to cryptography was initiated in a groundbreaking paper by Goldwasser and Micali [80]. That paper introduced the notion of semantic security, and showed how this goal could be achieved in the setting of public-key encryption (see Chapters 10 and 11). That paper also proposed the notion of indistinguishability (i.e., Definition 3.8), and showed that it implies semantic security. The converse was shown in [125]. The reader is referred to [76] for further discussion of semantic security.

Blum and Micali [37] introduced the notion of pseudorandom generators and proved their existence based on a specific number-theoretic assumption. In the same work, Blum and Micali also pointed out the connection between pseudorandom generators and private-key encryption as in Construction 3.17. The definition of pseudorandom generators given by Blum and Micali is different from the definition we use in this book (Definition 3.14); the latter definition originates in the work of Yao [179], who showed equivalence of the two formulations. Yao also showed constructions of pseudorandom generators based on general assumptions; we will explore this result in Chapter 7.

Formal definitions of security against chosen-plaintext attacks were given by Luby [115] and Bellare et al. [15]. Chosen-ciphertext attacks (in the context of public-key encryption) were first formally defined by Naor and Yung [129] and Rackoff and Simon [147], and were considered also in [61] and [15]. See [101] for other notions of security for private-key encryption.

Pseudorandom functions were defined and constructed by Goldreich et al. [78], and their application to encryption was demonstrated in subsequent work by the same authors [77]. Pseudorandom permutations and strong pseudorandom permutations were studied by Luby and Rackoff [116]. A proof of Proposition 3.27 (the "switching lemma") can be found in [24]. Block ciphers had been used for many years before they began to be studied in the theoretical sense initiated by the above works. Practical stream ciphers and block ciphers are studied in detail in Chapter 6.

The ECB, CBC, and OFB modes of operation (as well as CFB, a mode of operation not covered here) were standardized along with the DES block cipher [130]. CTR mode was standardized by NIST in 2001. CBC and CTR mode were proven CPA-secure in [15]. For more recent modes of operation, see `http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html`.

The attack on chained CBC was first described by Rogaway (unpublished), and was used to attack SSL/TLS in the so-called "BEAST attack" by Rizzo and Duong. The padding-oracle attack we describe here originated in the work of Vaudenay [171].

Exercises

3.1 Prove Proposition 3.6.

3.2 Prove that Definition 3.8 cannot be satisfied if Π can encrypt arbitrary-length messages and the adversary is *not* restricted to output equal-length messages in experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$.

> **Hint:** Let $q(n)$ be a polynomial upper-bound on the length of the ciphertext when Π is used to encrypt a single bit. Then consider an adversary who outputs $m_0 \in \{0,1\}$ and a uniform $m_1 \in \{0,1\}^{q(n)+2}$.

3.3 Say $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is such that for $k \in \{0,1\}^n$, algorithm Enc_k is only defined for messages of length at most $\ell(n)$ (for some polynomial ℓ). Construct a scheme satisfying Definition 3.8 even when the adversary is *not* restricted to outputting equal-length messages in $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$.

3.4 Prove the equivalence of Definition 3.8 and Definition 3.9.

3.5 Let $|G(s)| = \ell(|s|)$ for some ℓ. Consider the following experiment:

The PRG indistinguishability experiment $\mathsf{PRG}_{\mathcal{A},G}(n)$:

(a) A uniform bit $b \in \{0,1\}$ is chosen. If $b = 0$ then choose a uniform $r \in \{0,1\}^{\ell(n)}$; if $b = 1$ then choose a uniform $s \in \{0,1\}^n$ and set $r := G(s)$.

(b) The adversary $\mathcal{A}$ is given r, and outputs a bit b'.

(c) The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Provide a definition of a pseudorandom generator based on this experiment, and prove that your definition is equivalent to Definition 3.14. (That is, show that G satisfies your definition if and only if it satisfies Definition 3.14.)

3.6 Let G be a pseudorandom generator with expansion factor $\ell(n) > 2n$. In each of the following cases, say whether G' is necessarily a pseudorandom generator. If yes, give a proof; if not, show a counterexample.

(a) Define $G'(s) \stackrel{\text{def}}{=} G(s_1 \cdots s_{\lfloor n/2 \rfloor})$, where $s = s_1 \cdots s_n$.

(b) Define $G'(s) \stackrel{\text{def}}{=} G\left(0^{|s|} \| s\right)$.

(c) Define $G'(s) \stackrel{\text{def}}{=} G(s) \,\|\, G(s+1)$.

3.7 Prove the converse of Theorem 3.18. Namely, show that if G is not a pseudorandom generator then Construction 3.17 does not have indistinguishable encryptions in the presence of an eavesdropper.

3.8 (a) Define a notion of indistinguishability for the encryption of multiple *distinct* messages, in which a scheme need not hide whether the same message is encrypted twice.

(b) Show that Construction 3.17 does not satisfy your definition.

(c) Give a construction of a *deterministic* (stateless) encryption scheme that satisfies your definition.

3.9 Prove *unconditionally* the existence of a pseudorandom function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ with $\ell_{key}(n) = n$ and $\ell_{in}(n) = O(\log n)$.

Hint: Implement a uniform function with logarithmic input length.

3.10 Let F be a length-preserving pseudorandom function. For the following constructions of a keyed function $F' : \{0,1\}^n \times \{0,1\}^{n-1} \to \{0,1\}^{2n}$, state whether F' is a pseudorandom function. If yes, prove it; if not, show an attack.

(a) $F'_k(x) \stackrel{\text{def}}{=} F_k(0\|x) \,\|\, F_k(1\|x)$.

(b) $F'_k(x) \stackrel{\text{def}}{=} F_k(0\|x) \,\|\, F_k(x\|1)$.

3.11 Assuming the existence of pseudorandom functions, prove that there is an encryption scheme that has indistinguishable multiple encryptions in the presence of an eavesdropper (i.e., satisfies Definition 3.19), but is not CPA-secure (i.e., does not satisfy Definition 3.22).

Hint: The scheme need not be "natural." You will need to use the fact that in a chosen-plaintext attack the adversary can choose its queries to the encryption oracle *adaptively*.

3.12 Let F be a keyed function and consider the following experiment:

The PRF indistinguishability experiment $\mathsf{PRF}_{\mathcal{A},F}(n)$**:**

(a) A uniform bit $b \in \{0,1\}$ *is chosen. If* $b = 1$ *then choose uniform* $k \in \{0,1\}^n$.

(b) $\mathcal{A}$ *is given* 1^n *for input. If* $b = 0$ *then* $\mathcal{A}$ *is given access to a uniform function* $f \in \mathsf{Func}_n$. *If* $b = 1$ *then* $\mathcal{A}$ *is instead given access to* $F_k(\cdot)$.

(c) $\mathcal{A}$ *outputs a bit* b'.

(d) The output of the experiment is defined to be 1 *if* $b' = b$, *and* 0 *otherwise.*

Define pseudorandom functions using this experiment, and prove that your definition is equivalent to Definition 3.25.

3.13 Consider the following keyed function F: For security parameter n, the key is an $n \times n$ boolean matrix A and an n-bit boolean vector b. Define $F_{A,b} : \{0,1\}^n \to \{0,1\}^n$ by $F_{A,b}(x) \stackrel{\text{def}}{=} Ax + b$, where all operations are done modulo 2. Show that F is not a pseudorandom function.

3.14 Prove that if F is a length-preserving pseudorandom function, then $G(s) \overset{\text{def}}{=} F_s(1)\|F_s(2)\|\cdots\|F_s(\ell)$ is a pseudorandom generator with expansion factor $\ell \cdot n$.

3.15 Define a notion of perfect secrecy under a chosen-plaintext attack by adapting Definition 3.22. Show that the definition cannot be achieved.

3.16 Prove Proposition 3.27.

Hint: Use the results of Appendix A.4.

3.17 Assume pseudorandom permutations exist. Show that there exists a function F' that is a pseudorandom permutation but is *not* a strong pseudorandom permutation.

Hint: Construct F' such that $F'_k(k) = 0^{|k|}$.

3.18 Let F be a pseudorandom permutation, and define a fixed-length encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ as follows: On input $m \in \{0,1\}^{n/2}$ and key $k \in \{0,1\}^n$, algorithm Enc chooses a uniform string $r \in \{0,1\}^{n/2}$ of length $n/2$ and computes $c := F_k(r\|m)$.

Show how to decrypt, and prove that this scheme is CPA-secure for messages of length $n/2$. (If you are looking for a real challenge, prove that this scheme is CCA-secure if F is a *strong* pseudorandom permutation.)

3.19 Let F be a pseudorandom function and G be a pseudorandom generator with expansion factor $\ell(n) = n+1$. For each of the following encryption schemes, state whether the scheme has indistinguishable encryptions in the presence of an eavesdropper and whether it is CPA-secure. (In each case, the shared key is a uniform $k \in \{0,1\}^n$.) Explain your answer.

(a) To encrypt $m \in \{0,1\}^{n+1}$, choose uniform $r \in \{0,1\}^n$ and output the ciphertext $\langle r, G(r) \oplus m\rangle$.

(b) To encrypt $m \in \{0,1\}^n$, output the ciphertext $m \oplus F_k(0^n)$.

(c) To encrypt $m \in \{0,1\}^{2n}$, parse m as $m_1\|m_2$ with $|m_1| = |m_2|$, then choose uniform $r \in \{0,1\}^n$ and send $\langle r,\, m_1 \oplus F_k(r),\, m_2 \oplus F_k(r+1)\rangle$.

3.20 Consider a stateful variant of CBC-mode encryption where the sender simply increments the IV by 1 each time a message is encrypted (rather than choosing IV at random each time). Show that the resulting scheme is *not* CPA-secure.

3.21 What is the effect of a single-bit error in the ciphertext when using the CBC, OFB, and CTR modes of operation?

3.22 What is the effect of a dropped ciphertext block (e.g., if the trabsmitted ciphertext $c_1, c_2, c_3, \ldots$ is received as $c_1, c_3, \ldots$) when using the CBC, OFB, and CTR modes of operation?

3.23 Say CBC-mode encryption is used with a block cipher having a 256-bit key and 128-bit block length to encrypt a 1024-bit message. What is the length of the resulting ciphertext?

3.24 Give the details of the proof by reduction for Equation (3.12).

3.25 Let F be a pseudorandom function such that for $k \in \{0,1\}^n$ the function F_k maps $\ell_{in}(n)$-bit inputs to $\ell_{out}(n)$-bit outputs.

(a) Consider implementing CTR-mode encryption using F. For which functions ℓ_{in}, ℓ_{out} is the resulting encryption scheme CPA-secure?

(b) Consider implementing CTR-mode encryption using F, but only for *fixed-length* messages of length $\ell(n)$ (which is an integer multiple of $\ell_{out}(n)$). For which $\ell_{in}, \ell_{out}, \ell$ does the scheme have indistinguishable encryptions in the presence of an eavesdropper?

3.26 For any function $g : \{0,1\}^n \to \{0,1\}^n$, define $g^{\$}(\cdot)$ to be a *probabilistic* oracle that, on input 1^n, chooses uniform $r \in \{0,1\}^n$ and returns $\langle r, g(r)\rangle$. A keyed function F is a *weak pseudorandom function* if for all PPT algorithms D, there exists a negligible function negl such that:

$$\left| \Pr[D^{F_k^{\$}(\cdot)}(1^n) = 1] - \Pr[D^{f^{\$}(\cdot)}(1^n) = 1] \right| \leq \mathsf{negl}(n),$$

where $k \in \{0,1\}^n$ and $f \in \mathsf{Func}_n$ are chosen uniformly.

(a) Prove that if F is pseudorandom then it is weakly pseudorandom.

(b) Let F' be a pseudorandom function, and define

$$F_k(x) \overset{\text{def}}{=} \begin{cases} F'_k(x) & \text{if } x \text{ is even} \\ F'_k(x+1) & \text{if } x \text{ is odd.} \end{cases}$$

Prove that F is weakly pseudorandom, but *not* pseudorandom.

(c) Is CTR-mode encryption using a weak pseudorandom function necessarily CPA-secure? Does it necessarily have indistinguishable encryptions in the presence of an eavesdropper? Prove your answers.

(d) Prove that Construction 3.30 is CPA-secure if F is a weak pseudorandom function.

3.27 Let F be a pseudorandom permutation. Consider the mode of operation in which a uniform value $\mathsf{ctr} \in \{0,1\}^n$ is chosen, and the ith ciphertext block c_i is computed as $c_i := F_k(\mathsf{ctr} + i + m_i)$. Show that this scheme does not have indistinguishable encryptions in the presence of an eavesdropper.

3.28 Show that the CBC, OFB, and CTR modes of operation do not yield CCA-secure encryption schemes (regardless of F).

3.29 Let $\Pi_1 = (\mathsf{Enc}_1, \mathsf{Dec}_1)$ and $\Pi_2 = (\mathsf{Enc}_2, \mathsf{Dec}_2)$ be two encryption schemes for which it is known that at least one is CPA-secure (but you don't know which one). Show how to construct an encryption scheme Π that is guaranteed to be CPA-secure as long as at least one of Π_1 or Π_2 is CPA-secure. Provide a full proof of your solution.

Hint: Generate two plaintext messages from the original plaintext so that knowledge of either one reveals nothing about the original plaintext, but knowledge of both enables the original plaintext to be computed.

3.30 Write pseudocode for obtaining the entire plaintext via a padding-oracle attack on CBC-mode encryption using PKCS #5 padding, as described in the text.

3.31 Describe a padding-oracle attack on CTR-mode encryption (assuming PKCS #5 padding is used to pad messages to a multiple of the block length before encrypting).